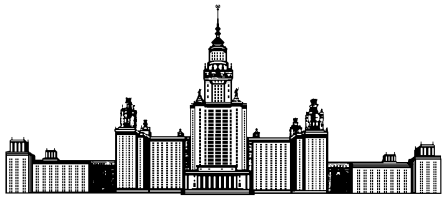

Московский государственный университет имени М. В. Ломоносова

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 2018, OpenAI Five в игре Dota2 в 2019. 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 сборке Кубика Рубика. Кубик Рубик с точки зрения обучения с подкреплением — среда с большим количеством состояний и редким обратным 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

Актуальность ...

В данной работе ...

План работы ...

# 2 Постановка задачи

## 2.1 Задача обучения с подкреплением

В отличие от классической задачи обучения с учителем в обучении с подкреплением обучение 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 (средой). Взаимодействие представляется в виде последовательности дискретных шагов $t = 0, 1, \ldots$ в рамках одного эпизода. На шаге $t$ агент 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).

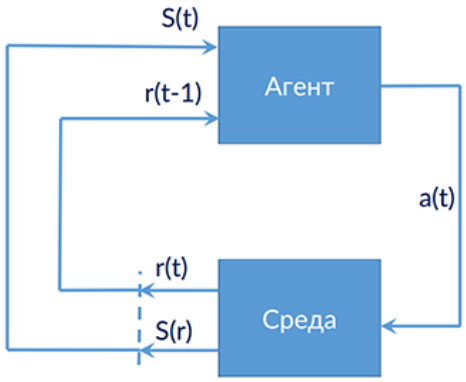


Рис. 1: Один шаг взаимодействия агента с окружением

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 $p(r_t|a_t, s_t)$.

## 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.

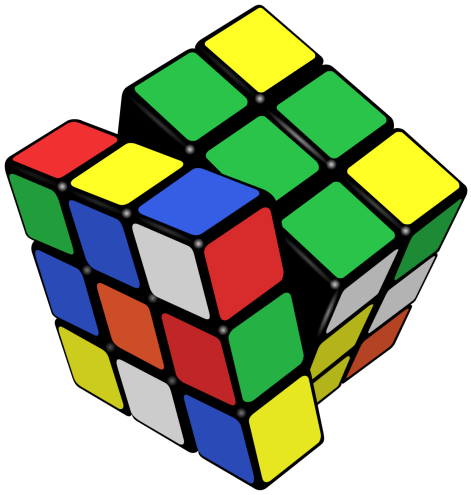

Рис. 2: Классический Кубик Рубика

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 соответственно 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) [1], [2]. 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. Будем называть Кубиком весь Кубик Рубик, кубиками — составные части большого Кубика, наклейками — грани кубиков.

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, используемые в рамках данной работы.

## 3.1 Известные подходы

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

IN PROGRESS

supervised-подходы в статьях

unpublished-репозитории с результатами

та самая статья preprint со сложно воспроизводимыми результатами

## 3.2 Deep Q-learning

Одним из базовых методов обучения с подкреплением является метод Q-обучения, предложенный в 1992 году [4]. В этом методе используется понятие функции ценности пары состояние и действие $Q(s, a)$, которая определена для некоторой стратегии $\pi$ и является математическим ожиданием дисконтированного вознаграждения $Q^{\pi}(s, a) = \mathbb{E}_{\pi,s,a}(R_t|s_t = s, a_t = a)$. 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 [5]. 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 рассматривать первый подход.

В процессе обучения Q-функции для агента используется механизм Experience Replay [5]. 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, веса которой периодически синхронизируются с обучаемой сетью. Альтернативный подход заключается в использование второй Q-сети для вычисления действия, по котором достигается максимум Q-функции [6].

При применении DQN на этапе обучения сети необходимо одновременно улучшать Q-функцию с точки зрения точности предсказаний и с точки зрения исследования новых состояний. Существует две базовых стратегии исследования среды. $\epsilon$-жадная стратегия с вероятности $\epsilon$ 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.

## 3.3 Поиск по дереву

Обученная Q-сеть или любой другой алгоритм обучения с подкреплением уже способны решать задачу. Обычно на этапе применения используется жадная стратегия выбора действия, например, максимизирующая Q-функцию на каждом шаге взаимодействия. Для улучшения качества применяются более сложные по сравнению с стратегии применения обученных алгоритмов.

Большой класс таких подходов основан на поиске по дерево. 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 состо-

яния, 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 сэмплированием для исследования и использования полученных результатов для уточнения поиска называют методами Монте-Карло для поиска в дереве [7].

Один из популярных подходов, использованных в AlphaZero для шахмат и го [8], [9] [10] использует модификацию алгоритма Upper Confidence Bound (UCB1) [11]. 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.

# 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 [12].

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).

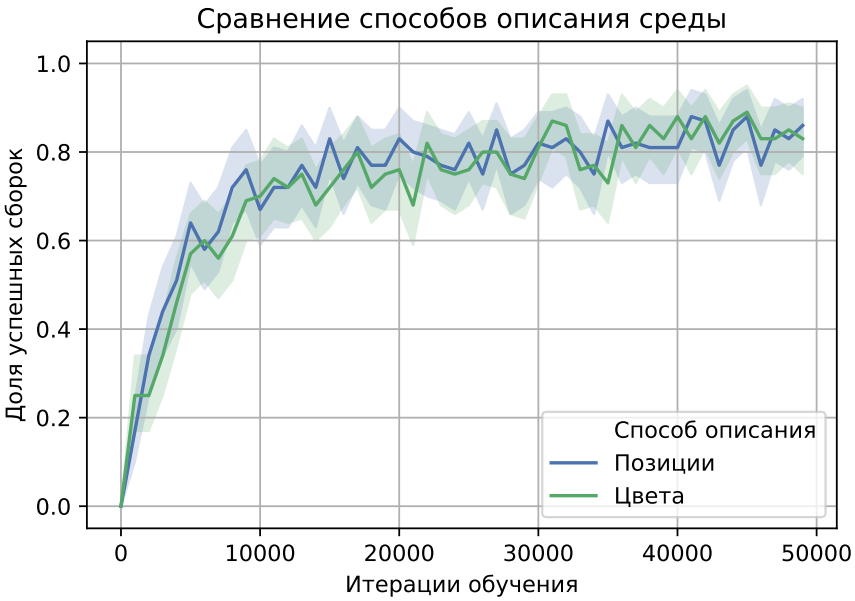


Рис. 3: Сравнение способов описания состояний через позиции кубиков и через цвета

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 окружение является то, что сигнал несёт только финальное состояние. Естественной функцией награды является индикатор успешности 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 сравнимого качества в 96-97%

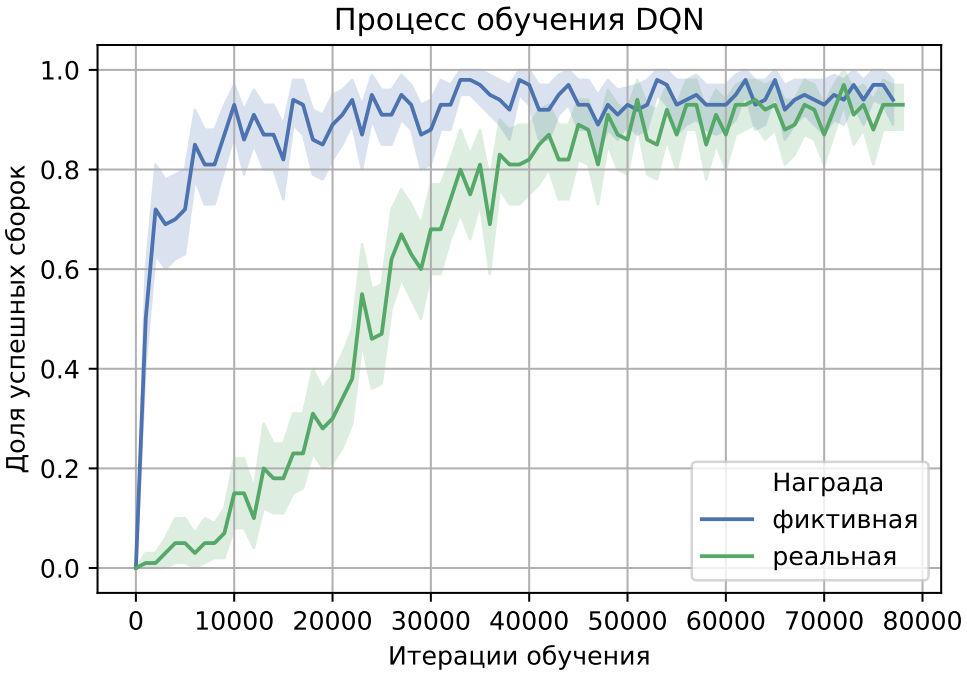


Рис. 4: Сравнение функций наград

успешных сборок кубика (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.

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

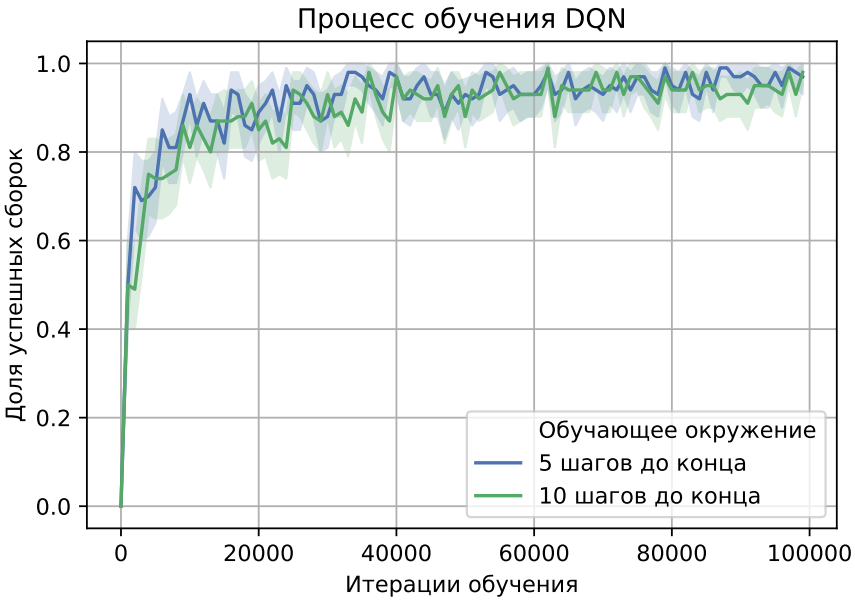


Рис. 5: Процесс обучения DQN для разных по сложности обучающих сред

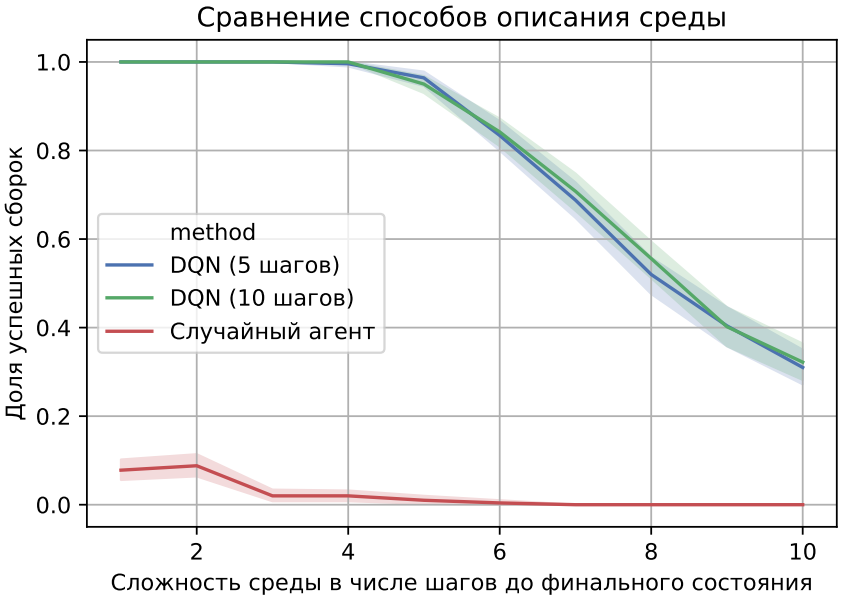


Рис. 6: Сравнение мета-алгоритмов для разной сложности сред

## 4.4 Применение дерева поиска

В качестве мета-алгоритма поверх обученной Q-функции было предложено два подхода 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.

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 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.

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.

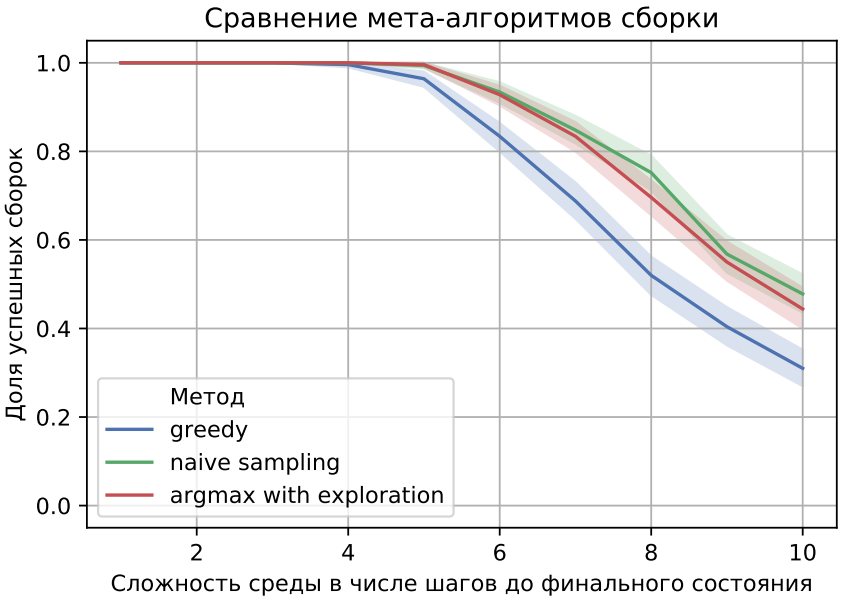


Рис. 7: Сравнение мета-алгоритмов для разной сложности сред

Применение мета-алгоритмов к окружениям разной сложности показало, что предложенные подходы показывают между собой сравнимое качество и показывают в среднем на 18 п.п. большую долю успешных сборок по сравнению с жадным алгоритмом. 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 кубок показывают сравнимое качество.

2. Фиктивная функция награды через долю совпадающих цветов с финальным состоянием позволяет агенту обучаться быстрее, но в долгосрочной перспективе показывает сравнимое качество 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-алгоритме на основе поиска по дереву показывают средний прирост в 18 п.п. по сравнению с жадной стратегией применения DQN по доле успешных сборок Кубика Рубика. 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 обучению с подкреплением к задаче сборке 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, происследованы 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] The diameter of the rubik's cube group is twenty / Tomas Rokicki, Herbert Kociemba, Morley Davidson, John Dethridge // *SIAM Review*. — 2014.

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

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

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

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

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

[7] 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.

[8] 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.

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

[10] 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`.

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

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

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

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