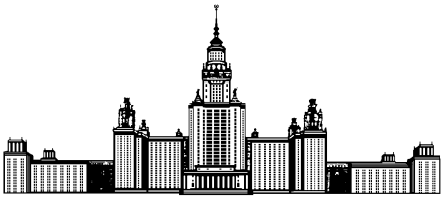

Московский государственный университет имени М. В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра математических методов прогнозирования

Каюмов Эмиль Марселевич

Сборка кубика Рубика с помощью обучения с подкреплением

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Научный руководитель:

д.ф-м.н., профессор

А. Г. Дьяконов

Москва, 2019

Содержание

Аннотация

В последние годы активно развивается обучение с подкреплением — область машинного обучения, в которой агент обучается не на готовых примерах, а в результате взаимодействия со средой. Достигнуты впечатляющие результаты в играх Atari в 2013 году, AlphaZero в го и шахматах в 2016-2018 годах, OpenAI Five в игре Dota2 в 2019 году. Агент, ориентируясь только на правила игры, на собственном опыте научился играть лучше человека.

В данной работе сделана попытка применить методы обучения с подкрепления к задаче сборки кубика Рубика. Кубик Рубика с точки зрения обучения с подкреплением — среда с большим количеством состояний и редким обратным положительным сигналом. Предложены способы применения Deep Q-learning к сборке Кубика Рубика и использования поиска по дереву для улучшения результата жадной стратегии. Показана успешность решения для среды с низкой сложностью без использования человеческих знаний, исследованы некоторые аспекты настройки агентов.

1 Введение

Обучение с подкреплением — активно развивающаяся в последнее время область машинного обучения. Агент взаимодействует с окружением и обучается на основе обратной связи от окружения. Развитие техник обучения нейронных сетей и вычислительных мощностей дало толчок в развитии обучения с подкреплением.

Проводятся попытки применить обучение с подкреплением в различных областях. В 2013 году показаны впечатляющие результаты на играх Atari [1]. В 2016 году AlphaGo выиграл чемпиона мира в игру Go [2]. OpenAI Five в 2019 выиграл большинство матчей у любителей и профессиональных игроков в Dota2.

Кубик Рубика — известная многим механическая головоломка, представляющая собой куб с вращающимися гранями разных цветов. Цель игрока состоит в получении кубика, у которого каждая грань будет окрашена в один цвет. Известны алгоритмы для решения головоломки на основе алгебраических [3] и переборных подходов. Кубик Рубика является примером задачи комбинаторной оптимизации с большим количеством возможных состояний, поэтому применение новых техник для решения задачи может помочь решению других задач.

Существуют попытки применения обучения с подкреплением к задаче сборке кубика Рубика. Некоторые их них опираются на особенности кубика Рубика, что мешает переносу решений на другие задачи. В некоторых исследованиях получены успешные результаты для кубиков, находящихся на небольшом расстоянии от финального состояния.

В данной работе предложен метод обучения нейросетевой Q-функции для сборки кубика Рубика, исследованы некоторые аспекты использования фиктивной награды и описания среды для обучения агента. Предложены и экспериментально проверены два мета-алгоритма для применения обученной Q-функции на основе поиска по дереву с сэмплированием действий и на основе алгоритма Upper Confidence Bound. Экспериментально показано, что мета-алгоритмы позволяют значительно улучшить качество решения задачи жадной стратегией.

2 Постановка задачи

2.1 Задача обучения с подкреплением

В отличие от классической задачи обучения с учителем в обучении с подкреплением настройка алгоритма происходит не на основе набора данных, полученного внешним процессом, а на основе взаимодействия в рамках достижения некоторой цели.

Агентом в обучении с подкреплением называют единицу, которая обучается и принимает решения. Агент взаимодействует с окружением или средой. Взаимодействие представляется в виде последовательности дискретных шагов $t = 0, 1, 2, \ldots$ в рамках одного эпизода. На шаге t агент в состоянии среды s_t совершает действие a_t, в ответ на которое получает от окружения новое состояние среды s_{t+1} и числовую награду за совершённое действие r_{t+1} (подкрепление). Агент максимизирует суммарную награду в процессе взаимодействия с окружением. Схематично взаимодействие агента с окружением изображено на рис. (1).

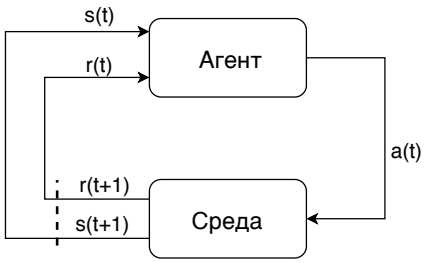

Рис. 1: Один шаг взаимодействия агента с окружением

Обучение — процесс поиска стратегии действий агента $\pi(s_t)$, которая максимизирует суммарную награду $\sum_t r_t$. Агенту неизвестны истинные функции перехода $p(s_{t+1}|a_t, s_t)$ и награды $p(r_{t+1}|a_t, s_t)$.

2.2 Кубик Рубика

Кубик Рубика — механическая головоломка, изобретённая в 1974 году Эрнё Рубиком. В классической версии головоломка представляет собой пластмассовый куб

3x3x3 с 54 видимыми цветными наклейками. Грани большого куба способны вращаться вокруг 3 внутренних осей куба. Каждая из шести граней состоит из девяти квадратов и окрашена в один из шести цветов. Повороты граней позволяют переупорядочить цветные квадраты множеством различных способов. Задача игрока заключается в том, чтобы «собрать» или «решить» кубик Рубика»: поворачивая грани куба, вернуть его в первоначальное состояние, когда каждая из граней состоит из квадратов одного цвета.

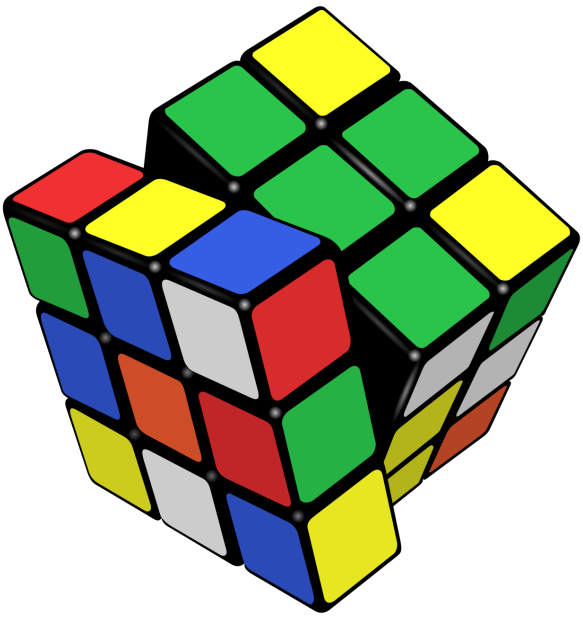

Рис. 2: Классический кубик Рубика

Число различных состояний Кубика составляет $43 * 10^{18}$ (без учёта ориентации центральных неподвижных кубиков). За счёт большого числа состояний Кубик является сложной для решения головоломкой. Сущестуют различные алгоритмы по сборке Кубика. Алгоритмы, ориентированные на человека и поэтому простоту запоминания, являются неоптимальными с точки зрения количества совершаемых действий. С 1981 года ведутся исследования по определению минимального количества ходов, необходимого для сборки кубика из любой позиции. В 2014 году было показано, что кубик может быть собран за 26 ходов (если считать только действия «поворот на 90 градусов») и за 20 ходов (если также считать за одно действие повороты на

90 или 180 градусов) [4], [5]. Существуют также машинные алгоритмы для сборки Кубика, например, алгоритмы Коцембы или Корфа, ориентированные на сборку за минимальное число ходов [3].

Всего у кубика 6 граней, каждую из которых можно вращать по и против часовой стрелки. Таким образом, взаимодействие с кубиком заключается в 12 возможных действиях.

С точки зрения обучения с подкреплением кубик описывается с помощью цветов на гранях, функция перехода между состояниями детерминированная (можно получить только одно состояние кубика при применении конкретного действия в конкретном состоянии), награда равна нулю при переходе во все состояния кроме финального.

Целью данной работы является применение методов обучения с подкреплением к задаче сборки кубика Рубика.

3 Подходы

В этом разделе рассмотрим существующие подходы обучения с подкреплением к задаче сборки кубика Рубика и техники обучения с подкреплением, используемые в рамках данной работы.

3.1 Известные подходы

Существует несколько попыток применения обучения с подкреплением к задаче сборки кубика Рубика.

Есть попытки применения подходов обучения с учителем в статье [6] и нескольких исследованиях [7], [8]. Однако обучение с учителем противоречит идее обучения с подкреплением. В рамках данной работы стояла цель не использовать особенности среды, позволяющие генерировать выборку и сводить задачу к обучению с учителем.

В работе [9] исследуется поиск по дереву с помощью нескольких функций оценки состояний. Обучение этих функций производится на сгенерированных случайным блужданием выборках и использует механизм хранения паттернов кубика в базе

данных. Уделяется большее внимание оптимальности решения, чем доле успешных сборок.

В [10] исследуется применение temporal difference техник на примере Кубиков сложности до 5 шагов от финального состояния, с которыми у большинства статей и исследований проблем нет.

В незавершённом на момент написания данной работы исследовании [11] используется поиск по дереву в процессе обучения основной сети и на этапе применения алгоритма. Качество сборки сетью с жадным алгоритмом применения близко к результатам данной работы, а применение поиска по дереву также улучшает общее качество. Используется более сложная архитектура решения.

Наиболее близкий к успешному решению задачи результат достигнут в статье [12]. Предложен подход по обучению сети Autodidactic Iteration, который явно использует особенности кубика. При применении сети используется поиск по дереву. Это позволило собирать кубик Рубика из произвольной позиции, что не удалось в рамках остальных исследований, однако сборка кубика занимает на порядок больше времени, чем в данной работе. Недостаточная подробность в реализации описанного подхода не позволяют воспроизвести результаты работы.

3.2 Deep Q-learning

Одним из базовых методов обучения с подкреплением является метод Q-обучения, предложенный в 1992 году [13]. В этом методе используется понятие функции ценности пары состояние и действие $Q(s, a)$, которая определена для некоторой стратегии π и является математическим ожиданием дисконтированного вознаграждения $Q^\pi(s, a) = \mathbb{E}_{\pi,s,a}(R_t|s_t = s, a_t = a)$, $R_t = \sum_{i=1}^{t} \gamma^{t-i} r_i$. Детерминированная стратегия агента восстанавливается по функции ценности: $\arg\max_a \pi(a|s) = \arg\max_a Q(s, a)$.

Алгоритм Q-обучения восстанавливает оптимальную функцию ценности. Для этого используется факт, что функцию можно выразить через саму себя. Для оптимальной функции ценности:

$$Q^*(s_t, a_t) = \mathbb{E}(r_{t+1} + \gamma \max_a Q^*(s_{t+1}, a))$$

В случае конечного пространства состояний S небольшой размерности саму Q-функцию можно хранить в виде таблицы и обновлять с помощью следующего соотношения:

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a))$$

В случае пространств большой размерности хранение и обновление функции в виде таблицы не представляется возможным, поэтому вместо таблицы строится аппроксимация Q-функции. Например, это делается с помощью нейронных сетей (Deep Q-learning или DQN) [1]. Для этого существует два подхода. В первом из них на вход сети подаётся описание среды, сеть выдаёт одновременно значения Q-функции для всех возможных действий. Во втором подходе сеть оценивает значение Q-функции для конкретного действия, которое подаётся на вход сети вместе с состоянием среды. В данной работе будем использовать первый подход.

В процессе обучения Q-функции для агента используется механизм Experience Replay [1]. Его суть заключается в том, что вся история взаимодействия агента с окружением сохраняется в память ограниченной длины по стратегии FIFO. Каждый батч для обновления весов сети берётся случайным образом из этой памяти. Ограничение на объём памяти позволяет использовать только свежие примеры для обучения, так как взаимодействие агента с окружением в процессе обучения агента меняется. Использование же примеров только с последнего эпизода взаимодействия сказывается на процессе обучения негативно, так как примеры в рамках одного батча будут скоррелированы.

Если при обучении Q-функции в виде нейронной сети подсчёт градиентов производить с помощью этой же Q-функции, то Q-функция будет завышать значения. Самый простой способ обойти эту проблему — использовать для подсчёта градиентов зафиксированную Q-сеть (target network), веса которой периодически синхронизируются с обучаемой сетью. Альтернативный подход заключается в использование второй Q-сети для вычисления действия, по котором достигается максимум Q-функции [14].

При применении DQN на этапе обучения сети необходимо одновременно улучшать Q-функцию с точки зрения точности предсказаний и с точки зрения иссле-

дования новых состояний. Существует две базовых стратегии исследования среды. ϵ-жадная стратегия с вероятностью ϵ выбирает случайное действие, с вероятностью $1-\epsilon$ — действие, максимизирующее Q-функцию. Стратегия исследования по Больцману сэмплирует каждое действие с вероятностями, полученными после применения softmax-преобразования к Q-значениям.

Полная версия используемого алгоритма обучения Q-функции описана на схеме 3.1.

Algorithm 3.1 Deep Q-learning with experience replay and target network

Параметры: N, M, S, T, U
Инициализировать experience replay память D размера N
Инициализировать Q_{main} случайными весами
Инициализировать Q_{target} весами Q_{main}
для эпизода $e = 1, M$
  Инициализировать состояние s_0 кубика Рубика заданной сложности S
  для шага $t = 1, T$
    Выбрать действие a_t из распределения, полученного по Q_{main}
    Сделать шаг и получить новое состоение s_{t+1} и награду r_{t+1}
    Добавить в память D значения $(s_t, a_t, r_{t+1}, s_{t+1})$
    если s_{t+1} — финальное состояние **то**
      Выйти из цикла
  Получить очередной батч из памяти D
  Рассчитать целевые значения через Q_{target}:
  $y_j = r_j + \gamma \max_a Q_{target}(s_{j+1}, a)$
  Сделать градиентный шаг обновления весов Q_{main} по квадратичной ошибке:
  $(y_j - Q_{main}(s_j, a_j))^2$
  если прошло U эпизодов с момента обновления Q_{target} **то**
    Обновить Q_{target} текущими весами Q_{main}

3.3 Поиск по дереву

Обычно на этапе применения используется жадная стратегия выбора действия, максимизирующая Q-функцию на каждом шаге взаимодействия. Однако детерминированные среды, такие как, например, кубик Рубика, допускают «откаты», то есть после применения нескольких действий можно вернуться в исходное состояние и применить другую последовательность действий. Для улучшения качества в таких средах применяются более сложные по сравнению с жадной стратегии применения обученных алгоритмов.

Большой класс таких подходов основан на поиске по дереву. Представим каждое состояние окружения в качестве вершины графа, в котором направленные рёбра будут означать переход из одного состояния в другое с помощью некоторого действия. Так как взаимодействие с окружением начинается с некоторого начального состояния, то можно построить дерево, корнем которого будет начальное состояние. От корня при использовании всех возможных действий можно перейти к другим состояниям окружения (новым вершинам), из которых можно переходить в следующие состояния (состояния в дереве могут повторяться). Конечные состояния окружения будут листьями дерева.

Если знать местоположение листьев (финальных состояний), то можно построить путь до них и, таким образом, решить задачу. Однако ширина дерева растёт экспоненциально количеству возможных действий, поэтому построить всё дерево целиком для среды оказывается чаще всего невозможным. Из-за этого применяются различные техники по эффективному исследованию состояний в дереве. Семейство методов по поиску оптимального пути с исследованием состояний блужданием и использованием полученных результатов для уточнения поиска называют методами Монте-Карло для поиска в дереве [15].

Один из популярных подходов, использованных в AlphaZero для шахмат и го [2], [16] [17] использует модификацию алгоритма Upper Confidence Bound (UCB1) [18]. В UCB1 на каждом шаге выбирается действие, максимизирующее следующую величину:

$$a_i = \arg\max_i \left(v_i + \sqrt{\frac{2N}{N_i}} \right),$$

где v_i — средний выигрыш по всем симуляциям в вершине i, в которую можно попасть с помощью действия a_i, N — количество посещений текущей вершины, из которой совершается действие, N_i — количество посещений вершины i.

В модификациях UCB1 могут использоваться априорные вероятности на действия, полученные с помощью некоторого алгоритма и другие способы вычисления средней оценки или учёта количества посещений вершин.

В данной работе предложено два подхода к применению обученной Q-функции при сборке кубика на основе поиска по дереву.

В первом подходе (naive sampling) выбор каждого следующего действия производится сэмплированием с вероятностями, полученными из Q-функции после softmax-преобразования:

$$a_i \sim \pi_a = \frac{\exp Q(s,a)}{\sum_j \exp Q(s,a_j)}$$

При достижении листовой вершины, не являющейся финальным состоянием, создаются все её потомки, то есть вычисляются вероятности для переходов к следующим вершинам. Каждая симуляция представляет собой спуск из корневой вершины до листа и создание потомков листовой вершины. Работа алгоритма останавливается при нахождении финального состояния (успешное решение задачи) или после достижения лимита на число симуляций.

Полная версия алгоритма описана на схеме 3.2.

Algorithm 3.2 Поиск по дереву с сэмплированием действий

Параметры: Q_{main}, M, s_0
Инициализировать корень *root* начальным состоянием s_0 кубика
для итерации $i = 1, M$
  Инициализировать *node* корнем дерева
  пока *node* — не листовая вершина
    Выбрать действие a из распределения на действия в вершине *node*
    Перейти в новую вершину *node* из текущей с помощью действия a
  Создать всех потомков вершины *node* с априорными вероятностями действий p_a, полученных с помощью Q_{main}
  если среди потомков *node* есть финальное состояние кубика **то**
    Выйти и вернуть последовательность действий

Второй подход (argmax with exploration) представляет собой упрощение UCB1 с использованием априорного распределения. Выбор каждого следующего действия производится на основе вероятностей, полученных из Q-функции после softmax-преобразования, дисконтированных числом посещений каждой вершины:

$$a_i = \arg\max_a \pi_a \sqrt{\frac{\log N}{N_a}} = \arg\max_a \frac{\exp Q(s,a)}{\sum_j \exp Q(s,a_j)} \sqrt{\frac{\log N}{N_a}}$$

Вместо сэмплирования действий берётся действие с максимальным значением оценки на «интересность посещения», исследование новых состояний происходит за счёт дисконтирования вершин, которые были посещены большое число раз. Аналогично первому подходу при достижении листовой вершины, не являющейся финаль-

ным состоянием, создаются все её потомки, то есть вычисляются вероятности для переходов к следующим вершинам. Каждая симуляция представляет собой спуск из корневой вершины до листа и создание потомков листовой вершины. Работа алгоритма останавливается при нахождении финального состояния (успешное решение задачи) или после достижения лимита на число симуляций.

Полная версия алгоритма описана на схеме 3.3.

Algorithm 3.3 Поиск по дереву на основе UCB1

Параметры: Q_{main}, M, s_0
Инициализировать корень *root* начальным состоянием s_0 кубика, количеством посещений $N = 1$ и априорными вероятностями на действия π_a, полученными с помощью Q_{main}
для итерации $i = 1, M$
  Инициализировать *node* корнем дерева
  пока *node* — не листовая вершина
    Выбрать действие a, максимизирующее функцию $\pi_a \sqrt{\frac{\log N}{N_a}}$
    Перейти в новую вершину *node* из текущей с помощью действия a
  Создать всех потомков вершины *node* с количеством посещений $N = 0$, априорными вероятностями действий π_a, полученные с помощью Q_{main}
  если среди потомков *node* есть финальное состояние кубика **то**
    Выйти и вернуть последовательность действий
  пока *node* — не корневая вершина
    Увеличить счётчик количества посещений $N = N + 1$
    Перейти в родительскую вершину

4 Эксперименты

В этом разделе опишем конкретнее используемые модели, эксперименты и результаты.

В экспериментах использовалась библиотека PyCuber для симуляции кубика Рубика и Pytorch для обучения сети и реализации DQN. Код экспериментов опубликован [19].

Основной функционал качества «доля успешных сборок кубика Рубика». В случае с тестированием на этапе обучения DQN накладывался лимит в 100 шагов в рамках одного эпизода. При применении дерева поиска накладывался лимит в 500 итераций спуска по дереву до листьев.

В качестве сети для Q-функции использовалась полносвязная сеть с 3 скрытыми слоями (2048/512/128 нейронов) и ReLU в качестве функции активации. Коэффициент дисконтирования равен 0.9, веса target-сети обновлялись каждые 500 итераций, размер батча при обучении 512. В качестве метода оптимизации использовался Adam. Размер Experience Replay равен 1000000.

Кубик представляет собой сложную среду с большим количеством состояний, поэтому в экспериментах сложность среды уменьшалась. Начальное состояние среды задавалось как произвольное состояние в x шагов от успешной сборки Кубика. Увеличение x приводит к усложнению сборки Кубика. $x \approx 100$ соответствует произвольному состоянию Кубика (большие значения x не делают сборку сложнее).

4.1 Описание кубика

Описывать среду, то есть состояние Кубика можно двумя способами:

1. По цветам наклеек, то есть описывать цвета на каждой из граней. В таком случаев Кубик описывается с помощью бинарного вектора длины $6 \times 9 \times 6 = 324$.

2. По позициям кубиков, то есть описывать местоположение каждого конкретного кубика. В таком случаев Кубик описывается с помощью бинарного вектора длины $8 \times 8 \times 3 + 12 \times 12 \times 2 = 480$.

Первой подход требует от агента дополнительно решать задачу сопоставления цветов одного кубика (чтобы понять, где находится некоторый угловой кубик, надо найти три его наклейки и по ним определить местоположение).

В эксперименте по сравнению двух подходов к описанию среды было получено сравнимое качество у обоих подходов (рис. (3)). Из-за меньшей размерности входного вектора в дальнейшем будем использовать первый подход.

4.2 Фиктивная функция награды

Особенностью кубика Рубика как окружения является то, что сигнал несёт только финальное состояние. Естественной функцией награды является индикатор успеш-

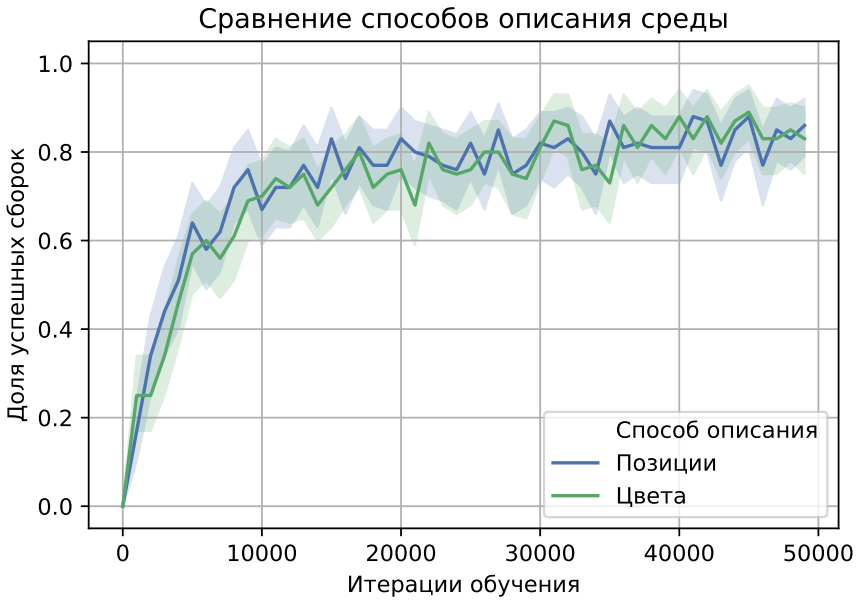

Рис. 3: Сравнение способов описания состояний через позиции кубиков и через цвета

ности сборки, однако с такой функцией сложно вести обучение модели из-за недостаточной обратной связи. Был предложен другой вариант функции награды: оценка состояния через долю наклеек (видимых сторон кубиков), совпадающих по цвету с наклейками собранного Кубика. Для стандартного Кубика это сумма $3 \times 3 \times 6 = 54$ индикаторов. Недостатком является то, что совпадение по цвету не означает нахождение на конкретной позиции нужного кубика. Награда после каждого шага равна разности оценок до и после шага.

С помощью фиктивной функции награды агент обучается быстрее, чем с помощью исходной разреженной награды, и достигает более высокого качества в 97% успешных сборок кубика против 92% (рис. (4)). Для дальнейших экспериментов используем агента, обученного с помощью фиктивной функции награды.

4.3 Сложность обучающего окружения

Для обучения агента использовались среды двух версий: сложности в 5 и 10 шагов до финального состояния.

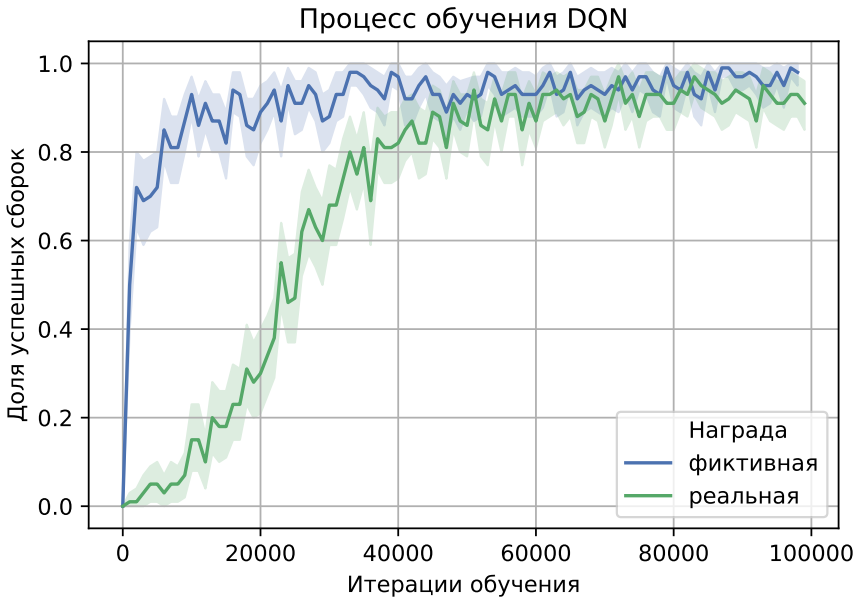

Рис. 4: Сравнение функций наград

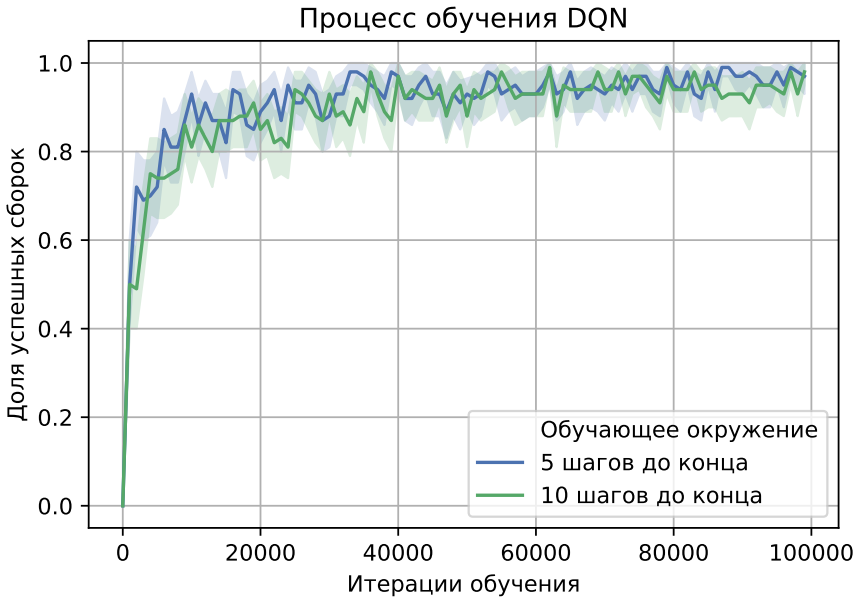

Рис. 5: Процесс обучения DQN для разных по сложности обучающих сред

На рис. (5) можно увидеть прогресс при обучении модели (в качестве тестового окружения выступает Кубик сложностью в 5 шагов). Сравнение двух версий DQN производилось со случайным агентом (выбирает действие из равномерного распределения) на окружениях с разной сложностью. Обе версии показали сравнимое ка-

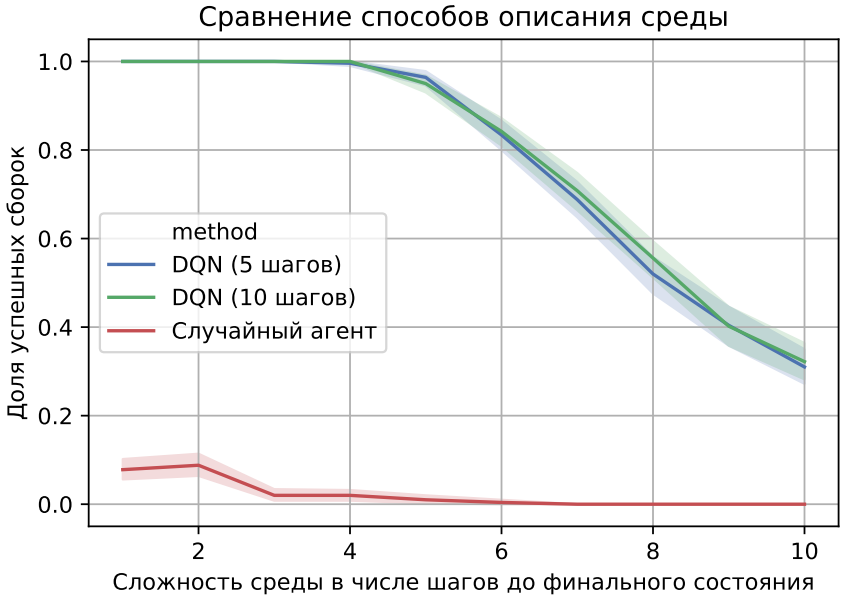

Рис. 6: Сравнение мета-алгоритмов для разной сложности сред

чество в 97-98% (рис. (6)). Дальнейшая работа велась на основе DQN, обученного на окружении сложности 5.

4.4 Применение дерева поиска

В качестве мета-алгоритма поверх обученной Q-функции было предложено два подхода на основе поиска по дереву: с сэмплированием действий и на основе UCB1.

Оба подхода сравнивались с жадной стратегией (greedy) выбора действия:

$$a_i = \arg\max_a \pi_a = \arg\max_a Q(s, a)$$

.

Для всех подходов использовалась одна и та же обученная Q-функция. Лимит на число итераций для двух первых мета-алгоритмов — 500, ограничение на время работы жадного подхода — 10 секунд. Количество попыток сборки Кубика — 500.

Применение мета-алгоритмов к окружениям разной сложности показало, что предложенные подходы показывают между собой сравнимое качество и показывают рост доли успешных сборок по сравнению с жадным алгоритмом от 6 до 23 п.п. в

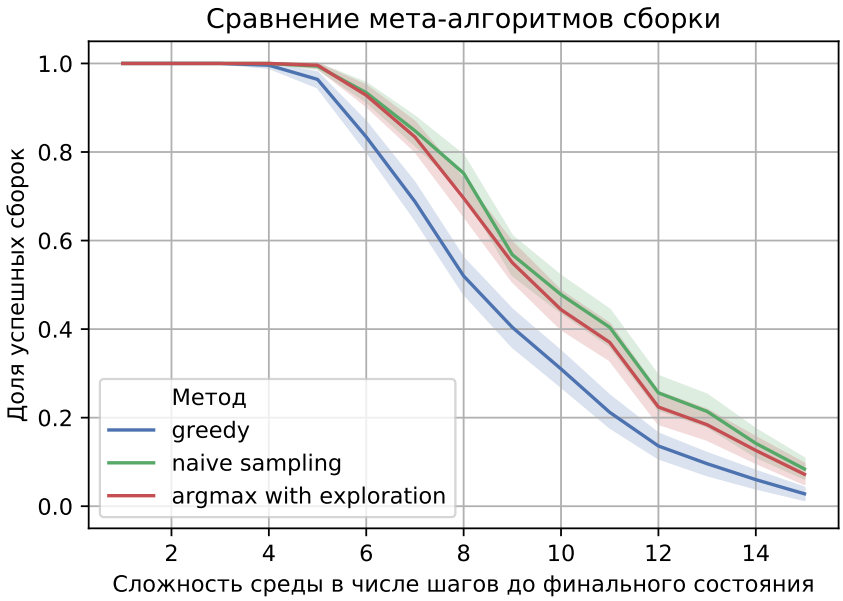

Рис. 7: Сравнение мета-алгоритмов для разной сложности сред

зависимости от сложности кубика. На окружении сложности в 10 шагов от финального состояния доля успешных сборок для первого подхода (naive sampling) составляет 47.8%, для второго подхода (argmax with exploration) — 44.4%.

4.5 Обсуждение и выводы

В результате экспериментов установлено следующее:

1. Способ описания Кубика через цвета и через позиции кубиков показывают сравнимое качество. То есть агенту не становится сложнее решать задачу сборки из-за дополнительной сложности в сопоставлении цветов и кубиков.

2. Фиктивная функция награды через долю совпадающих цветов с финальным состоянием позволяет агенту обучаться быстрее и показывает более высокое качество по сравнению с истинной наградой.

3. DQN-агенты обученные на окружениях с разной сложностью показывают сравнимое между собой качество на всех средах. Достигают 95-96% успешных сборок Кубика в 5 шагах от финального состояния и 31% в 10 шагах от финального состояния.

4. Предложенные мета-алгоритмы на основе поиска по дереву показывают прирост доли успешных сборок от 6 до 23 п.п. в зависимости от сложности кубика по сравнению с жадной стратегией применения DQN. Между собой подход с сэмпилрованием действий и выбором действия с максимальной оценкой показывают сравнимое качество и достигают 44-48% успешных сборок для Кубика в 10 шагах от финального состояния.

5 Заключение

В данной работе были рассмотрены подходы к применению обучения с подкреплением к задаче сборки кубика Рубика. Были предложены мета-алгоритмы на основе поиска по дереву и подход по обучению Deep Q-learning агента.

В работе предложен конкретный метод обучению нейросетевой Q-функции для сборки кубика Рубика, исследованы некоторые аспекты описания среды, фиктивной награды и используемой среды для обучения агента. Предложены и экспериментально проверены два мета-алгоритма для более эффективного использования обученной Q-функции на основе поиска по дереву с сэмплированием действий и на основе алгоритма Upper Confidence Bound. Экспериментально показано, что мета-алгоритмы позволяют значительно улучшить качество решения задачи жадной стратегией с помощью нейросетевой Q-функции.

Список литературы

[1] Playing Atari with Deep Reinforcement Learning / Volodymyr Mnih, Koray Kavukcuoglu, David Silver et al. // *CoRR*. — 2013.

[2] Mastering the game of Go with deep neural networks and tree search / David Silver, Aja Huang, Chris J. Maddison et al. // *Nature*. — 2016. — Vol. 529. — Pp. 484–489.

[3] *Kociemba Herbert*. Two-phase algorithm details.

[4] The diameter of the rubik's cube group is twenty / Tomas Rokicki, Herbert Kociemba, Morley Davidson, John Dethridge // *SIAM Review*. — 2014.

[5] *Rokicki Tomas*. God's number is 26 in the quarter-turn metric. — 2014.

[6] *Irpan Alexander*. Exploring Boosted Neural Nets for Rubik's Cube Solving. — 2016.

[7] How to train a simple convnet to solve a rubiks cube. — `https://github.com/jerpint/rubiks_cube_convnet`. — 2017.

[8] Trying to find God's algorithm on a Rubik's cube. — `https://github.com/unixpickle/godsalg`. — 2017.

[9] *Brunetto Robert*. Deep Heuristic-learning in the Rubik ' s Cube Domain : an Experimental Evaluation. — 2017.

[10] *Lichodzijewski Peter, Heywood Malcolm*. The Rubik Cube and GP Temporal Sequence Learning: An Initial Study // Genetic Programming Theory and Practice VIII / Ed. by Rick Riolo, Trent McConaghy, Ekaterina Vladislavleva. — New York, NY: Springer New York, 2011. — Pp. 35–54. — URL: `https://doi.org/10.1007/978-1-4419-7747-2_3`.

[11] Solving the puzzle cube with deep reinforcement learning and tree search. — `https://github.com/jasonrute/puzzle_cube`. — 2017.

[12] Solving the Rubik's Cube Without Human Knowledge / Stephen McAleer, Forest Agostinelli, Alexander Shmakov, Pierre Baldi // *arXiv preprint arXiv:1805.07470*. — 2018.

[13] *Watkins Christopher JCH, Dayan Peter.* Q-learning // *Machine learning.* — 1992.

[14] *Van Hasselt Hado, Guez Arthur, Silver David.* Deep Reinforcement Learning with Double Q-learning. — 2015. — 09.

[15] A Survey of Monte Carlo Tree Search Methods / Cameron Browne, Edward Powley, Daniel Whitehouse et al. // *IEEE Transactions on Computational Intelligence and AI in Games.* — 2012. — 03. — Vol. 4:1. — Pp. 1–43.

[16] Mastering the game of Go without human knowledge / David Silver, Julian Schrittwieser, Karen Simonyan et al. // *Nature.* — 2017. — 10. — Vol. 550. — Pp. 354–359.

[17] A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play / David Silver, Thomas Hubert, Julian Schrittwieser et al. // *Science.* — 2018. — Vol. 362, no. 6419. — Pp. 1140–1144. — URL: `https://science.sciencemag.org/content/362/6419/1140`.

[18] *Kocsis Levente, Szepesvari Csaba.* Bandit Based Monte-Carlo Planning. — 2006. — Pp. 282–293.

[19] Реализация и эксперименты. — `https://github.com/emilkayumov/rubiks-cube-reinforcement-learning`. — 2019.

[20] *Brunetto Robert, Trunda Otakar.* Deep heuristic-learning in the rubik's cube domain: an experimental evaluation // *CEUR Workshop Proceedings.* — 2017.