

# Методы эвристического поиска в задаче планирования движений роботов-манипуляторов

Ваше имя

20 декабря 2023 г.

## Содержание

<b>1 Введение</b>	<b>2</b>
<b>2 Описание задачи</b>	<b>3</b>
2.1 Общие положения . . . . .	3
2.2 Постановка задачи с фиксированным конечным положением . . . . .	5
2.3 Постановка задачи с фиксированным конечным положением рабочего органа . . . . .	6
<b>3 Известные алгоритмы</b>	<b>6</b>
3.1 Эвристический поиск . . . . .	6
3.2 A* . . . . .	7
3.3 Weighted A* . . . . .	9
3.4 Lazy A* . . . . .	10
<b>4 Реализация</b>	<b>10</b>
4.1 Симулятор . . . . .	10
4.2 Представление препятствий и манипулятора . . . . .	11
4.3 Работа планировщика . . . . .	12
4.4 Симуляция . . . . .	13
<b>5 Подготовка экспериментов</b>	<b>14</b>
5.1 Выбор карты . . . . .	14
5.2 Методология проведения экспериментов . . . . .	18
<b>6 Экспериментальные исследования</b>	<b>19</b>
6.1 Ускорение планировщика за счёт меньшей точности проверки коллизий . . . . .	19
6.2 Влияние веса эвристики на эффективность A* . . . . .	21
6.3 Влияние количества звеньев на время работы планировщика	23

6.4	Сравнение двух постановок задач . . . . .	25
6.5	Сравнение A* с Lazy A* . . . . .	27
6.6	Выводы . . . . .	31
<b>7</b>	<b>Заключение</b>	<b>31</b>
<b>8</b>	<b>Приложение</b>	<b>32</b>
8.1	Приложение 1. Пример xml-файла . . . . .	32

## 1 Введение

В современном мире робототехника стремительно развивается, уже сегодня человеку во многих сферах деятельности помогают роботы, в том числе и манипуляторы. Они выполняют задачи, которые раньше были доступны только людям: разгружают и загружают транспорт, помогают человеку в проведении сложных хирургических операций, осуществляют сборку техники, сортируют предметы на складах и даже помогают человеку в космосе. Во многих областях нужно, чтобы эти роботы могли работать адаптивно, то есть выполнять не одинаковые последовательные действия, а справляться с разными задачами путём автоматического подбора решений, и для этого нужно уметь планировать их движения.



Рис. 1

1

Планирование движений манипуляторов является важной областью исследований в робототехнике. Однако, эффективное планирование движений манипуляторов является сложной задачей, требующей учета множества факторов, таких как кинематика и динамика манипулятора, препятствия в окружающей среде и требования к точности и скорости выполнения задач. На сегодняшний день не существует универсального алгоритма решения

---

<sup>1</sup>Картинка на Рис. 1 скачана с сайта <https://fabricators.ru/>

данной задачи, поэтому исследования, направленные на её решение являются актуальными.

Цель этой работы - исследование алгоритмов эвристического поиска для планирования роботов-манипуляторов и их анализ. Для её достижения были сформулированы следующие задачи:

- Изучение литературы по теме работы
- Реализация алгоритмов эвристического поиска, применимых к описанной задаче
- Интегрирование алгоритмов в известные симуляторы манипуляторов
- Анализ алгоритмов путём проведения экспериментов

## 2 Описание задачи

### 2.1 Общие положения

В пространстве или на плоскости расположен манипулятор, основание которого закреплено. У манипулятора  $n$  звеньев, соединённых сочленениями, в которых он может поворачиваться. Поворот сочленений вызывает движение манипулятора. Также в этом пространстве расположены препятствия. Положение манипулятора считается допустимым, если его звенья не пересекают препятствия и не пересекают друг друга, а движение считается допустимым, если любое промежуточное положение манипулятора является допустимым.

Формально, есть рабочее пространство  $W \subset \mathbb{R}^2$  or  $\mathbb{R}^3$ <sup>2</sup> и препятствия  $W_{obs} \subset W$ . Свободным рабочим пространством называется  $W_{free} = W \setminus W_{obs}$ , то есть  $W = W_{free} \sqcup W_{obs}$ .

Манипулятор  $M$  состоит из  $n$  звеньев, каждое из которых моделируется прямоугольником. Начало первого звена зафиксировано в какой-то точке. Обозначим за  $\phi_i$  угол поворота звена  $i$  относительно звена  $i-1$  для  $2 \leq i \leq n$  и  $\phi_1$  за угол поворота первого звена относительно глобальных координат в некоторой конфигурации  $c$  данного манипулятора. Заметим, что конфигурация манипулятора (то есть его расположение в пространстве) однозначно задаётся вектором значений  $(\phi_1, \phi_2, \dots, \phi_n)$ , то есть пространство конфигураций манипулятора соответствует подпространству в  $\mathbb{R}^n$ . Пространство конфигураций манипулятора называется  $C$ -пространством. В этой работе конфигурация  $c$  обозначается вектором углов  $c = (\phi_1, \phi_2, \dots, \phi_n)$ .

Рассматривается функция  $shape : C \rightarrow 2^W$ , которая по конфигурации  $c = (\phi_1, \phi_2, \dots, \phi_n)$  манипулятора задаёт множество точек, которое он занимает в рабочем пространстве. Конфигурация  $c$  называется допустимой, если

---

<sup>2</sup>Задачи в  $\mathbb{R}^2$  и  $\mathbb{R}^3$  не имеют качественных отличий, поэтому в рамках этой работы рассматривается  $W \subset \mathbb{R}^2$  и используется соответствующая терминология

$shape(x) \cap W_{obs} = \emptyset$  и нет самопересечений манипулятора.<sup>3</sup> Множество допустимых конфигураций обозначается за  $C_{free}$ , а множество не допустимых  $C_{obs}$ . Тогда  $C = C_{free} \sqcup C_{obs}$ .

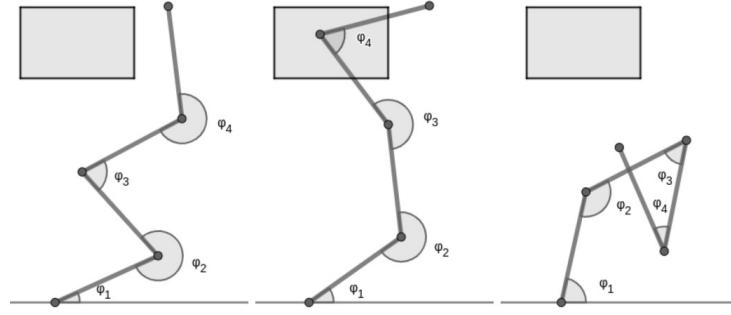


Рис. 2

На рис. 2 проиллюстрирован манипулятор, состоящий из 4 звеньев в конфигурации  $(\phi_1, \phi_2, \phi_3, \phi_4)$  рабочем пространстве с одним прямоугольным препятствием. На левом рисунке эта конфигурация допустимая, на среднем конфигурация допустимой не является, так как манипулятор пересекает препятствие, а конфигурация на правом рисунке не является допустимой из-за самопересечения.

Движение (или действие) манипулятора - это непрерывная последовательность конфигураций манипулятора, то есть отображение  $act : [0, 1] \rightarrow C$ .<sup>4</sup> В этой работе рассматриваются только линейные движения, которые задаются поворотом в одном или нескольких сочленениях. Такое движение можно описать в виде  $n$ -мерного вектора

$a = (\alpha_1, \alpha_2, \dots, \alpha_n)$ , которое переводит манипулятор из конфигурации  $c_0 = (\phi_1, \phi_2, \dots, \phi_n)$  в конфигурацию  $c_1 = (\phi_1 + \alpha_1, \phi_2 + \alpha_2, \dots, \phi_n + \alpha_n)$ . Причём промежуточные конфигурации манипулятора  $M_t = act(t)$  имеют вид  $(\phi_1 + t \cdot \alpha_1, \phi_2 + t \cdot \alpha_2, \dots, \phi_n + t \cdot \alpha_n)$ . Поэтому в этой работе отождествляются понятия движения как отображения и как  $n$ -мерного вектора. Движение называется допустимым, если для всех  $t \in [0, 1]$   $act(t) \in C_{free}$ , то есть все промежуточные конфигурации являются допустимыми.

На практике работа с непрерывным пространством заменяется работой с дискретными моделями, и в нашей работе рассматривается только конечное число конфигураций манипулятора. Для этого вводится минимальный угол поворота  $\delta$  и все доступные конфигурации кратны этому углу.

<sup>3</sup>В этой работе намеренно не вводится формальное определение самопересечения манипулятора, чтобы избежать излишних усложнений конструкции

<sup>4</sup>Область определения этого отображения не играет роли, отрезок  $[0, 1]$  выбран из-за удобной нормировки

Формально, дальше будут рассматриваться только конфигурации из  $C \cap (\delta\mathbb{Z})^n$  и движения  $(\alpha_1, \alpha_2, \dots, \alpha_n), \alpha_i \in \delta\mathbb{Z}$ .

## 2.2 Постановка задачи с фиксированным конечным положением

Неформально, задача заключается в том, чтобы найти последовательность действий между двумя заданными конфигурациями манипулятора при доступных действиях, то есть возможностях манипулятора. Обычно, каждому действию из каждой конфигурации соответствует некоторая стоимость этого действия и к условиям добавляется необходимость минимизировать суммарную стоимость пути.

Формально. Пусть есть множество доступных действий  $A$ . Путём из конфигурации  $c_{start}$  в конфигурацию  $c_{finish}$  называется последовательность конфигураций  $c_{start} = c_1, \dots, c_{k+1} = c_{finish}$ ,  $c_i \in C_{free}$  и действий  $a_1, a_2, \dots, a_k$ ,  $a_i \in A$  манипулятора так, чтобы для каждого  $i \in 1, 1, \dots, k$  действие  $a_i$  из конфигурации  $c_i$  было допустимым, и  $a_i$  переводило бы манипулятор из положения  $c_i$  в положение  $c_{i+1}$ .

Для данной задачи вводится функция стоимости действия  $cost : A \times A \rightarrow \mathbb{R}$ . Стоимость пути  $\pi$  определяется как  $\sum_1^k cost(a_{i-1}, a_i)$ , где  $a_i$  - действия пути,  $a_0 = 0$  по определению, и обозначается  $cost(\pi)$ . Это та величина, которая в рамках задачи будет минимизироваться. В действительности, функция  $cost$  может зависеть только от текущего действия  $a_i$ , если затраты считаются на повороты в сочленениях, или же она может зависеть от предыдущей конфигурации, если пытаться минимизировать опасность пути и стараться двигаться как можно дальше от препятствий. В нашей же модели одна из стоимостей пути будет учитывать, что манипулятору проще продолжить поворачивать одним сочленением, чем сначала повернуть одним, потом другим, поэтому  $cost$  также будет учитывать также предыдущее действие. Остальные тонкости в этой работе не используются.

**Задача.** Задано рабочее пространство  $W = W_{free} \sqcup W_{obs}$ , манипулятор  $M$  и множество доступных действий  $A$ , а также  $c_{start}, c_{finish} \in C_{free}$  - начальная и конечная конфигурации манипулятора. Нужно построить путь  $\pi$  из конфигурации  $c_{start}$  в конфигурацию  $c_{finish}$  с минимально возможным значением  $c(\pi)$ . Эта задача о поиск оптимального решения. Также часто рассматривается задача, при которой требуется найти путь, стоимость  $c(\pi)$  которого превышает минимальную стоимость не более чем в  $w > 1$  раз. Такая задача называется задачей о поиске ограниченно субоптимального решения.

Эта задача может быть сведена к поиску пути на взвешенном ориентированном графе, так как пространство конфигураций дискретизировано. В нём вершины - допустимые конфигурации, а ребром соединены те конфигурации  $c_1, c_2$ , для которых существует  $a \in A$ , переводящее манипулятор из  $c_1$  в  $c_2$ . Весом этого ребра является  $c(c_1, a)$ . Замечание: размерность этого

графа  $n$ , то есть количество вершин этого графа экспоненциально зависит от количества звеньев манипулятора.

### 2.3 Постановка задачи с фиксированным конечным положением рабочего органа

Зачастую, манипулятор - это не просто набор звеньев. На последнем звене устанавливается механизм, с помощью которого можно хватать предметы. Называется этот механизм - рабочий орган. В связи с этим часто перед манипулятором стоит задача не достичь конкретной конфигурации, а захватить определённый предмет, то есть переместить свой рабочий орган или в определённую точку рабочего пространства. В нашей модели рабочим органом называется последний конец  $n$ -ного звена манипулятора.

**Задача** Даны рабочее пространство  $W = W_{free} \sqcup W_{obs}$ , манипулятор  $M$ , множество доступных действий  $A$ , начальная конфигурация  $c_{start} \in C_{free}$  и точка  $x \in W_{free}$ . Требуется также найти путь  $\pi$  между  $c_{start}$  и некоторой конфигурацией  $c_{finish}$ , чтобы в этой конфигурации рабочий орган был в точке  $x$ . По аналогии с предыдущей, в этой задаче можно требовать оптимальность пути или ограниченную субоптимальность. Так как в нашей задаче график конечен, оптимальное решение существует, при условии, что существует хотя бы одно решение. На практике также рабочему органу разрешено находиться в  $\epsilon$ -окрестности точки  $x$  из-за дискретизации пространства.

Также стоит отметить, что на практике важно не только подвести рабочий орган в заданную точку, но и разместить при этом последнее звено манипулятора под нужным углом к этой точке, чтобы захватить цель. Но в задачу текущей работы это условие не вошло, хотя алгоритмы несложно адаптируются к нему.

Эта задача может быть сведена к предыдущей, если найти конфигурацию, для которой end-effector будет находиться в точке  $x$ . Это называется задачей обратной кинематики. Но таких конфигураций много и большая часть из них может быть недостижима, поэтому на практике сведение не упрощает задачу.

## 3 Известные алгоритмы

### 3.1 Эвристический поиск

Эвристический поиск - это метод решения проблемы или поиска оптимального решения, основанный на использовании эвристик. Эвристика - это эвристическое правило или стратегия, которая помогает сократить пространство поиска и упростить процесс принятия решений.

В эвристическом поиске используются эвристические функции, которые оценивают перспективность каждого возможного решения. Эти функции

основаны на знаниях и опыте в предметной области, и могут быть разработаны для конкретной проблемы или задачи.

Основная идея эвристического поиска заключается в том, чтобы исследовать пространство решений, выбирая на каждом шаге наиболее перспективные варианты. Это позволяет сократить количество проверок и ускорить процесс поиска оптимального решения.

Примером эвристического поиска является алгоритм A\*, который используется для нахождения кратчайшего пути во взвешенном графе. Он комбинирует информацию о стоимости пути от начальной вершины до текущей и эвристическую оценку стоимости пути от текущей вершины до конечной. Это позволяет алгоритму выбирать на каждом шаге наиболее перспективные вершины для исследования.

### 3.2 A\*

Алгоритм A\* берёт за основу алгоритм Дейкстры, последовательно рассматривая уже открытые вершины и раскрывая их соседей. Изначально множество открытых вершин состоит только из начальной вершины. В алгоритме Дейкстры выбирается вершина с наименьшей стоимостью пути от начальной вершины, что называется g-значением вершины. А в алгоритме A\* для каждой вершины вычисляется значение эвристической функции heuristic, которая оценивает стоимость пути от текущей вершины в конечную, что называется h-значением вершины и выбирается вершина с минимальным значением суммы  $g + h = f$ -значение вершины.

Алгоритм имеет следующие шаги:

1. Инициализация: Вычисляется h-значение для стартовой вершины и так как g-значение равно нулю, то  $f = h$ . Формируется список открытых вершин, состоящий только из стартовой вершины и пустой список закрытых вершин.
2. Выбор вершины: Из списка открытых вершин выбирается вершина с наименьшим f-значением и устанавливается как текущая.
3. Проверка цели: Если текущая вершина является конечной, то путь найден и алгоритм завершается.
4. Раскрытие вершины: Рассматриваются все соседние вершины текущей вершины. Для каждой соседней вершины вычисляются ее g-значение и h-значение. Если соседняя вершина уже находится в множестве открытых вершин и новое g-значение больше, или соседняя вершина находится в множестве закрытых вершин, то эта вершина пропускается. В противном случае, эта вершина добавляется в множество открытых вершин с вычисленными на этом шаге f,g,h-значениями. По завершению шага текущая вершина перемещается из множества открытых вершин в множество закрытых.

5. Повторение: Шаги 2-4 повторяются до тех пор, пока конечная вершина не будет достигнута или пока множество открытых вершин не станет пустым. Если множество открытых вершин становится пустым, то путь не существует.

Ниже представлен псевдокод алгоритма A\*:

```
function AStar(start, goal, heuristic):
    openSet := {start} // Множество вершин, которые нужно исследовать
    closedSet := {} // Множество уже исследованных вершин

    // Инициализация
    start.g := 0
    start.f := heuristic(start, goal)

    while openSet is not empty:
        current := vertex in openSet with lowest current.f

        if current == goal:
            return reconstructPath(cameFrom, current)

        openSet.remove(current)
        closedSet.add(current)

        for neighbor in current.neighbors:
            if neighbor in closedSet:
                continue

            newG := current.g + cost(current, neighbor)

            if neighbor not in openSet:
                openSet.add(neighbor)
            else if newG >= gScore[neighbor]:
                continue

            neighbor.parent := current
            neighbor.g := newG
            neighbor.f := neighbor.g + heuristic(neighbor, goal)

    return failure
```

Известно, что если эвристика не переоценивает расстояние, то есть  $heuristic(c, goal)$  не больше стоимости кратчайшего пути из  $c$  в  $goal$ , то найденный путь будет оптимальным. Такая эвристика называется допустимой.

В случае рассматриваемых в этой работе задач стоимость действия

$(\phi_1, \phi_2, \dots, \phi_n)$  будет равна  $\frac{1}{\delta} \sum_{i=1}^n |\phi_i|$ , то есть суммарное количество сдвигов на минимальный угол  $\delta$  в этом действии. В случае постановки задачи с фиксированным конечным положением допустимую эвристику можно определить следующим образом. Пусть  $c_1 = (\phi_1, \phi_2, \dots, \phi_n)$ ,  $c_2 = (\psi_1, \psi_2, \dots, \psi_n)$ , тогда  $heuristic(c_1, c_2) = \frac{1}{\delta} \sum_{i=1}^n |\phi_i - \psi_i|$ , то есть манхэттенское расстояние между двумя конфигурациями. Эта эвристика соответствует стоимости кратчайшего пути, если бы в рабочем пространстве не было препятствий.

В случае задачи с фиксированным положением рабочего органа эвристика рассчитывается следующим образом. Вычисляется наибольшее расстояние  $l$ , на которое за одно действие рабочий орган может переместиться. Это расстояние зависит только от манипулятора и набора доступных действий, но не от текущей конфигурации. Для двух конфигураций  $c_1, c_2$  вычисляется расстояние  $d$  между рабочими органами. Тогда манипулятору потребуется хотя бы  $heuristic(c_1, c_2) = \lceil \frac{d}{l} \rceil$  действий, поэтому такая эвристика допустима, но в то же время не очень информативна, так как кратчайший путь может быть намного больше. Таким образом эта эвристика строится только на конечном положении рабочего органа.

### 3.3 Weighted A\*

Алгоритм weighted A\*, или WA\*, в отличие от A\* является не оптимальным, а ограниченно субоптимальным алгоритмом. Модификация состоит в том, что для некоторого веса  $w \geq 1$  эвристика  $heuristic$  заменяется на эвристику  $w \cdot heuristic$ . И если эвристика  $heuristic$  является допустимой, то путь, найденный алгоритмом WA\* будет по стоимости не более чем в  $w$  раз больше оптимального.

Плюсом же такого алгоритма является то, что в процессе работы будет раскрыто гораздо меньше вершин. Можно заметить, что алгоритм A\* раскрывает все вершины, у которых f-значение меньше, чем стоимость кратчайшего пути (что является f-значением для целевой вершины) по правилу выбора вершины для раскрытия, поэтому увеличивая h-значения в  $w$  раз, WA\* увеличивает f-значения вершин и итоговое количество раскрытых вершин будет меньше, а значит меньше будет и время работы. На количество раскрытых вершин строгих гарантий нет.

Стоит отметить, что на практике действительная стоимость найденного пути меньше, чем в  $w$  раз превосходит оптимальную, поэтому с практической точки зрения теоретическая гарантия имеет маленькое значение. То, насколько быстрее работает WA\* в зависимости от веса и во сколько раз величина найденного пути превышает величину оптимального пути будет рассмотрено в секции экспериментов.

### 3.4 Lazy A\*

Разумно предположить, что в задаче планирования манипулятора проверка на столкновения с препятствиями - вычислительно сложный процесс, на который планировщик будет расходовать много времени. Поэтому было решено применить ленивые методы планирования, которые заключаются в следующем. В процессе генерации потомков алгоритм не проверяет, будет ли путь до этого потомка пересекать препятствия или нет и просто добавляет в дерево поиска. Тогда в тот момент, когда очередная вершина достаётся из дерева поиска, сначала алгоритм проверяет, можно ли её достичь без пересечений препятствия из родителя и, если нельзя, то пропускает эту вершину и достаёт следующую. Таким образом, проверки на столкновения откладываются во времени и их большая часть не будет произведена вовсе из-за маленького f-значения потомка. Также в A\* алгоритм мог проверять на пересечения путь до одного и того же состояния из разных предков, но выбирал всё равно ту вершину, которая соответствует кратчайшему пути в это состояние. С Lazy A\* проверка для каждого состояния будет происходить гораздо реже. Этот алгоритм не имеет серьёзных теоретических гарантий, кроме того, что все вершины с f-значением, большим, чем f-значение цели не будут проверены на столкновение, но на практике такие модификации дают серьёзные улучшения.

При этом алгоритме кратко возрастёт размер дерева поиска, потому что там теперь лежат и те вершины, путь до которых пересекает препятствия, но для дерева поиска это увеличение времени каждой операции на маленькую константу, поэтому ослаблением алгоритма это не является.

## 4 Реализация

### 4.1 Симулятор

Первым шагом в реализации алгоритмов были выбор языка программирования и симулятора. Нужно было найти проект симуляции манипуляторов, в который можно было бы встроить алгоритмы, написанные на быстром языке программирования.

Выбор остановился на [тиjосо](#). Это проект с открытым кодом, симулирующий движения роботов, в том числе и манипуляторов, написанный на С. На рисунке 3 показаны примеры роботов, изначально доступных в тиjосо<sup>5</sup>

Плюсы этого симулятора следующие:

- Так как проект написан на С, при разработке алгоритмов можно использовать язык программирования С++ для большей скорости работы
- В случае необходимости можно будет изучить внутреннюю реализацию проекта и доработать под свои нужды, так как это проект с открытым кодом

---

<sup>5</sup>Изображения взяты с сайта тиjосо.org

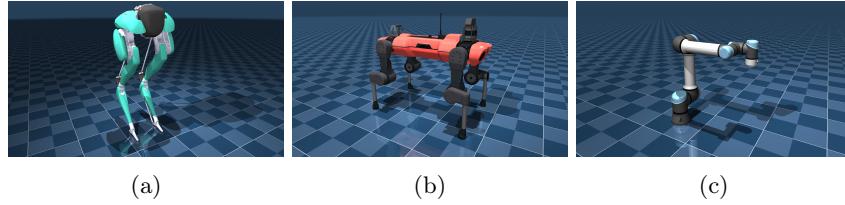


Рис. 3: Примеры моделей роботов в тијосо.

- У тијосо есть хорошая документация и обучающие видео, поэтому начало работы не такое сложное
- Для этого симулятора есть много моделей реальных роботов, на которых в будущем можно будет использовать алгоритмы, поэтому эта работа сохраняет практическую значимость
- Мијосо также симулирует физику предметов, поэтому появляется возможность тестировать алгоритмы и манипулятора в реальной динамической среде

Также есть минус:

- В тијосо и манипулятор и сцена представлены одним файлом, что создаёт некоторые неудобства при желании протестировать несколько роботов на нескольких картах.

Но эти недостатки на фоне всех достоинств незначительны. Видно, что последние два преимущества говорят о том, что проект можно будет масштабировать в дальнейшем.

Мијосо - это 3D симулятор, но благодаря возможности отключать гравитацию можно симулировать и манипуляторы на плоскости, что и было сделано на практике.

## 4.2 Представление препятствий и манипулятора

Препятствия, как и манипулятор, в тијосо задаются xml-файлом, в котором прописываются координаты препятствий, углы поворота относительно глобальных координат, размер и форма. Мијосо поддерживает несколько видов форм: плоскость, сфера, капсула, эллипсоид, цилиндр и прямоугольный параллелепипед. (рис 4) Препятствия в данном проекте статичны, то есть для них используется настройка, запрещающая движения во время симуляции.

В реальных задачах препятствия описываются набором вокселей (в случае 2D - пикселей), из-за особенности задачи распознавания препятствий. Это можно моделировать набором кубиков, но но в данной работе это не сделано и препятствия представляются доступными формами.

Так как симулируется двумерное пространство, то все эти препятствия, как и манипулятор, нужно размещать в одной плоскости. И разнообразие форм заменяется разнообразием их проекций.

Манипулятор задаётся в том же xml-файле, что и препятствия и представляется совокупностью цилиндров, соединённых шарнирами. Так как манипулятор рассматривается в 2D пространстве, то шарниры поворачиваются в одной общей плоскости. Сами звенья имеют форму цилиндра, потому что их проекция - это прямоугольники. Так, если камера повёрнута перпендикулярно к плоскости поворота звеньев, получается симуляция манипулятора в 2D пространстве.

Также для удобства на сцену добавляется ещё одна копия манипулятора другого цвета, которая визуализирует цель в задаче с фиксированным конечным положением, а для корректности симуляции отключается проверка на столкновения этой копии со всеми остальными объектами. То есть любые другие объекты могут безпрепятственно проходить сквозь эту копию.

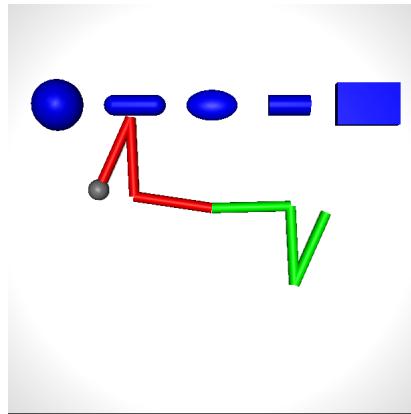


Рис. 4: Пример манипулятора и препятствий.

На рис. 4 изображен пример сцены с препятствиями и манипулятором. Синим нарисованы препятствия разной формы, красным - манипулятор, а зелёным его целевое конечное положение. Серый шарик моделирует рабочий орган, у которого, как у зеленой копии, отключена проверка на столкновения. Пример xml-файла для этой сцены представлен в приложении 1.

### 4.3 Работа планировщика

Планировщик состоит из двух модулей - алгоритма и информации о среде. Модуль алгоритма принимает на вход следующий набор методов: метод проверки конфигурации на целевую, метод проверки допустимости действия, метод расчёта стоимости действия и эвристику. Также модуль алгоритма принимает множество доступных действий. Эти входные параметры могут отличаться в разных постановках задач, так, например, в данном

проекте есть два разных типа задач, у которых отличается эвристика и метод проверки конфигурации на целевую.

В модуле алгоритма реализован А\*. Для его работы требуются описанные выше методы. Он выдаёт найденный путь или сообщение о том, что пути нет, а также статистику о времени работы различных методов, стоимости пути, количестве итераций алгоритма.

В модуле информации о среде реализуются методы, подаваемые на вход в модуль алгоритма. Здесь используются встроенные функции тијосо проверки на столкновение и рассчёта положения рабочего органа в данной конфигурации. Как правило, для этих целей планировщик использует копию сцены, меняет на ней состояние манипулятора и вызывает функции тијосо.

Ввиду того, что функция проверки на столкновения и расчёта положения рабочего органа является вычислительно сложными, было принято решение модифицировать и воспользоваться низкоуровневыми функциями симулятора для более быстрой работы этих методов. По умолчанию Мијосо предоставляет для взаимодействия метод симуляции, внутри которого производятся проверки на столкновения и физические расчёты, в том числе изменение скорости и ускорения объектов на сцене. В данном случае они лишние, так как для этого проекта характерны нулевые скорости и ускорения, поэтому для модуля информации о среде были удалены лишние методы, благодаря чему планировщик должен работать быстрее. При этом усовершенствования алгоритмов симулятора не проводилось. Подробнее о скорости работы описано в главе 5.

В проекте реализовано только две эвристики, описанные выше, и каждая используется для своего типа задач, так что без написания исходного кода эвристики поменять не получится.

#### 4.4 Симуляция

Проект работает сразу с набором из нескольких заданий. Это могут быть заранее заготовленные задания или сгенерированные случайным образом. Задание описывается типом - фиксированное конечное положение или фиксированное положение рабочего органа, а также координатами этого положения. Для набора заданий также задаются следующие параметры: вес эвристики, максимальное время, отведённое на решение задачи, количество случайных тестов или файл с описанием тестов, а также сцена, на которой происходит работа. При симуляции задания сначала выставляется начальное и конечное положение манипулятора на сцене. Затем планировщик ищет путь и, если путь найден успешно, он возвращается как последовательность действий. Эти действия по очереди применяются к манипулятору в рамках симуляции.

Симуляция в тијосо происходит по шагам, один шаг обычно симулирует одинаковый промежуток времени. Они называются шагами симуляции и за один такой шаг тијосо определяет столкновения между объектами, вычисливает их местоположение на следующий момент времени, скорость и ускорение. В промежутках между шагами проект может менять физические

характеристики объектов на сцене, но не в каждый шаг симуляции такие изменения производятся, чтобы симуляция не была слишком быстрой. Также несколько раз в секунду текущее расположение объектов отображается на экране.

Симулятор позволяет взаимодействовать с каждым сочленением, выставляя ему угол, скорость и ускорение поворота. Это максимально приближено к реальности, так как манипулятор приводится в движение под действием силы, то есть контролируется именно ускорение поворота сочленений. Но так как в данной работе путь - это последовательность конфигураций, то для симуляции движений меняется именно угол поворота, а не скорость или ускорение. Для того, чтобы применить действие  $a = (\alpha_1, \alpha_2, \dots, \alpha_n)$  за  $N$  шагов симуляции к текущему состоянию прибавляется  $\frac{1}{N}a$ , где  $N$  - заранее выбранная константа, для моделирования непрерывности движения.

Так как планировщик может выдавать некорректные пути, в среде симуляции была реализована возможность заметить, что манипулятор столкнулся с препятствием на своём пути. При этом в консоль будет выведено сообщение о столкновении. В большинстве случаев таким столкновениям несущественны, так как лишь немногие задавают препятствия, и после него симуляция продолжится. Но в некоторых случаях в результате столкновения тицосо может работать некорректно и процесс завершится с ошибкой. Таким образом, приложение не только визуализирует план действий манипулятора, но и позволяет проверить корректность найденного решения.

Для запуска проекта перейдите в репозиторий на гитхабе по ссылке [https://github.com/machine-solution/motion\\_planning\\_for\\_manipulators](https://github.com/machine-solution/motion_planning_for_manipulators) и следуйте инструкции в README.md

## 5 Подготовка экспериментов

### 5.1 Выбор карты

Для того, чтобы сравнить вариации алгоритма на практике нужны задания, но общезвестных датасетов для задачи планирования манипуляторов найти не удалось, поэтому нужно было самостоятельно выбирать карту с препятствиями и задания на ней. Для сравнения было сгенерировано несколько карт и так как планировщик ищет решения в пространстве конфигураций, то также для каждой карты было визуализировано это пространство. На изображениях слева нарисована карта, а справа пространство конфигураций, которое, для наглядности, было удвоено, так как первое сочленение может поворачиваться на  $360^\circ$  без самопересечений. По горизонтали угол поворота первого сочленения, по вертикали - второго, таким образом, точка обозначает конфигурацию манипулятора. Красным изображены конфигурации, которые пересекают препятствия, черным - которые не пересекают. Для каждой карты было сгенерировано 25 случайных заданий. Для каждого задания ломаной нарисовано элементарное решение, получающее-

еся поворотом сначала первого сочленения, а потом второго. Если решение пересекает препятствия, оно раскрашено оранжевым, иначе зелёным.

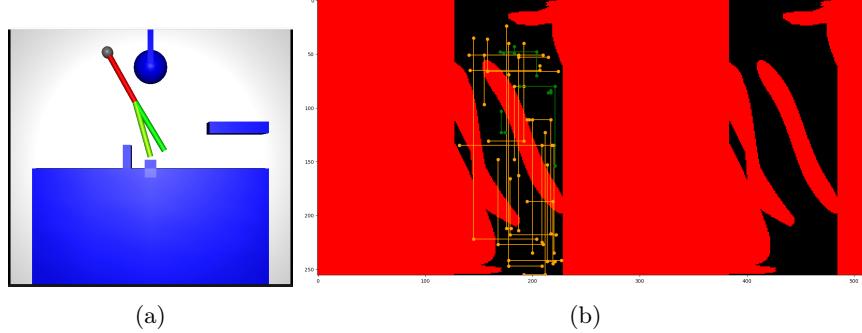


Рис. 5: Карта в форме стола

Карта 5 имеет вид стола с небольшим количеством препятствий. Эта карта хороша тем, что на ней очень много заданий, у которых элементарное решение пересекает препятствия. Но у этой карты очень мало свободного пространства и задания будут генерироваться не разнообразными, что не очень подходит для экспериментов.

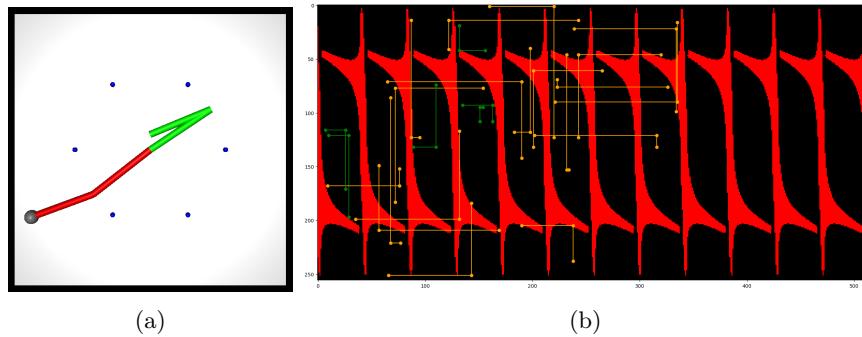


Рис. 6: Карта с симметричными шарами

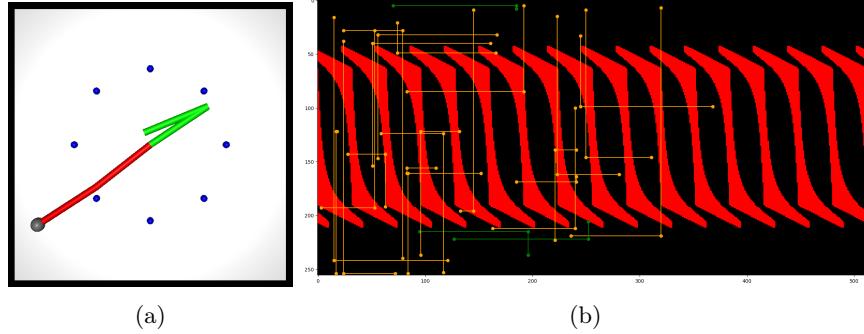


Рис. 7: Карта с симметричными шарами, расположенными дальше от центра

Карты 6 и 7 имеют регулярную структуру из одинаковых расположенных по кругу шаров, они отличаются количеством и размером шаров и расстоянием до крепления манипулятора. На этих картах ещё больше заданий пересекают препятствия элементарными решениями, но разнообразности заданий ещё меньше. Здесь если задания генерируются в одной и той же области между шарами, то задание получается очень простым, а все остальные задания имеют примерно одинаковую сложность.

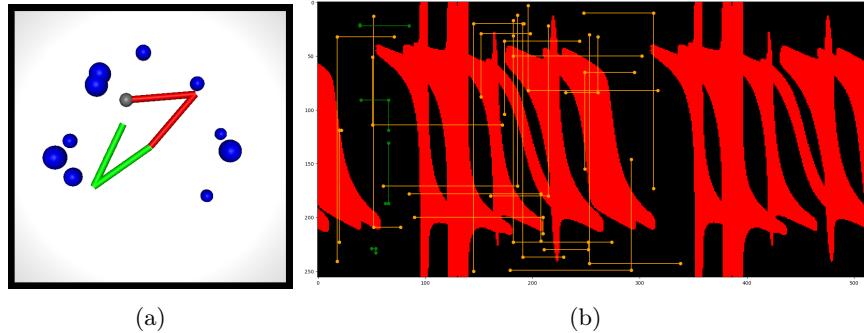


Рис. 8: Карта из случайных шаров

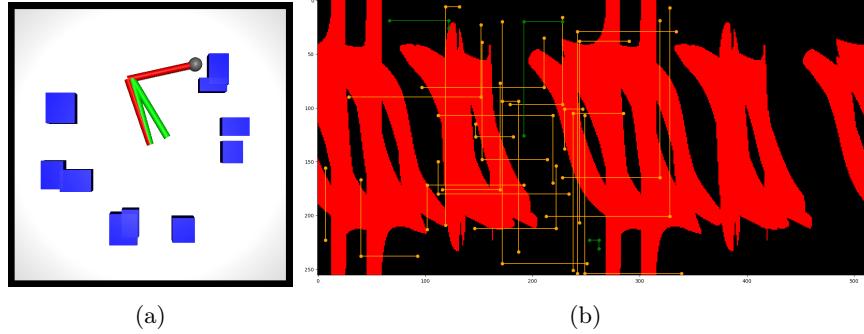


Рис. 9: Карта из случайных параллелепипедов

Карты 8 и 9 были сгенерированы случайным образом и на них достаточно большое разнообразие заданий, хоть и большее число из них решаются элементарно. Единственный минус в том, что препятствия имеют выпуклую форму и такие препятствия легко обходить планировщику, поэтому одна из случайных карт была доработана вручную так, чтобы в пространстве конфигураций было больше невыпуклых частей и получилась карта 10. На ней сложные задания должны быть ещё сложнее из-за такой структуры пространства конфигураций.

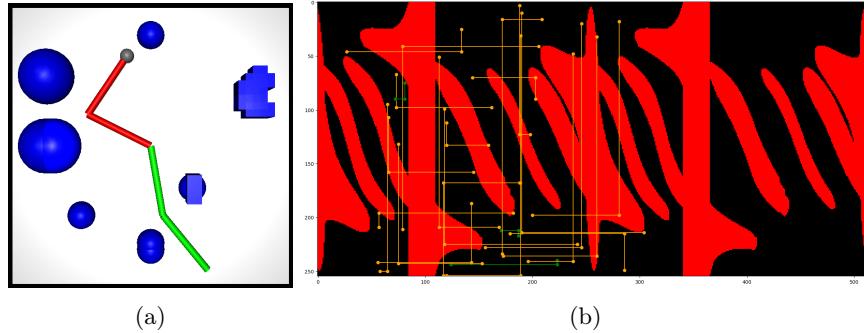


Рис. 10: Первая карта с добавленным наверх шаром

Также для сравнения количества заданий, которые решаются элементарно, было сгенерировано 1000 заданий на каждой карте, по таблице 1 видно, что на всех картах большая часть заданий сложная, то есть элементарное решение пересекает препятствия, но самый большой процент сложных заданий на картах 6 и 10. Поэтому из-за разнообразности, сложности и большого количества заданий, которые не решаются элементарно, для экспериментов была выбрана карта 10.

map	5	6	7	8	9	10
difficulty tasks of 1000, %	74.1	94.9	85.8	79.6	83.3	86.9

Таблица 1: Процент заданий из 1000, которые не решаются элементарно для каждой карты.

## 5.2 Методология проведения экспериментов

Все эксперименты проводились на одной карте рис. 11 из прошлой главы. Она одновременно свободная в верхней части и достаточно плотно заставлена препятствиями в нижней части, поэтому на этой сцене генерируются и простые и сложные задания.

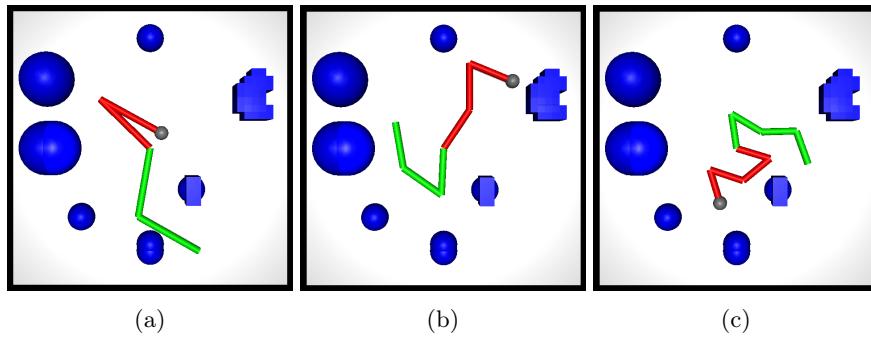


Рис. 11: Карта, на которой проводились эксперименты.

Набор заданий генерировался случайным образом для каждого эксперимента, при этом из всех генерированных заданий выбирались те, у которых существует решение. Количество заданий в каждом эксперименте находится в диапазоне от 50 до 300.

В экспериментах отслеживались основные показатели алгоритмов эвристического поиска: найден ли путь, число итераций алгоритма, максимальный размер дерева поиска в памяти, стоимость найденного пути и время работы. Также во время экспериментов измерялись количество вызовов конкретных функций, таких как проверка конфигурации на столкновение с препятствиями и методы работы с деревом поиска, и их суммарное время работы.

Большая часть экспериментов проводилась с 2-dof манипулятором как на рис 11a, но отдельные эксперименты были проведены с большим количеством звеньев (рис 11b, 11c), при этом совокупная длина манипулятора оставалась неизменной. То есть с увеличением количества звеньев уменьшалась длина одного звена.

Все эксперименты проводились на компьютере с процессором Intel Xeon Gold 6338 с частотой 2 GHz, и с 16 Гб оперативной памяти.

## 6 Экспериментальные исследования

### 6.1 Ускорение планировщика за счёт меньшей точности проверки коллизий

Проверка на столкновения манипулятора и препятствий является вычислительно сложной задачей, поэтому целью первого эксперимента было выяснить какую долю времени работы планировщика она занимает и насколько можно сократить общее время работы, если делать менее качественные проверки на столкновение с препятствиями и как часто путь, найденный планировщиком, будет задевать препятствия.

При симуляции минимальный угол  $\delta$  (примерно  $1.4^\circ$ ) делится на 8 частей и за шаг симуляции манипулятор поворачивает сочленение на  $\frac{\delta}{8}$ , то есть если проверять все 8 промежуточных состояний, то найденный путь не будет задевать препятствия. При этом можно проверять не 8, а меньшее число промежуточных состояний, например 4, 2 и 1. Благодаря такому способу меньше времени будет расходоваться на проверку столкновений, но некоторые действия, при применении которых манипулятор задевает препятствие планировщик может посчитать допустимыми. Для 8 проверок распределение времени работы функций представлено на рис 12.

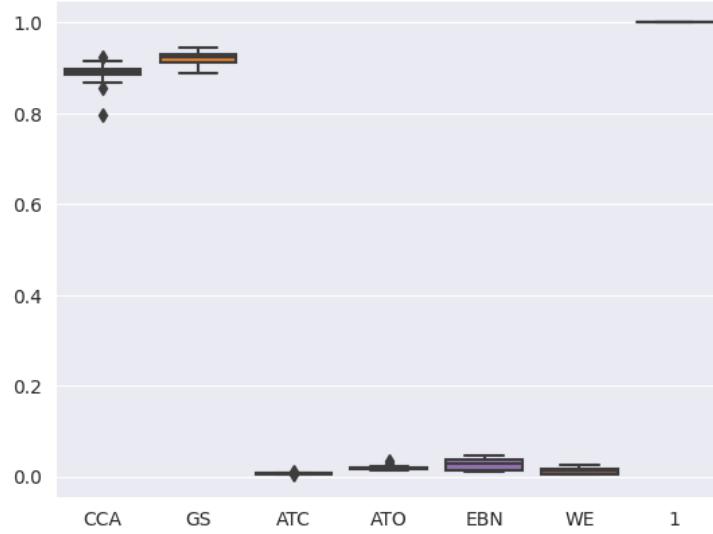


Рис. 12: Время, затрачиваемое на разные методы планировщика, при 8 проверках на столкновение

Пояснение к графику 12: по горизонтали отмечены названия основных функций, используемых планировщиком. CCA - CheckCollisionAction - проверка на столкновение с препятствиями, GS - GenerateSuccessors - определяет доступные шаги, вызывая CheckCollisionAction, ATC - AddToClosed -

type	4 checks	4 checks + boost	1 check	1 check + boost
mean runtime, ms	269.3	201.2	128.5	107.4

Таблица 2: Среднее время работы планировщика в миллисекундах при разных проверках на столкновение.

функция дерева поиска, добавляет вершину в список closed, ATO - AddToOpen - добавляет вершину в список open, EBN - ExtractBestNode - выдаёт вершину с наименьшим f-значением, вызывает WasExpanded, WE - WasExpanded - проверяет находится ли вершина в списке closed, 1 - общее время работы планировщика.

Для каждого из 100 тестов время работы функций было отнормировано на время работы планировщика и представлено в виде box-and-whisker plot. По вертикали, таким образом, отмечено время работы функции относительно времени работы планировщика. Видно, что 90% времени работы занимает функция проверки на столкновения.

Также тесты запускались с меньшим количеством проверок промежуточных состояний - 4, 2 и 1. Также были сделаны доработки в функции проверки на столкновения, описанные в главе 4. Для 4 проверок ни в одном из 100 тестов не было обнаружено столкновений с препятствиями, а для 2 и 1 проверок было 23 одинаковых теста, в которых итоговый путь манипулятора незначительно задевал препятствия. Поэтому дальше представлены результаты экспериментов для четырёх и одной проверки соответственно.

В таблице 2 представлено среднее время работы планировщика в миллисекундах с разными типами проверки на столкновения. "boost" означает наличие доработки, описанной в главе 4 в разделе "Работа планировщика". Из таблицы видно, что при 4 проверках общее время работы алгоритма в среднем более чем в 2 раза привышает время работы алгоритма с одной проверкой промежуточного состояния при потери точности 23%. Стоит отметить, что на практике на вход алгоритму препятствия подаются в немногом увеличенном размере в целях увеличения надежности, ввиду того, что перемещения манипулятора в симуляторе и в действительности могут немного отличаться. Поэтому незначительное столкновение с препятствием на сцене не означает столкновение с препятствием в действительности. Также видно, что доработка функции проверки на столкновения даёт ещё примерно 17% уменьшения общего времени работы планировщика.

На рисунке 13 представлено распределение времени работы между функциями при проверке одного промежуточного состояния и с доработками. Как видно, теперь функция проверки на столкновения занимает уже 50%, а не 90% от общего времени работы.

На графике 14 отображено время работы на каждом из тестов, отсортированных в порядке возрастания времени работы. По вертикали отмечено время работы в миллисекундах, по горизонтали перцентили. Синяя кривая визуализирует 4 проверки без доработки, зелёная 4 проверки с доработкой, оранжевая 1 проверка без доработки, красная 1 проверка с доработкой. По

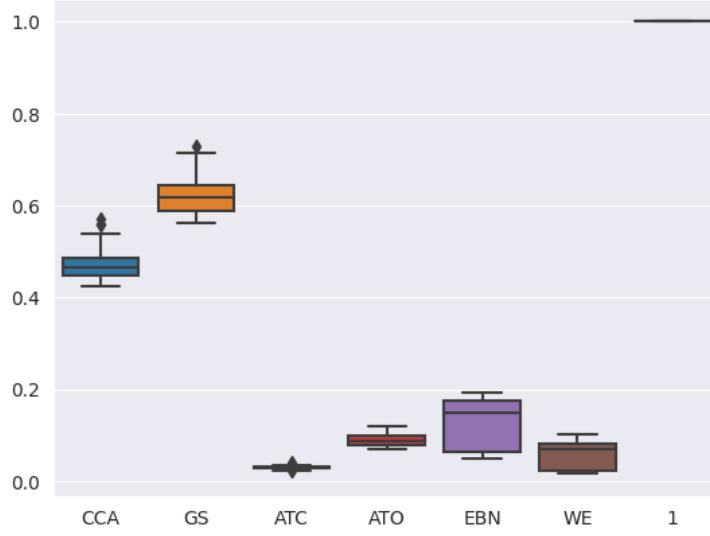


Рис. 13: Время, затрачиваемое на разные методы планировщика, при 1 проверке на столкновение и ускоренной проверке на столкновения

weight	1.0	1.1	1.2	1.5	2.0	4.0	10.0	30.0	100.0
runtime	1025.0	1003.7	977.0	901.7	774.6	525.0	424.7	391.4	387.1

Таблица 3: Среднее время работы планировщика в миллисекундах при различном весе эвристики

графику видно, что на простых тестах (левые 40%) разницы практически нет, а на сложных (правые 20%) время работы отличается в разы.

## 6.2 Влияние веса эвристики на эффективность A\*

Целью данного эксперимента было выяснить, как вес эвристики A\* меняет время работы и качество пути в задаче планирования манипулятора. На основном количестве тестов разницы не было, так как путь к цели был простой, поэтому из 5000 случайных заданий было выбрано 200 с самым большим временем работы. Результаты представлены на графике 15 и таблице 3.

На графике 15 по вертикальной оси отмечено количество итераций алгоритма, а на горизонтальной - вес эвристики A\*. В таблице 3 представлено среднее время работы планировщика в миллисекундах для каждого веса. Видно, что на сложных тестах вес, начиная от 4, существенно ускоряет алгоритм. При весе 100 среднее время работы в 2.6 раза меньше, чем у A\*.

На графике 16 по вертикальной оси представлено превышение стоимости пути над оптимальным, то есть отметка 1.5 означает, что найденный

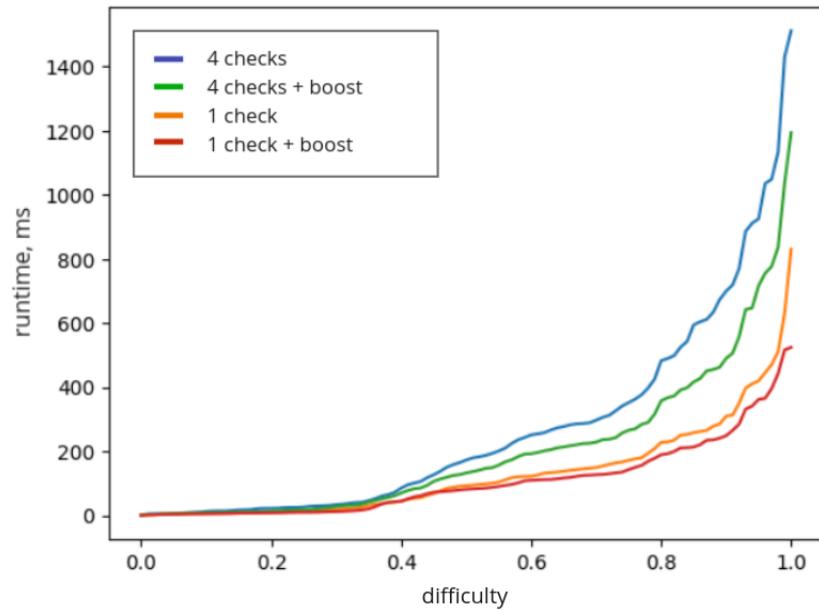


Рис. 14: Время работы планировщика с 4 версиями проверки на столкновения.

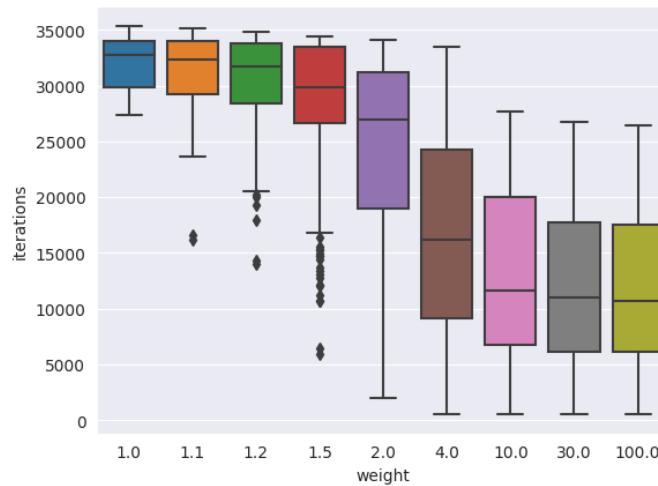


Рис. 15: Количество итераций в зависимости от веса.

путь в 1.5 раза больше по стоимости, чем оптимальный. По горизонтали также отмечен вес WA\*. На этом графике видны тенденции, обратные двум

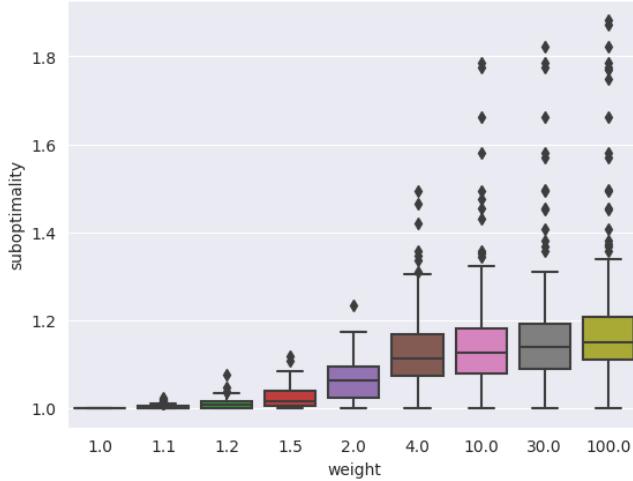


Рис. 16: Превышение длины пути над оптимальным волях.

прошлым графикам. Начиная с веса 2 субоптимальность найденного пути существенно возрастает, но все отметки далеки от теоретических значений, то есть величины  $weight$  на этом графике.

### 6.3 Влияние количества звеньев на время работы планировщика

Целью данного эксперимента было проверить, много ли заданий для большего количества звеньев сможет выполнить планировщик. Было выбрано 100 случайных заданий, для которых существует решение, для трёхзвеного и четырёхзвенного манипулятора. Для каждого задания выставлялся лимит времени работы 5 минут.

На графике 17 представлена зависимость процента заданий трёхзвенного манипулятора, которые успели завершиться за отведённое время в зависимости от веса эвристики при разных временных бюджетах. Видно, что при ограничении в 1 секунду процент таких заданий составляет от 40 до 65 в зависимости от эвристики, но уже при менее строгом ограничении планировщик может решить от 90 до 100 процентов заданий. Самое долгое задание при весе 1.0 занимало почти 280 секунд времени.

На графике 18 представлены аналогичные данные для четырёхзвенного манипулятора, но с немного другими временными ограничениями, так как сами задания выполняются дольше. Видно, что при добавлении ещё одного звена все задания решаются только при большом весе эвристики и за 5 минут, вместо одной минуты. Также можно заметить, что линия для бюджета времени в 1 секунду несколько выше, чем у графика для трёхзвенного манипулятора. Это можно объяснить тем, что так как длина манипулятоа

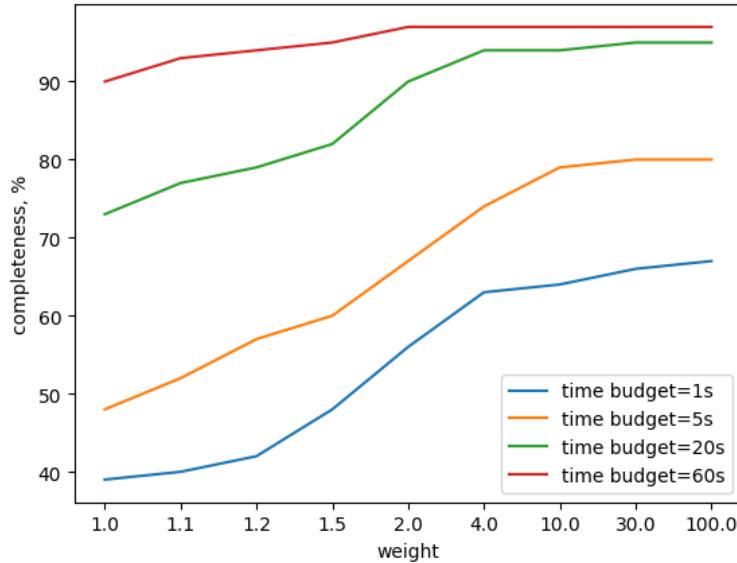


Рис. 17: Процент выполненных алгоритмом A\* заданий трёхзвенного манипулятора в зависимости от веса эвристики.

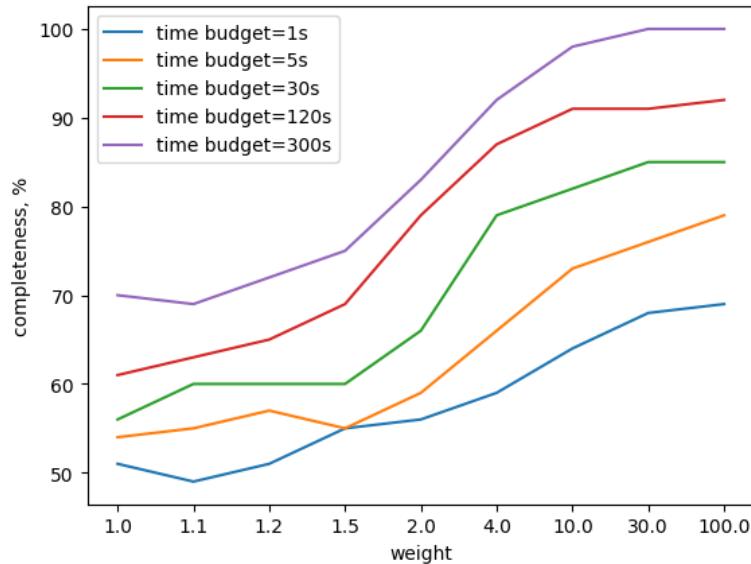


Рис. 18: Процент выполненных алгоритмом A\* заданий четырёхзвенного манипулятора в зависимости от веса эвристики.

weight	1.0	1.1	1.2	1.5
mean runtime cfg, ms	1025.0	1003.7	977.0	901.7
mean runtime xy, ms	2491.2	2483.5	2478.2	2445.2

Таблица 4: Сравнение среднего времени работы для двух типов задач, маленькие веса

осталась той же, а длина каждого звена в отдельности уменьшилась, в большем числе заданий на пути между манипулятором и целью не встречается препятствий, поэтому такие задания решаются за несколько миллисекунд.

#### 6.4 Сравнение двух постановок задач

В отличие от задачи с фиксированным конечным положением, где эвристикой выступает минимально возможное количество действий, за которые в пространстве без препятствий можно достичь одного состояния из другого, в задаче с фиксированным положением рабочего органа эвристика хуже - она оценивает только расстояние между текущим и целевым положением рабочего органа. Целью данного эксперимента было проверить насколько дольше решается задача с конечным положением рабочего органа чем задача с конечной конфигурацией манипулятора.

Для каждого типа задания случайным образом генерировалось 5000 примеров, из которых выбиралось 200 с наибольшим временем работы.

В таблице 4 представлено время работы планировщика в постановке задачи с конечной конфигурацией, обозначенной cfg, и в новой постановке с конечным положением рабочего органа, обозначенной xy. Видно, что при маленьких весах задания в постановке задачи с конечной конфигурацией решаются примерно в 2.5 раза быстрее.

Но уже на графике 19 видно, что при большом весе время работы практически одинаковое у двух постановок задач.

Также это видно на графике 20, на котором по вертикали отмечено количество итераций алгоритма. Можно заметить, что количество итераций даже меньше. Возможно, это случается потому что в большинстве заданий существует две конфигурации с общим положением рабочего органа, являющиеся решением задачи. С другой стороны, это компенсируется тем, что расчёт положения рабочего органа по конфигурации - вычислительно сложный процесс, поэтому одна итерация занимает больше времени.

Также из графика 21 видно, что большой вес компенсирует неточность эвристики по тому насколько близкие к оптимальным путям находит планировщик для весов  $\leq 4.0$ .

Анализируя все 3 графика можно прийти к выводу, что новая постановка задачи не является более сложной для алгоритма, представленного в этой работе.

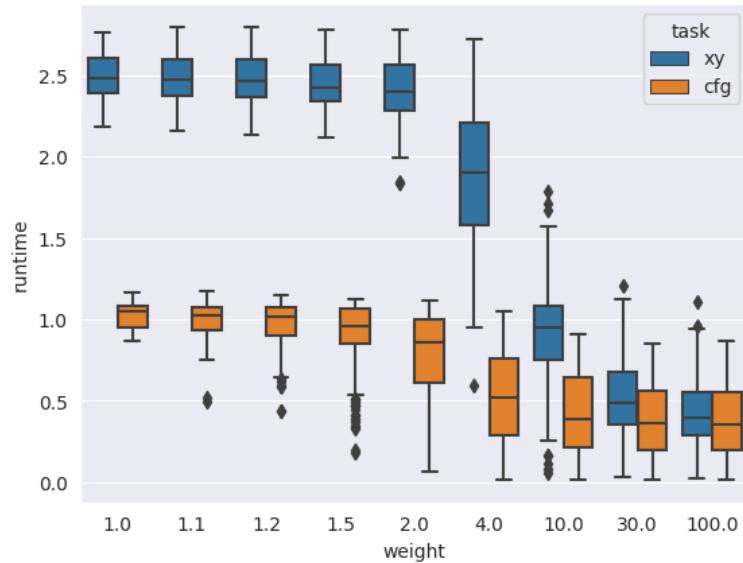


Рис. 19: Сравнительное время работы планировщика с разными постановками задач, все веса.

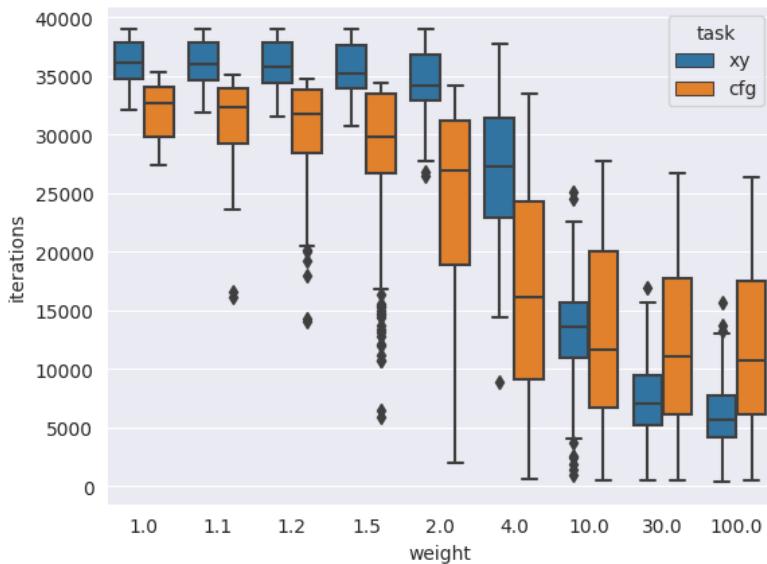


Рис. 20: Сравнительное количество итераций с разными постановками задач, все веса.

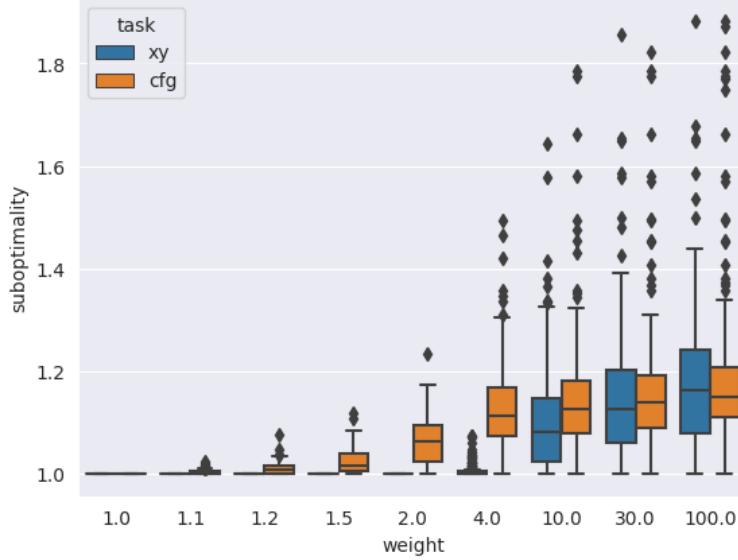


Рис. 21: Сравнительная субоптимальность пути с разными постановками задач, все веса.

## 6.5 Сравнение A\* с Lazy A\*

Сравнения результатов двух алгоритмов были проведены на манипуляторах с разным количеством звеньев. Для 2-dof использовались 200 сложных тестов из прошлых экспериментов. Для сравнения времени работы в 3-dof было выбрано 50 из 500 случайных тестов с самым большим временем работы при весе эвристики 1. Так как для 4-dof манипуляторов время работы довольно большое, для сравнения использовались веса эвристики начиная с 4. Из 200 случайных тестов было выбрано 40 с самым большим временем работы с весом эвристики 4.

На графике 22 представлена разница во времени работы A\* и Lazy A\*. Видно, что улучшенная версия алгоритма работает в 2 раза быстрее и отклонение времени работы от среднего значения меньше. Это очень хороший результат с учётом того, что качество найденного пути остаётся неизменным. Также по графикам 23, 24 видно, что с увеличением количества звеньев время работы Lazy A\* остаётся в 2 раза меньше времени работы A\*. На этом графике значение 300 означает, что за 300 секунд алгоритм A\* не нашёл решения, так как использовалось ограничение по времени 5 минут.

Для алгоритма Lazy A\* важной статистикой является отношения количества раскрытых вершин к количеству вершин, добавленных в дерево поиска. В таблице 5 представлен процент раскрытых вершин из всех рассмотренных под названием evaluated. Видно, что эти значения близки к  $\frac{1}{2 \cdot dof}$ , которое достигается в случае, когда алгоритм каждый раз выбирает

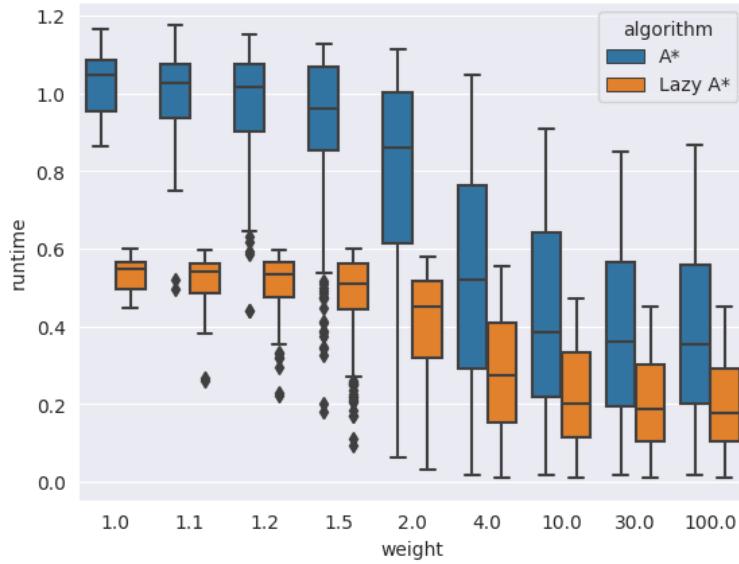


Рис. 22: Сравнение времени работы разных алгоритмов для двузвленного манипулятора

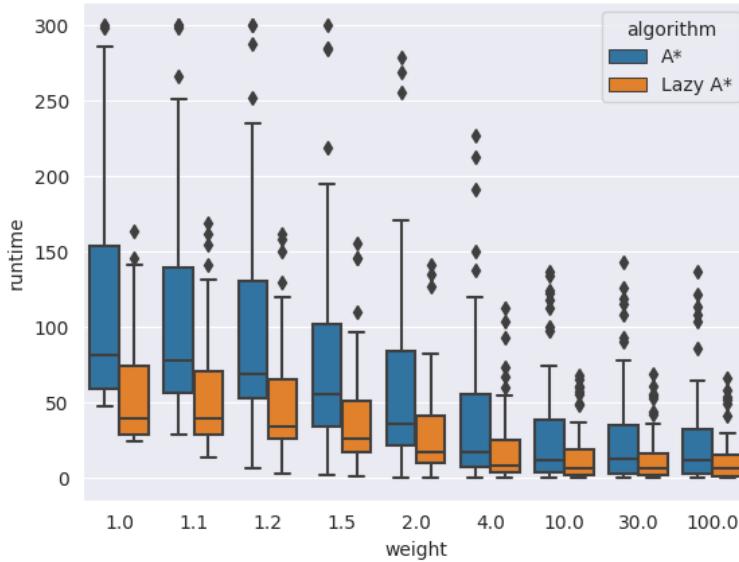


Рис. 23: Сравнение времени работы разных алгоритмов для трёхзвленного манипулятора

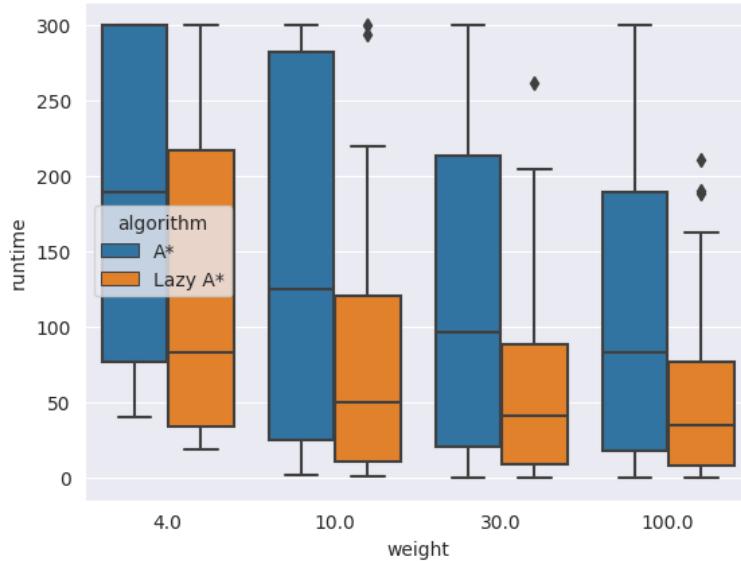


Рис. 24: Сравнение времени работы разных алгоритмов для четырёхзвеного манипулятора

dof	2	3	4
evaluated, %	28	19	16

Таблица 5: Сравнение среднего времени работы для двух типов задач, маленькие веса

вершину на оптимальном пути. Также было замечено, что с ростом веса эвристики эта статистика немного возрастает, что соответствует менее качественному выбору вершин алгоритмом.

Также было проведено сравнение процента выполненных заданий для 3 и 4-звенного манипулятора в зависимости от лимита по времени для A\* и Lazy A\*. Для трёхзвенного манипулятора использовалось 500 случайных заданий, для четырёхзвенного - 200.

Сравнивая графики 25, 26 для Lazy A\* с аналогичным графиками для A\* из прошлых разделов (17, 18) можно увидеть, что процент выполненных заданий стал выше, но не сильно, это объясняется тем, что есть немалое количество очень сложных заданий, которые планировщик решает долго даже с ускорением в 2 раза.

Исходя из описанных выше данных можно сделать вывод, что Lazy A\* - существенное улучшение алгоритма, но недостаточное для того, чтобы с лёгкостью решать задания с большим количеством звеньев.

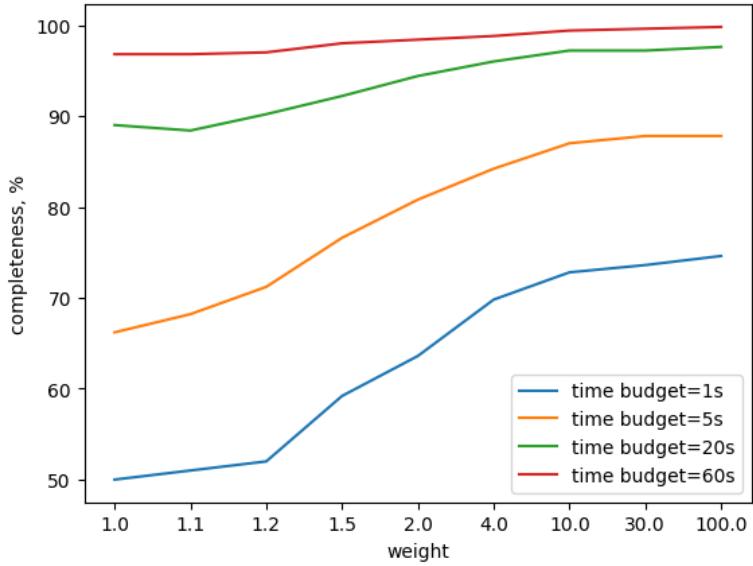


Рис. 25: Процент выполненных алгоритмом Lazy A\* заданий трёхзвенного манипулятора в зависимости от веса эвристики.

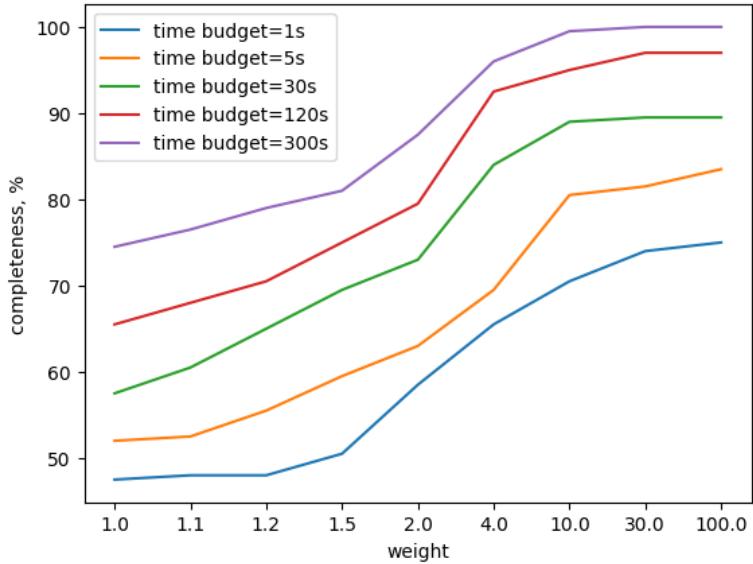


Рис. 26: Процент выполненных алгоритмом Lazy A\* заданий четырёхзвенного манипулятора в зависимости от веса эвристики.

## 6.6 Выводы

Исходя из экспериментальных исследований, представленных выше, можно сформулировать следующие выводы:

- Процедура проверки на столкновения затрачивает значительные ресурсы планировщика, поэтому, уменьшив количество таких проверок можно уменьшить общее время работы алгоритма планирования в 2 раза, но при этом часть задач будет решаться с незначительными столкновениями..
- Увеличение веса эвристики может значительно уменьшить время работы на сложных заданиях, например, при весе равном 100 медианное время работы в 3 раза меньше, чем при весе 1, а длина пути в среднем всего на 20% больше оптимальной. При этом существенное влияние на время работы оказывают веса в промежутке от 1.5 до 30.
- С увеличением числа звеньев время работы значительно растёт. Согласно экспериментам, для двух звеньев все задания выполняются за секунду с весом эвристики 1, для трёх звеньев большая часть задачий примерно за 60 секунд, для четырёх звеньев не все тесты при весе 1 успевают отработать за 5 минут (300 секунд).
- В сравнении двух постановок задач видно, что для больших весов эвристики время работы и субоптимальность пути практически не отличаются, а в задаче с конечным положение рабочего органа итераций алгоритма даже меньше, чем в задаче с конечной конфигурацией. Это объясняется тем, что в первом случае одна итерация вычислительно сложнее за счёт необходимости по конфигурации находить положение рабочего органа.
- Lazy A\* обеспечивает намного лучшее время работы, но задания для четырёхзвенного манипулятора даже при большом весе всё ещё решаются за минуту.

## 7 Заключение

В данной курсовой работе были детально рассмотрены и проанализированы алгоритмы планирования пути для манипуляторов, в результате чего были выявлены важные особенности их функционирования.

В ходе экспериментов было установлено, что основным фактором, влияющим на время планирования пути, является необходимость расчета положения манипулятора в пространстве, что связано с вычислительной сложностью задачи восстановления конфигурации манипулятора по известным углам поворота его сочленений.

Также было отмечено, что с увеличением количества звеньев манипулятора существенно возрастает размер пространства его возможных конфигураций, что в свою очередь приводит к увеличению времени, необходимого для планирования пути.

Кроме того, было обнаружено, что использование больших весов для эвристических функций позволяет существенно ускорить процесс планирования, и потенциально, применение более информативных эвристик может привести к еще большему ускорению алгоритма.

Модификация Lazy A\* даёт ускорение в два раза для разного количества звеньев манипулятора, но этого всё равно не достаточно для того, чтобы быстро планировать пути для четырёхзвенного манипулятора.

Таким образом, на основе проведенного исследования можно сделать вывод о том, что алгоритм A\* в его стандартном виде недостаточно эффективен для планирования движений многозвенных манипуляторов без применения специальных модифицирующих техник и очень информативных эвристик.

Реализация проекта находится в github-репозитории по ссылке [https://github.com/machine-solution/motion\\_planning\\_for\\_manipulators/tree/main](https://github.com/machine-solution/motion_planning_for_manipulators/tree/main). В файле README.md описано как установить необходимые компоненты, запустить проект и настроить параметры для запуска.

## 8 Приложение

### 8.1 Приложение 1. Пример xml-файла

```
<mujoco>
<asset>
<material name="red"    rgba="1 0 0 1"/>
<material name="green"  rgba="0 1 0 1"/>
<material name="blue"   rgba="0 0 1 1"/>
<material name="white"  rgba="1 1 1 1"/>
<material name="gray"   rgba=".5 .5 .5 1"/>
</asset>

<option collision="predefined"/>

<worldbody>
    <light diffuse=".5 .5 .5" pos="0 0 10" dir="0 0 -1"/>
    <geom type="plane" size="2 2 0.1" rgba="1 1 1 1"/>

    <body name="edge 0" pos="0.375 0 0.1" euler="0 90 0">
        <joint name="joint 0" type="hinge" axis="-1 0 0" pos="0 0 -0.375"/>
        <geom name="geom link 0" type="cylinder" size="0.05 0.375" material="red"/>
        <body name="link 1" pos="0 0 0.75" euler="0 0 0">
            <joint name="joint 1" type="hinge" axis="-1 0 0" pos="0 0 -0.375"/>
            <geom name="geom link 1" type="cylinder" size="0.05 0.375" material="red"/>
            <body name="link 2" pos="0 0 0.75" euler="0 0 0">
                <joint name="joint 2" type="hinge" axis="-1 0 0" pos="0 0 -0.375"/>
                <geom name="geom link 2" type="cylinder" size="0.05 0.375" material="red"/>
                <site name="tip" size="0.1" pos="0 0 0.375"/>
            </body>
        </body>
    </body>
</worldbody>
```

```

        </body>
    </body>

    <body name="link 0 shade" pos="0.375 0 0.1" euler="0 90 0">
        <joint name="joint 0 shade" type="hinge" axis="-1 0 0" pos="0 0 -0.375"/>
        <geom name="geom link 0 shade" type="cylinder" size="0.05 0.375" material="green"/>
    <body name="link 1 shade" pos="0 0 0.75" euler="0 0 0">
        <joint name="joint 1 shade" type="hinge" axis="-1 0 0" pos="0 0 -0.375"/>
        <geom name="geom link 1 shade" type="cylinder" size="0.05 0.375" material="green"/>
    <body name="link 2 shade" pos="0 0 0.75" euler="0 0 0">
        <joint name="joint 2 shade" type="hinge" axis="-1 0 0" pos="0 0 -0.375"/>
        <geom name="geom link 2 shade" type="cylinder" size="0.05 0.375" material="green"/>
    </body>
</body>
</body>

<body name="obstacle 0">
    <geom name="geom obstacle 0" type="sphere" size="0.25" pos="-1.5 1 0" material="blue"/>
</body>
<body name="obstacle 1">
    <geom name="geom obstacle 1" type="capsule" size="0.1 0.2" pos="-0.75 1 0" euler="0 90 0" material="blue"/>
</body>
<body name="obstacle 2">
    <geom name="geom obstacle 2" type="ellipsoid" size="0.25 0.15 0.1" pos="0 1 0" material="blue"/>
</body>
<body name="obstacle 3">
    <geom name="geom obstacle 3" type="cylinder" size="0.1 0.2" pos="0.75 1 0" euler="0 90 0" material="blue"/>
</body>
<body name="obstacle 4">
    <geom name="geom obstacle 4" type="box" size="0.3 0.2 0.1" pos="1.5 1 0" material="blue"/>
</body>

</worldbody>

<contact>
    <pair geom1="geom link 0" geom2="geom obstacle 0"/>
    <pair geom1="geom link 0" geom2="geom obstacle 1"/>
    <pair geom1="geom link 0" geom2="geom obstacle 2"/>
    <pair geom1="geom link 0" geom2="geom obstacle 3"/>
    <pair geom1="geom link 0" geom2="geom obstacle 4"/>
    <pair geom1="geom link 1" geom2="geom obstacle 0"/>
    <pair geom1="geom link 1" geom2="geom obstacle 1"/>
    <pair geom1="geom link 1" geom2="geom obstacle 2"/>
    <pair geom1="geom link 1" geom2="geom obstacle 3"/>
    <pair geom1="geom link 1" geom2="geom obstacle 4"/>
    <pair geom1="geom link 2" geom2="geom obstacle 0"/>
    <pair geom1="geom link 2" geom2="geom obstacle 1"/>
    <pair geom1="geom link 2" geom2="geom obstacle 2"/>
    <pair geom1="geom link 2" geom2="geom obstacle 3"/>
    <pair geom1="geom link 2" geom2="geom obstacle 4"/>
    <pair geom1="geom link 0" geom2="geom link 2"/>
</contact>

<sensor>
    <framepos objtype="site" objname="tip"/>

```

```
</sensor>
</mujoco>
```