Санкт–Петербургский государственный университет

Факультет математики и компьютерных наук

Пермяков Марк Владиславович

Выпускная квалификационная работа

Методы машинного обучения в задаче планирования траектории мобильного агента

Уровень образования: бакалавриат

Направление 02.03.01 «Математика и компьютерные науки»

Основная образовательная программа СВ.5152.2020 «Математика, алгоритмы и анализ данных»

Научный руководитель:

доцент, Факультет математики и компьютерных наук, СПбГУ

к.ф.-м.н., Яковлев Константин Сергеевич

Рецензент:

доцент, НИУ ВШЭ - Санкт-Петербург,

к.ф.-м.н., Суворова Алена Владимировна

Санкт-Петербург

2024 г.

ОГЛАВЛЕНИЕ

Введение

Нахождение кратчайшего пути в графе – классическая задача, актуальная и на сегодняшний день. В век развития технологий эта задача нашла применение в самых различных сферах: транспортная навигация, робототехника, сети связи и многие другие. Особенно возрастает потребность уметь решать эту задачу для мобильных агентов. Планирование оптимальной траектории агента поможет в обеспечении безопасности, эффективности и точности перемещения.

Классический алгоритм A*, решающий эту задачу, во многом опирается на значения эвристических функций, которые зачастую являются функциями расстояния. Основная проблема этих эвристик в том, что они являются **независимыми от экземпляра**, то есть не учитывают препятствия и особенности карты, на которой осуществляется поиск.

Также, алгоритм A* обычно используют на графах, не учитывающих мобильные ограничения агента. Агент не является материальной точкой и не умеет скользить между вершинами. На этапе планирования хотелось бы принимать во внимание, что агенту придется поворачивать во время прохождения пути.

Для решения этих проблем и построения оптимальной траектории мобильного агента также могут помочь современные методы машинного обучения. Они активно используются в задаче нахождения кратчайшего пути и могут значительно улучшить поиск.

Целью данной дипломной работы является повышение эффективности планирования траектории мобильного агента. Для этого был разработан модифицированный алгоритм поиска TurnA* и создан метод построения зависимых от экземпляра эвристик на основе моделей машинного обучения. По результатам экспериментального исследования, с помощью этих эвристик удалось добиться улучшений в решении поставленной задачи.

1. Обзор

1.1. Предварительные сведения о задаче

Имеется карта местности, представляющая собой граф, и две точки: старт и цель. На этой карте есть свободное пространство, по которому агент может передвигаться, и препятствия. Через препятствия нельзя пройти или перешагнуть, можно только обойти. Мобильный агент изначально находится в вершине старт. У агента существует ориентация – направление, в котором он умеет двигаться вперед. Менять ориентацию можно с помощью ***поворотов***. В этой работе считается, что стоимость поворота (например затраченная энергия или время) такая же, как стоимость одного шага. Цель агента – добраться из старта до цели так, чтобы суммарная стоимость шагов и поворотов была оптимальной.

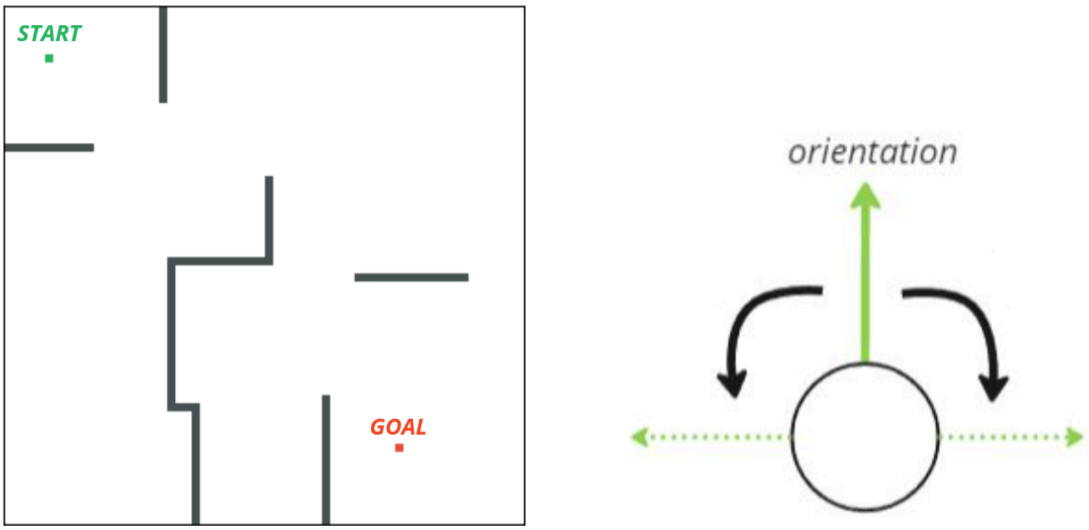

Рис.1: Карта, старт и цель. Агент с ориентацией и повороты.

1.2. Обзор литературы

Для решения задачи нахождения кратчайшего пути существует множество различных методов:

Методы на основе систематического поиска:

Классический метод решения задачи нахождения кратчайшего пути в взвешенном графе – **алгоритм Дейкстры** [1]. Этот алгоритм запоминает множество вершин, для которых уже вычислены длины кратчайших путей от старта. Затем выбирается вершина не из этого множества, которой на текущий момент соответствует минимальная оценка кратчайшего пути. Эта вершина добавляется во множество посещенных, после чего у ее соседей происходит пересчет оценок кратчайшего пути.

Алгоритм A* [2], по сути, является расширением алгоритма Дейкстры. Теперь для выбора очередной вершины учитывается не только оценка кратчайшего пути от старта, но и эвристическое приближение расстояния до цели. Эти два числа складываются и дают значение f-*функции*, по которой будут определяться вершины для дальнейшего раскрытия.

Часто используемый на практике алгоритм **взвешенный A* (WA*)** [3] использует размен оптимальности на эффективность вычислений следующим способом: запускается алгоритм A*, но во время вычисления f-функции значения эвристического приближения умножаются на некоторую константу $w \geq 1$, что может позволить быстрее добраться до цели и найти решение, превосходящее оптимальное не более чем в w раз. Также хорошо известен алгоритм **Focal Search (FS)** [4], в котором также задается некоторый вес w и в котором для раскрытия выбираются вершины, у которых оценка f-функции хуже оптимальной не более чем в w раз. Во время поиска зачастую используется второстепенная эвристика, которую можно получить, обладая какими-то начальными знаниями о карте.

Двусторонний алгоритм, **Bidirectional A*** [5], запускает 2 параллельных A* от старта до цели и от цели до старта, что может позволить быстрее найти

оптимальное решение. Алгоритм **Iterative Deepening A* (IDA*)** [6] – модификация поиска в глубину, итеративно увеличивающего свою глубину, которая заимствует идею эвристических функций ради оценки расстояния до цели. В отличие от A*, IDA* не использует динамическое программирование, что может приводить к повторным раскрытиям вершин, но зато, как правило, использует меньше памяти.

Из более недавних результатов стоит отметить алгоритм **Theta*** [7]. Этот алгоритм ищет кратчайший путь на карте местности, в котором, в отличие от A*, из клетки напрямую можно попасть не только в своего соседа, но в любую клетку, на пути к которой нет препятствий. Такой путь называется *под любым углом*. Еще одно улучшение A* – алгоритм **Jump Point Search (JPS)** [8], который вводит понятие прыжковых точек – точек, между которыми можно добираться с помощью прыжков. Приемники, то есть вершины, в которые мы можем прыгнуть, для прыжковой точки определяются с помощью специальных правил отсечений соседей.

Методы на основе сэмплирования:

Данная группа методов не требует полного изучения пространства поиска. Вместо этого, они случайно выбирают и соединяют какое-то число точек пространства, создавая граф без препятствий. Два самых известных алгоритма, использующих эту идею, это **PRM (Probabilistic Roadmap)** [9] и **RRT (Rapidly exploring random trees)** [10]. Их основное отличие заключается в построении графа поиска: PRM – это алгоритм множественного запроса, то есть, создав граф, PRM использует его для любой пары старт и цель. RRT же, в свою очередь, алгоритм одного запроса, то есть он создает отдельный граф на каждую пару старт и цель. Оба этих алгоритма являются *вероятностно полными*, то есть вероятность не найти существующее решение стремится к 0 при увеличении числа сэмплируемых точек. Оптимальные версии этих алгоритмов, **PRM*** и **RRT*** [11], были придуманы позднее, и за счет модификаций построения графов поиска эти алгоритмы обеспечивают

асимптотическую оптимальность: найденные пути сходятся почти наверное к оптимальным при увеличении числа сэмплируемых точек.

Одним из *state-of-the-art* методов этой группы является алгоритм **Batch Informed Trees (BIT*)** [12]. Основная его идея в том, чтобы сэмплировать точки и вести поиск некоторыми партиями – батчами, каждый раз находя все лучшее решение. Этот алгоритм также обеспечивает асимптотическую оптимальность и показывает более сильные результаты по сравнению с прошлыми методами.

Методы этой группы нашли широкое применение в робототехнике за счет своей возможности эффективно работать с непрерывным пространством поиска.

Методы машинного обучения:

Развитие машинного обучения и нейронных сетей привело к появлению новых методов и идей для задачи нахождения кратчайшего пути. В работе (Takahashi et al. 2019; Ariki and Narihira 2019) [13] был предложен метод использования сверточных нейронных сетей для обучения эвристики, предсказывающей точное расстояние от каждой вершины до цели. Алгоритм **RETRO*** [14] разработан для нахождения реакций синтеза продукта в органической химии. Поиск ведется на *AND-OR* деревьях и направляется обученной нейронной сетью. В работе (Yonetani et al. 2021) был предложен алгоритм **NeuralA*** [15], использующий дифференцируемый A* и *Guidance Maps* – карты, на которых каждой клетке сопоставлено число, которое можно интерпретировать как стоимость прохождения через вершину. Более общие идеи решения задач комбинаторной оптимизации с помощью методов машинного обучения были представлены в работе (Li, Chen, and Koltun 2018) [16].

Другое популярное направление машинного обучения – методы имитационного обучения (IL). Их суть заключается в том, чтобы обучить агента некоторой оптимальной политике действий, где в роли учителя выступает оракул – человек или некоторые сгенерированные истинные значения. В 2017 году в работе (Mohak Bhardwaj, Sanjiban Choudhury, Sebastian Scherer) был

предложен алгоритм **SaIL** [17], а позднее в этом году этими же авторами была написана подробная статья [18], изучающая текущие на тот момент методы IL. (Edward Groshev et al, 2017) [19] обучали имитационную политику с помощью глубоких нейронных сетей, а (Leah Chrestien et al, 2021) [20] использовали механизм внимания. В 2022 году был представлен алгоритм **PHIL** [21] с новой архитектурой нейронной сети, алгоритмом обучения и использующий методы *graph representation learning*, который обогнал на тестах прочие современные алгоритмы имитационного обучения.

У описанных методов есть свои недостатки для решения нашей задачи: взвешенные алгоритмы жертвуют своей оптимальностью; методы на основе сэмплирования применяются к непрерывным задачам; методы IL вынуждены обучать политику агента на каждой новой карте, хотя в нашей задаче предполагается последовательный процесс принятия решений. В этой работе будет предложен алгоритм, использующий эвристику, предсказанную моделью на основе архитектуры трансформера, а также учитывающий повороты мобильного агента.

2. Формальная постановка задачи

Карта (grid map) размером H x W – декартово произведение $[0, 1, ..., H - 1] \times [0, 1, ..., W - 1]$, состоящее из свободных вершин и вершин-препятствий. **Свободные вершины** – это вершины, по которым может передвигаться агент и которые задаются как: $\{(x, y) \mid GridMap[x, y] = 0\}$. **Препятствиями** же являются вершины, по которым нельзя передвигаться: $\{(x, y) \mid GridMap[x, y] = 1\}$

Старт и цель – это две свободные вершины *start* и *goal* с координатами $(start_x, start_y)$ и $(goal_x, goal_y)$ соответственно. Агент находится в вершине *start* и ему нужно попасть в *goal*.

В классической задаче каждая вершина задается двумя координатами (x, y), в нашей же задаче появляется новая характеристика: **ориентация**. Агент находится в вершине (x, y, ori) тогда и только тогда, когда он находится в координатах (x, y) и смотрит в направлении ori. В данной работе считается, что $ori \in \{[0, 1], [1, 0], [0, -1], [-1, 0]\}$, что соответствует направлениям “вниз”, “вправо”, “вверх” и “влево”.

Соседи вершины (x, y, ori) определяются следующим образом: $Neighbours((x, y, ori)) = \{(x, y, ori_{\curvearrowleft}), (x, y, ori_{\curvearrowright}), (x + ori[0], y + ori[1], ori)\}$, что означает, что мы можем либо повернуться по/против часовой стрелки, либо сделать шаг в сторону ориентации.

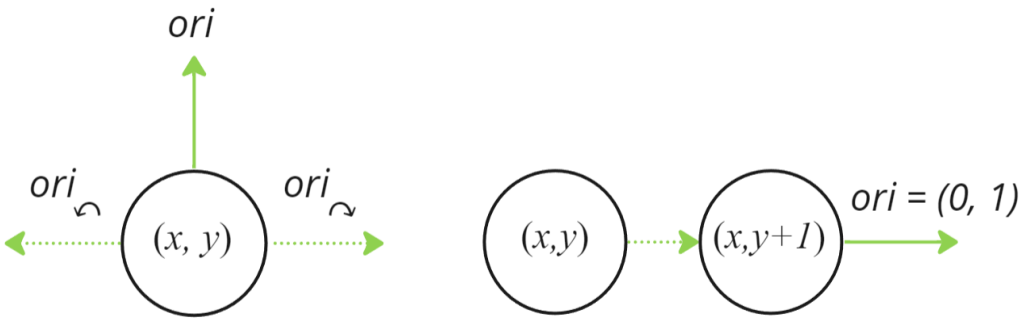

Рис.2: Соседи вершины (x, y, ori)

Стоимость перемещения в соседнюю вершину будем считать равной 1, как для шага так и для поворота. Перейти в соседнюю вершину можно только если она является свободной.

Путь π(start, goal) – последовательность свободных соседних вершин, начиная от старта и заканчивая целью:

$$\pi(start,\ goal)\ =\ (v_0\ =\ start,\ v_1,\ v_2,\ \ldots,\ v_n\ =\ goal)$$

Стоимость пути – сумма стоимостей всех переходов между соседними вершинами в пути:

$$cost(\pi)\ =\ \sum_{i=0}^{n-1} cost(v_i,\ v_{i+1})$$

Пусть П – набор всех путей, от старта до цели. Тогда путь $\pi^* \in П$ называется **кратчайшим**, если $\forall\pi \in П$: $cost(\pi^*) \leq cost(\pi)$.

Каждая задача определяется тройкой $(GridMap,\ start,\ goal)$, в которой надо найти кратчайший путь $\pi^*(start,\ goal)$. Если такой путь существует, то он будет **оптимальным**.

Также, для $\omega > 1$ введем понятие **субоптимального** решения, стоимость которого превосходит стоимость оптимального не более чем в ω раз:

$$\pi^{\omega} \in П: \quad cost(\pi^{\omega}) \leq \omega \cdot cost(\pi^*)$$

3. Методы

3.1. Алгоритм A*

Алгоритм A* – алгоритм поиска, который находит во взвешенном графе путь наименьшей стоимости от начальной вершины до выбранной конечной. A* в ходе работы строит дерево поиска, где каждая вершина соответствует клетке карты. Каждая такая вершина хранит в себе следующую информацию о клетке: ее координаты, указатель на своего предка, а также значения функций $f(v)$, $g(v)$, $h(v)$, где $g(v)$ – наименьшая стоимость пути из стартовой вершины в v, $h(v)$ – эвристическое приближение стоимости пути от v до конечной цели и $f(v) = g(v) + h(v)$.

Алгоритм генерирует новые вершины путем последовательного ***раскрытия*** вершин. Раскрыть вершину – значит сгенерировать ее допустимых соседей, посчитать у них значение функции f и добавить их в дерево поиска. Список сгенерированных вершин, которые еще не раскрыты принято называть $OPEN$, а список уже раскрытых вершин – $CLOSED$.

Алгоритм начинает с вершины $start$, которую изначально добавляют в $OPEN$. Если у вершины v_1 есть сосед v_2, то v_1– **предок** вершины v_2, тогда значение функции $g(v_2)$ считается как $g(v_2) = min(g(v_2), g(v_1) + cost(v_1, v_2))$.

Эвристической функцией, как правило, выступает некоторое расстояние: евклидово, Манхэттена, октильное и т.д. Эвристическая оценка $h(v)$ всегда считается от вершины до цели, поэтому второй аргумент опускается: $h(v) = h(v, goal)$.

Таким образом, сгенерировав соседа, мы считаем в нем значение функции f и добавляем в дерево поиска, то есть в список $OPEN$. Отметим, что одна и та же вершина добавляется в поиск либо если ее там еще нет, либо если она есть, но с большим значением функции f, чем у новой посчитанной.

Очередная вершина для раскрытия выбирается из списка $OPEN$ и имеет минимальное значение функции f. После того как вершина раскрыта, она добавляется в $CLOSED$. Алгоритм работает до тех пор, пока не выберет для раскрытия вершину $goal$, либо пока список $OPEN$ не пуст, в случае если пути между $start$ и $goal$ не существует.

Рассмотрим, как эвристики влияют на нахождение оптимального пути. Эвристика h^* называется ***идеальной***, если $\forall\, v:\ h^*(v) = cost(\pi^*(v,\ goal))$. Эвристика h называется ***допустимой***, если она никогда не переоценивает идеальную: $\forall\, v:\ h(v) \leq h^*(v)$. Эвристика h называется ***монотонной***, если $\forall\, v_1,\ v_2:\ h(v_1) \leq h(v_2) + cost(\pi^*(v_1,\ v_2))$.

Известно, что алгоритм A* с допустимой эвристикой гарантированно найдет оптимальное решение, если оно существует. Более того, если эвристика монотонна, то алгоритм найдет кратчайшие расстояния для всех раскрытых вершин, что означает, что никакая вершина не будет раскрыта более одного раза.

3.2. WA*

Взвешенный A* отличается лишь одной деталью. Выбирается некоторый постоянный вес $w \geq 1$, и значения функции f теперь считаются как $f(v) = g(v) + w \cdot h(v)$, то есть используются взвешенные эвристики. Благодаря такой модификации, алгоритм в приоритете раскрывает вершины, которые, по оценкам эвристики, находятся ближе к цели. На практике этот алгоритм уменьшает число раскрытых вершин, но найденный путь оказывается длиннее оптимального. Известно, что алгоритм WA* с допустимой эвристикой находит субоптимальное решение с коэффициентом w.

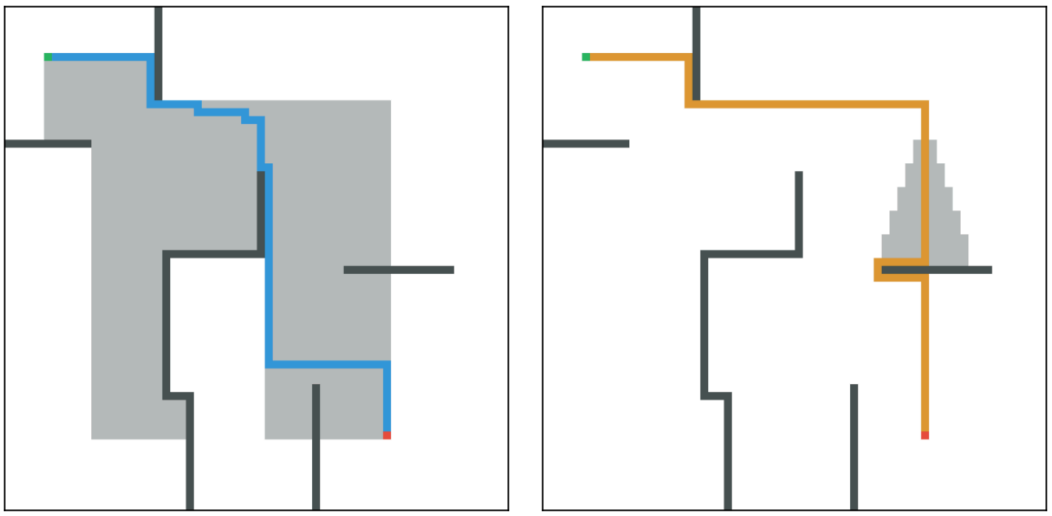

Рис.3: Пример работы A* (слева) и WA*, w=2 (справа), решающих задачу (x, y). Зеленая и красная клетки – старт и цель, серые клетки – раскрытые вершины. Эвристика – расстояние Манхэттена.

Algorithm 1 Классический A* и WA*

Input: Grid map *Map*, *start* node, *goal* node,
heuristic h, weight w
Output: path π(start, goal)

```
1:  OPEN := {start}, CLOSED := {}
2:  while OPEN not empty do
3:      v = getBestNodeFromOpen()
4:      OPEN.remove(v)

5:      if v == goal then
6:          return makePath(v)

7:      CLOSED.add(v)
8:      neigbours = Map.getNeighbours(v)
9:      for n in neigbours do
10:         if g(n) > g(v) + cost(v, n) then
11:             g(n) = g(v) + cost(v, n)
12:             f(n) = g(n) + w · h(n)
13:             OPEN.update(n)          ▷ Either add or update f value
14: return path does not exist
```

Псевдокод 1: A* при w = 1, WA* при w > 1

3.3. Focal Search и GBFS

Focal Search – еще один алгоритм, позволяющий разменять оптимальность решения на стоимость вычислений. Основная его идея заключается в введении нового списка вершин $FOCAL$, в котором хранятся вершины немногим хуже чем в $OPEN$:

$FOCAL = \{v : v \in OPEN, f(v) \leq f_{min} \cdot w\}$, где f_{min} – минимальное значение f-функции среди вершин из $OPEN$, а $w \geq 1$ – выбранный постоянный вес.

Теперь вершина для раскрытия берётся не из $OPEN$, а из $FOCAL$, причем выбирается вершина с наименьшим (или наибольшим) значением второстепенной эвристики h_{FOCAL}. Она может быть не монотонной и даже не допустимой. Если $OPEN$ в результате раскрытия меняется, то пересчитывается и $FOCAL$ согласно определению. Алгоритм Focal Search является субоптимальным с коэффициентом w.

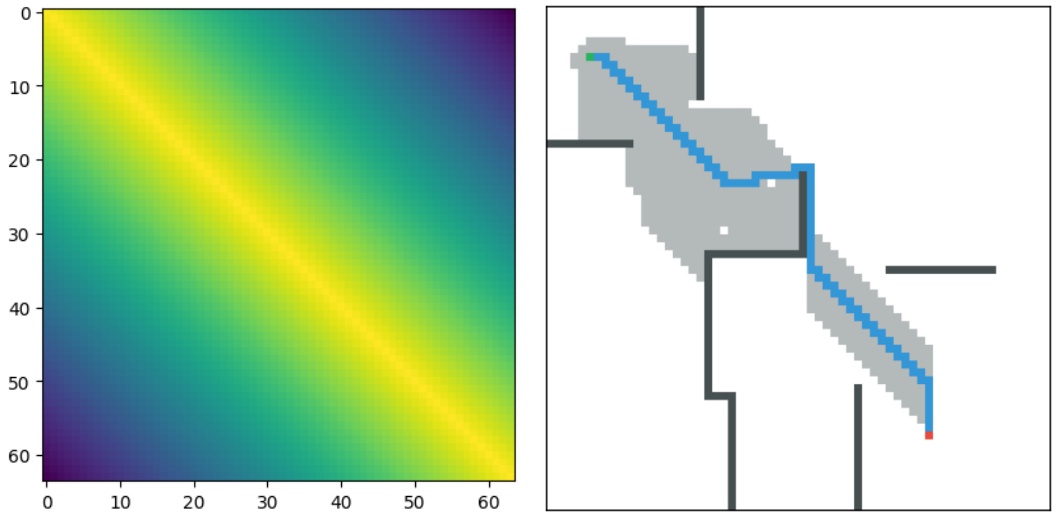

Рис. 4: Пример работы Focal Search с w=2, решающего задачу (x, y). Слева – второстепенная эвристика h_{FOCAL}, чем светлее клетка, тем больше в ней значение h_{FOCAL}. Справа – найденный путь и множество раскрытых вершин.

Algorithm 2 Focal Search

Input: Grid map *Map*, *start* node, *goal* node,
heuristic h, secondary heuristic h_{FOCAL}, weight w
Output: path π(start, goal)

```
1: OPEN := {start}, FOCAL := {start}, CLOSED := {}
2: while FOCAL not empty do
3:     f_min = f(getBestNodeFromOpen())
4:     v = getBestNodeFromFocal()                    ▷ w.r.t h_FOCAL
5:     OPEN.remove(v)
6:     FOCAL.remove(v)

7:     if v == goal then
8:         return makePath(v)

9:     CLOSED.add(v)
10:    neigbours = Map.getNeighbours(v)
11:    for n in neigbours do
12:        if g(n) > g(v) + cost(v, n) then
13:            g(n) = g(v) + cost(v, n)
14:            f(n) = g(n) + w · h(n)
15:            OPEN.update(n)
16:            if f(n) ≤ w · f_min then
17:                FOCAL.add(n)
18:    if OPEN not empty and f_min < f(getBestNodeFromOpen()) then
19:        updateFocal()
20: return path does not exist
```

Псевдокод 2: Focal Search

Greedy Best First Search (GBFS) – это, по сути, алгоритм FS с коэффициентом $w = +\infty$. Грань между $FOCAL$ и $OPEN$ стирается, и алгоритм жадно выбирает вершины с лучшим значением эвристики h_{FOCAL}. Этот алгоритм не может гарантировать какой-либо оптимальности найденного решения.

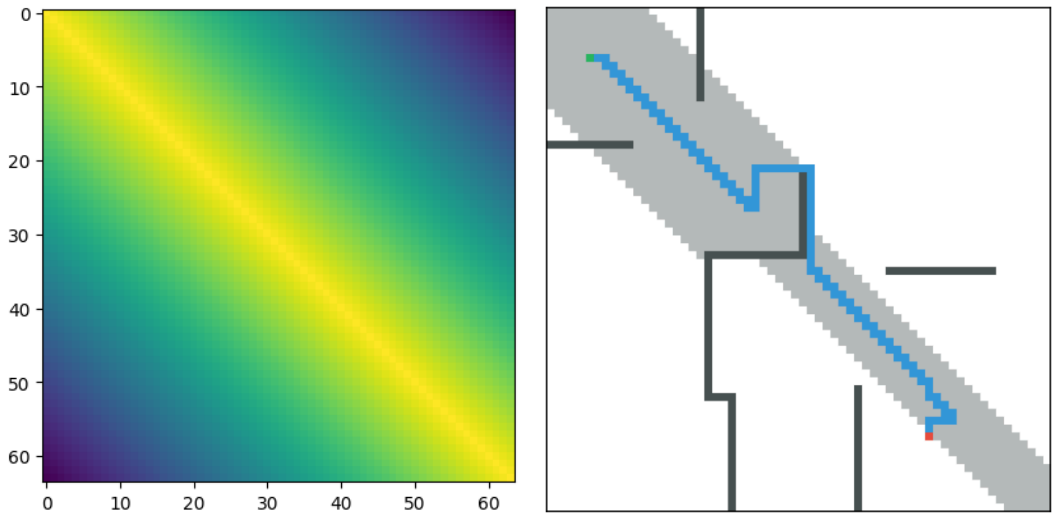

Рис. 5: Пример работы GBFS, решающего задачу (x, y). Из-за своей жадности, прежде чем дойти до цели, алгоритм сначала добрался до правого нижнего угла, так как значения второстепенной эвристики там были больше.

3.4. TurnA*

Алгоритм TurnA* – это предложенная для этой работы модификация A*, учитывающая повороты агента. В классическом A* каждая вершина дерева поиска задавалась своими координатами (x, y), теперь же каждая из 4 ориентаций будет задавать новую вершину (x, y, ori). Отличия будут лишь в нахождении соседей для вершины и в подсчете стоимостей для перемещения в соседей. Благодаря введенной стоимости на смену ориентации, найденный путь будет оптимален относительно суммарной стоимости пройденных шагов и поворотов.

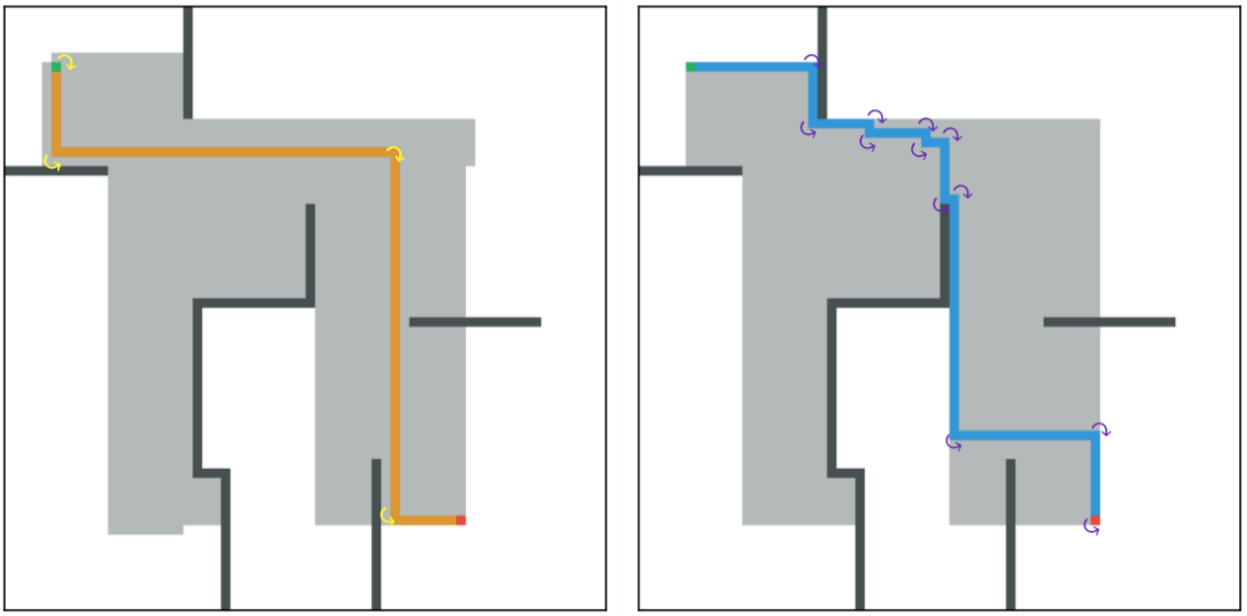

Рис. 6: Пример работы TurnA* (слева), решающего задачу (x, y, ori) по сравнению с A* (справа), решающего задачу (x, y). Ориентация в старте – “вправо”, в цели – “вправо”. Стрелками показаны повороты на пути агента. В TurnA* получилось 4 поворота, а в A* – 12.

Отметим, что это не означает, что в найденном пути будет минимальное число поворотов, необходимое для достижения цели:

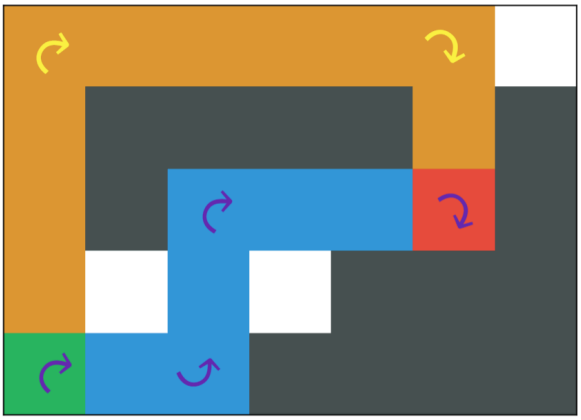

Рис. 7: Ориентация в старте – “вверх”, ориентация в цели – “вниз”. Как видим, в синем пути 4 поворота, в то время как в оранжевом всего 2. Но стоимость синего пути: 4 turns + 7 steps = 11, а оранжевого: 2 turns + 11 steps = 13.

3.5. PGM и FocalTurnA*

В алгоритме Focal Search для выбора очередной вершины для раскрытия используется второстепенная эвристика h_{FOCAL}. Идеальная эвристика могла бы

вести планировщик по вершинам, лежащим на оптимальном пути: в них значение $h_{FOCAL}(v)$ было бы высоким, а в остальных вершинах низким. Эта мысль привела к введению понятия **Path Guidance Map (PGM)** – эвристики, отражающей уверенность в том, что вершина лежит на оптимальном пути. PGM измеряет уверенность значениями от 0 до 1, где 0 можно интерпретировать как: “эта вершина точно не лежит на оптимальном пути”, а 1 как: “эта вершина точно лежит на оптимальном пути”.

Для нашей задачи с поворотами одной карты PGM будет недостаточно, ведь каждая вершина задается тройкой (x, y, ori), и PGM значения одной клетки в разных ориентациях могут сильно отличаться. Для этого нужна 4-х канальная PGM, по одному каналу на каждое возможное направление. Предложенный для этой работы алгоритм FocalTurnA* – это модификация Focal Search, которая использует в качестве второстепенной эвристики h_{FOCAL} эти 4-х канальные PGM:

$h_{FOCAL}((x, y, ori)) = PGM[ch_{ori}, x, y]$, где ch_{ori} – канал, соответствующий направлению ori.

Именно эти 4-х канальные PGM будут предсказываться моделью и использоваться в качестве второстепенной эвристики в алгоритме FocalTurnA*.

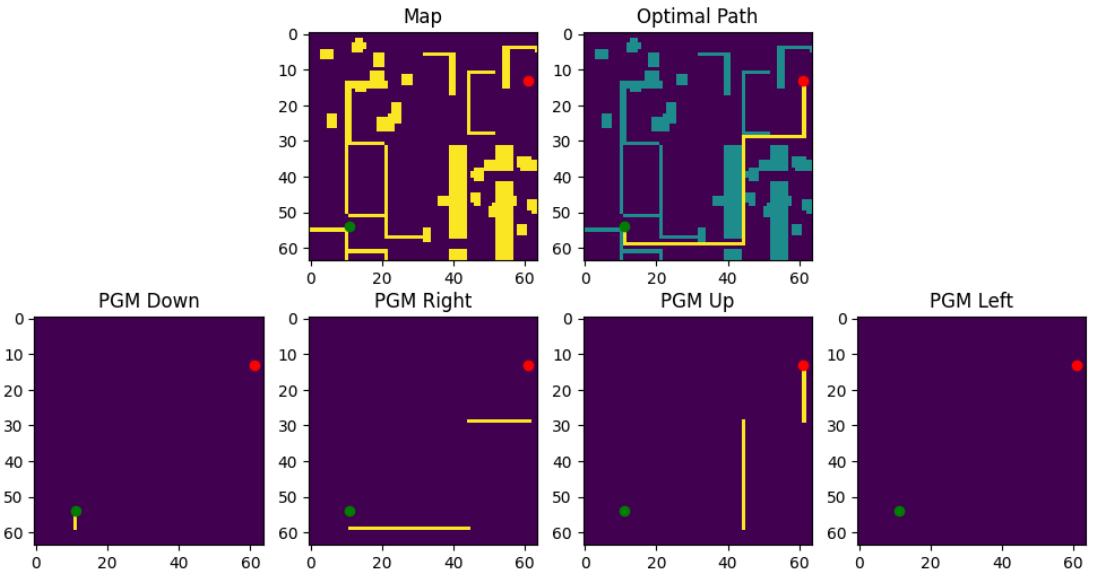

Рис. 8: Пример “идеальной” 4-х канальной PGM.

4. Обучение

4.1. Архитектура модели

За основу взята модель из работы (TransPath, 2022) [22]. Она состоит из 3 последовательных блоков: Encoder, Transformer, Decoder.

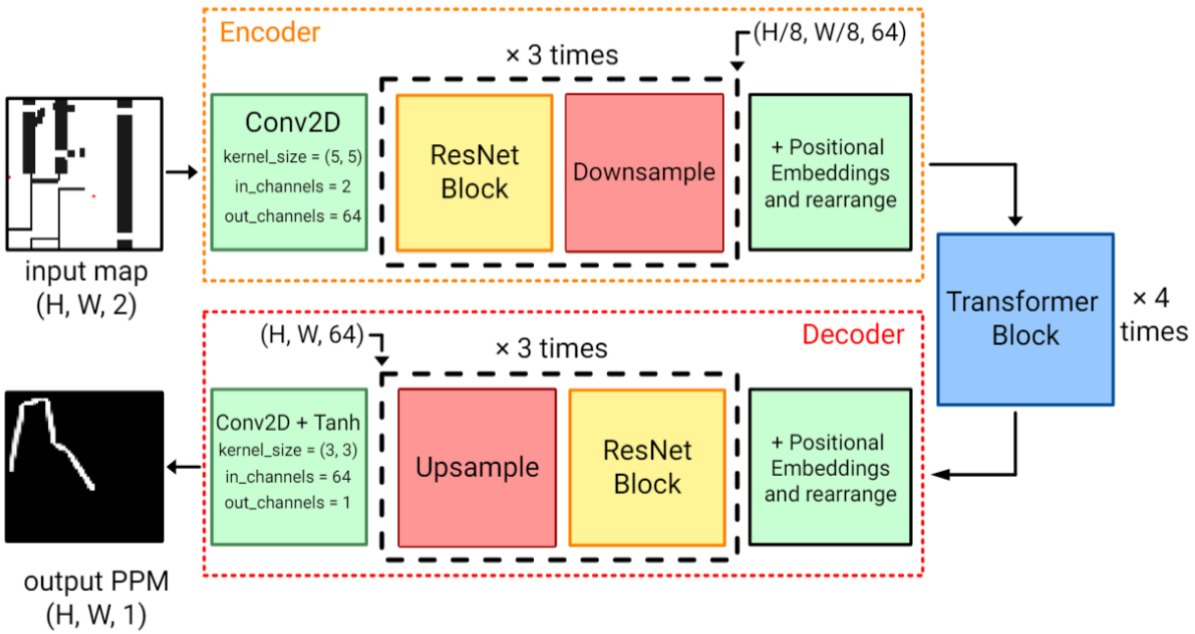

Рис. 9: Архитектура модели. Рисунок авторов модели TransPath.

Авторы статьи решали другую задачу, в связи с чем модель была адаптирована для наших целей.

В модели TransPath на вход подаются 2 матрицы размера (H x W): карта, где свободные клетки имеют значение 0, а препятствия значение 1 и карта, на которой в клетках старта и цели стоят значения 1, а на всех остальных клетках 0. Выходом является одна матрица размера (H x W), которую авторы использовали в качестве второстепенной эвристики.

В нашей же модели на вход будут подаваться 3 матрицы размера (H x W): карта с препятствиями; матрица, в которой в клетке старта стоит 1, а во всех остальных клетках 0; и аналогичная матрица цели. Выходом же будут являться

PGM размера (H x W x 4).

Первый блок Encoder принимает вход и сначала пропускает его через *сверточный слой* [23]. После этого идет 3 остаточных блока с *остаточным сверточным слоем*, ResNet [24], идея которого заключается в том, чтобы учитывать информацию не только с выходов блока, но и с его входов. Для этого выходные данные слоев складывают со входными данными блока, образуя тем самым выходы остаточного блока. Внутри него присутствуют слои нормализации по группам [25], которые ускоряют обучение и функция активации “взмах” [26], которая добавляет нелинейности в модель.

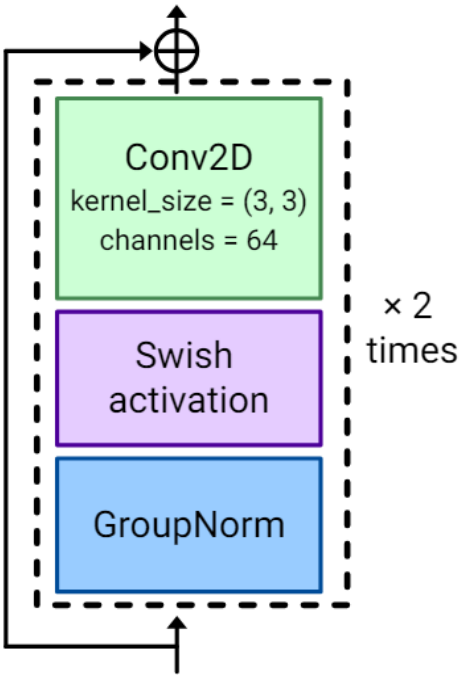

Рис. 10: Блок ResNet. Рисунок авторов модели TransPath.

Также, после каждого остаточного блока происходит операция *Downsample* – уменьшение размерности наших данных с целью более оптимальных вычислений. В нашей задаче остаточные блоки помогают извлечь некоторые особенности карты, например, узкие проходы, углы и т.д. Последний слой блока – это эмбеддинги – преобразование выходов Encoder-а в набор векторов, который пойдет на вход трансформера.

Блок трансформера использует механизм *внутреннего внимания* (*self-attention*) [27], который успел зарекомендовать себя как сильный инструмент в сфере обработки естественного языка и компьютерного зрения. Он помогает выяснить, насколько одни особенности карты важны относительно других. Также, внутри этого блока происходит нормализация слоев [28] для уменьшения времени обучения модели.

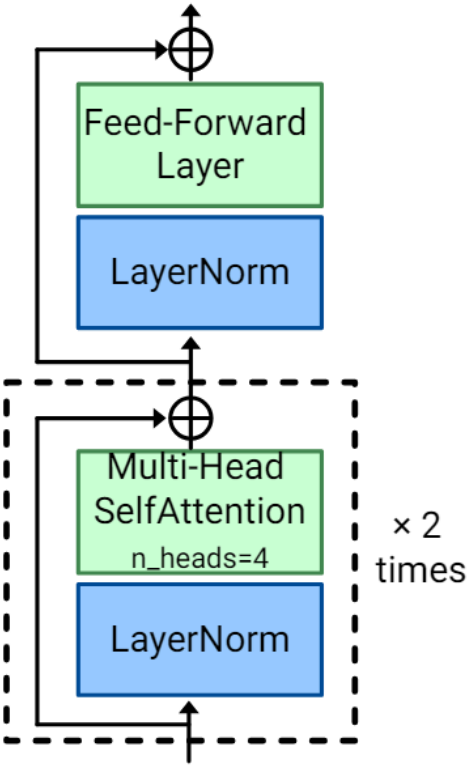

Рис. 11: Блок трансформера. Рисунок авторов модели TransPath.

Выходы трансформера передаются в Decoder, который первым делом снова производит эмбеддинги, чтобы преобразовать векторы обратно в двумерное пространство. Затем, снова проходит несколько остаточных блоков, но уже с операцией *Upsample* – добавлением размерности к данным с целью возвращения к исходному размеру, который был потерян в Encoder-е. Последний сверточный слой в Decoder-е имеет функцию активации гиперболический тангенс, который выдает результат модели.

4.2. Датасет

Для обучения модели необходимо сгенерировать датасет. Это включает в себя три пункта: карты, старты и цели, истинные значения PGM (*ground truth* или *gt*). Все карты в этом датасете будут размера 64x64.

Карты были взяты из датасета TMP (Tiled Motion Planning), который был использован в работе (Yonetani et al. 2021) [15]. В этой работе сначала были сгенерированы карты размера 32x32 с разными топологиями местности:

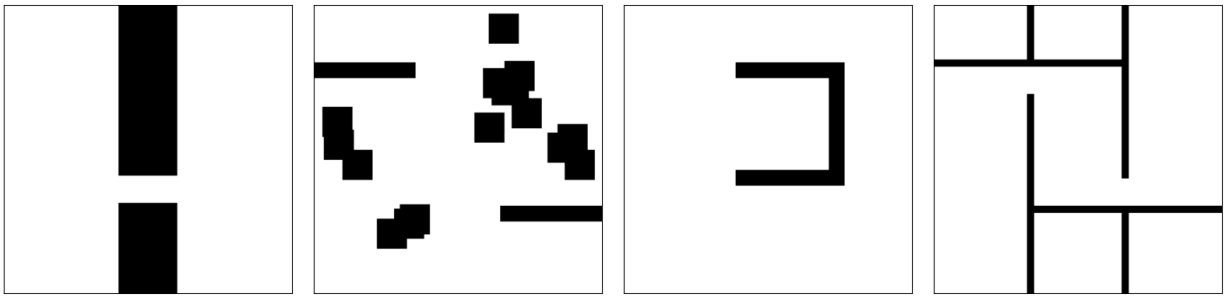

Рис. 12: Примеры различных топологий. Слева направо: зазор, лес, ловушка, лабиринт.

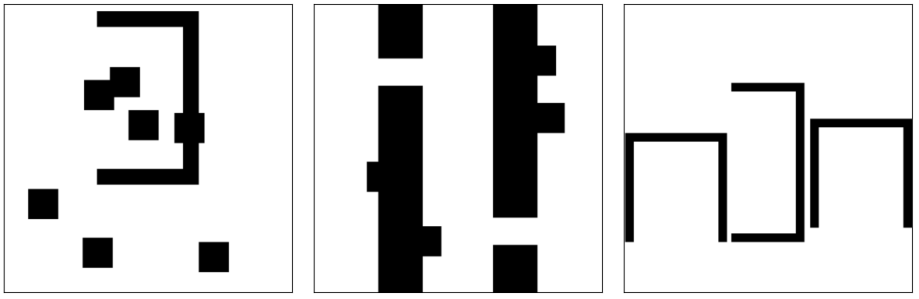

Рис. 13: Примеры комбинаций топологий. Слева направо: лес и ловушки, лес и зазоры, несколько ловушек.

Затем, чтобы составить одну карту для TMP размером 64x64, случайно берутся 4 карты 32x32 и соединяются вместе:

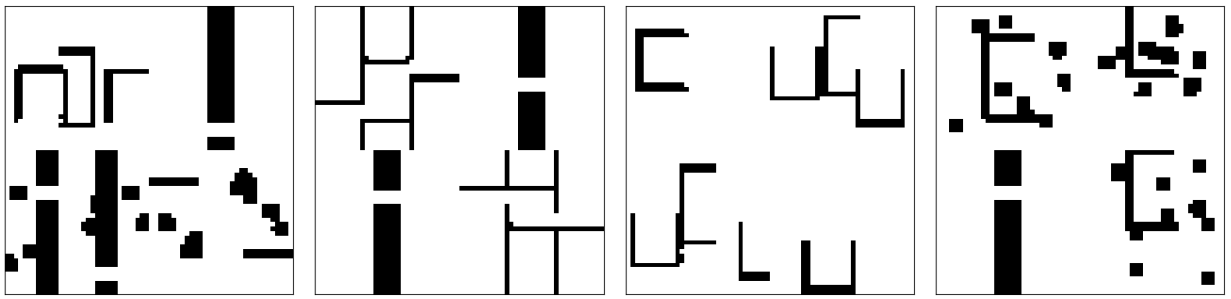

Рис. 14: Примеры сэмплов из датасета TMP. Размер 64x64.

Для каждой карты необходимо сгенерировать старт и цель, и для этого был разработан следующий алгоритм. Сначала с равными вероятностями выбирается четверть, в которой будет находиться старт. Затем, в этой четверти выбирается произвольная свободная клетка, которая объявляется стартом. После этого запускается алгоритм TurnA*, который находит стоимости путей от старта до всех свободных вершин на карте, в которые он может попасть. Чтобы это сделать, нужно запустить TurnA*, в котором единственным критерием остановки будет пустота списка *OPEN*. Среди всех вершин, в которые мы можем попасть из старта, берется 20% с самой большой стоимостью пути. Произвольная вершина из этих 20% объявляется целью.

Рис. 15: Стоимости путей от случайно выбранного старта до каждой вершины, 20% самых дальних вершин и конечная карта со стартом и целью.

Наконец, чтобы сгенерировать gt-значения для PGM, был разработан следующий алгоритм. Бралась задача $(GridMap, start, goal)$. Для нее составлялся список всех кратчайших путей по следующему принципу: находился кратчайший путь π*, затем, все клетки на этом пути, кроме старта и цели, становились препятствиями, и на этой новой карте снова искался кратчайший путь. Если алгоритм нашел путь π', причем $cost(\pi') = cost(\pi^*)$, то π' также включался в список кратчайших путей и процедура повторялась. Для каждого пути из составленного списка смотрелось, в какой ориентации

находится каждая его вершина, после чего она добавлялась в соответствующий канал нашей gt PGM. Вершины $start$ и $goal$ добавлялись во все каналы.

Algorithm 3 Генерация gt-PGM

Input: Grid map Map, $start$ node, $goal$ node, heuristic h
Output: Path Guidance Map

$ShortestPaths := [\,]$
$MinCost := +\infty$
$NewSolutions := True$
$Map' := Map$
$PGM = Zeroes(4 \times 64 \times 64)$ ▷ Matrix of zeroes with size 4x64x64
$PGM[:][start.x, start.y] = 1$
$PGM[:][goal.x, goal.y] = 1$ ▷ All channels
while $NewSolutions$ is True **do**
    $\pi := TurnA^*(Map', start, goal, h)$
    if π is None **then** ▷ Path does not exist
        $NewSolutions = False$
        continue
    if $cost(\pi) \leq MinCost$ **then**
        $MinCost = cost(\pi)$
        $ShortestPaths.add(\pi)$
        for $node$ in π **do** ▷ Except $start$ and $goal$
            $Map'[node.x, node.y] = obstacle$
            $PGM[node.ori][node.x, node.y] = 1$
return PGM

Псевдокод 3: Генерация gt PGM.

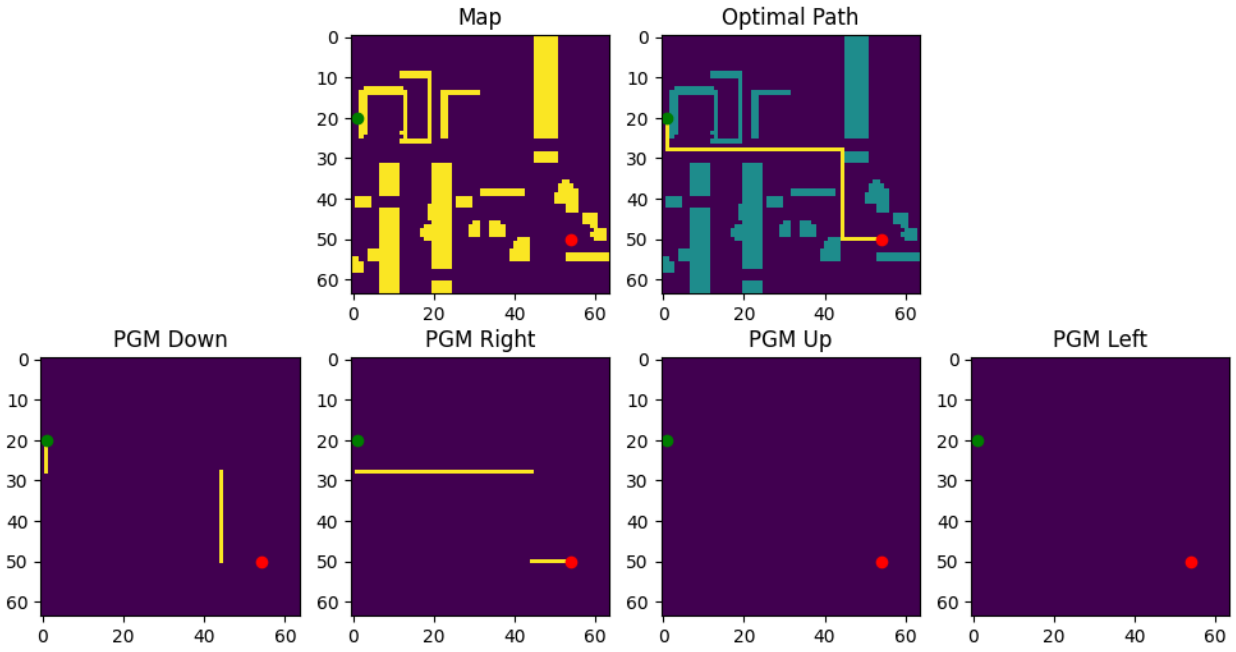

Рис. 16: Пример gt PGM с одним кратчайшим путем.

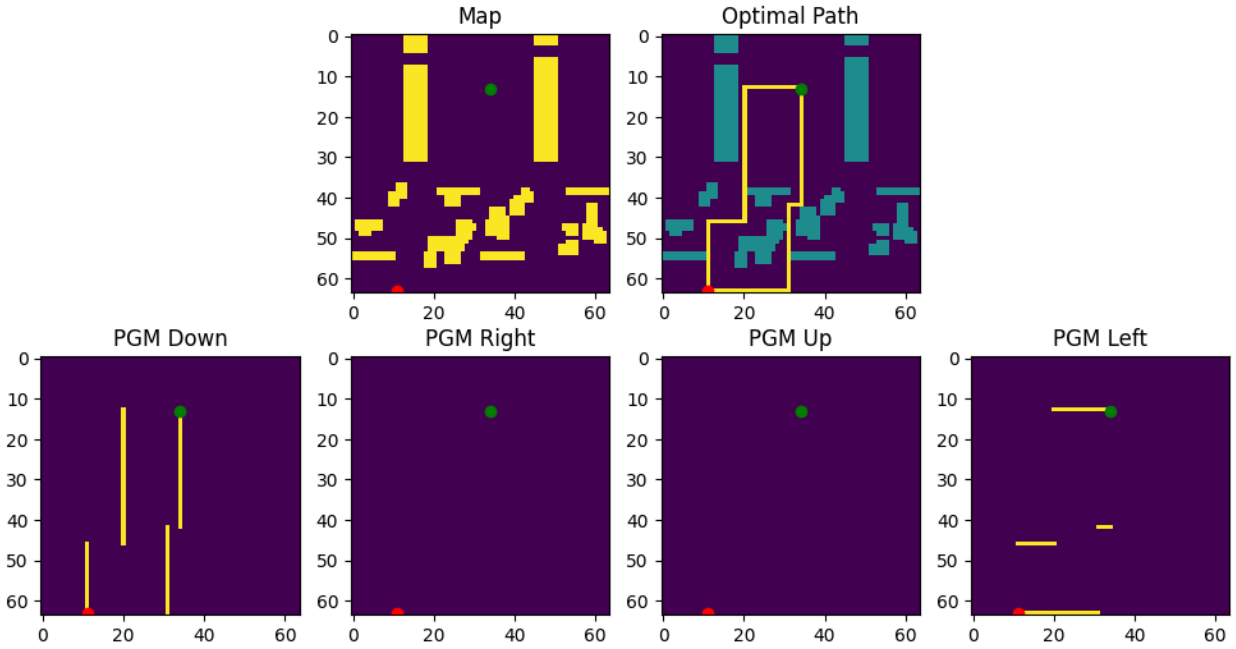

Рис. 17: Пример gt PGM с двумя кратчайшими путями.

Всего было использовано 64000 различных карт и для каждой из них сгенерировано 10 пар $(start, goal)$, таким образом размер датасета равен 640000 сэмплов. Разделение на обучающую, валидационную и тестовую выборку происходило в отношении 8:1:1.

Также, в этой работе будет обучена модель, предсказывающая PGM для карт размера 128x128. Чтобы создать датасет, все сгенерированные карты размера 64x64 были увеличены в 2 раза, а старты, цели и кратчайшие пути были отображены в соответствующие новые клетки:

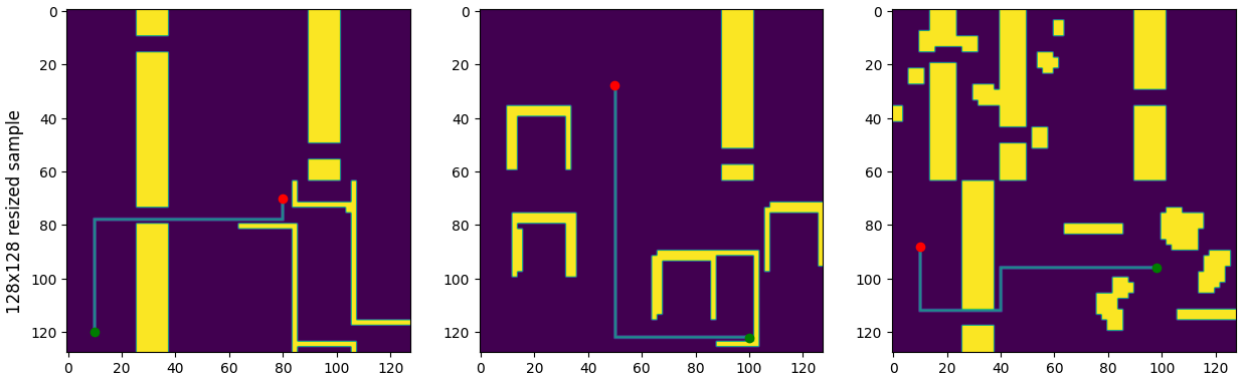

Рис. 18: Примеры увеличенных сэмплов до размера 128x128.

4.3. Сетап и обучение

Модели, предсказывающие PGM для карт размера 64x64 и 128x128 назовем **PGM_64 и PGM_128** соответственно. Обучение было написано на языке Python с помощью фреймворка PyTorch и библиотеки PyTorchLightning. PGM_64 обучалась 55 эпох, а PGM_128 30 эпох, в силу того что в ней больше весов. В обеих моделях использовался оптимизатор Adam [29] с шагом обучения $\lambda = 0.001$. В качестве функции потерь была выбрана средняя квадратичная ошибка MSE. Также, каждую эпоху подсчитывалась функция потерь на валидационной выборке, которая не участвовала в обучении:

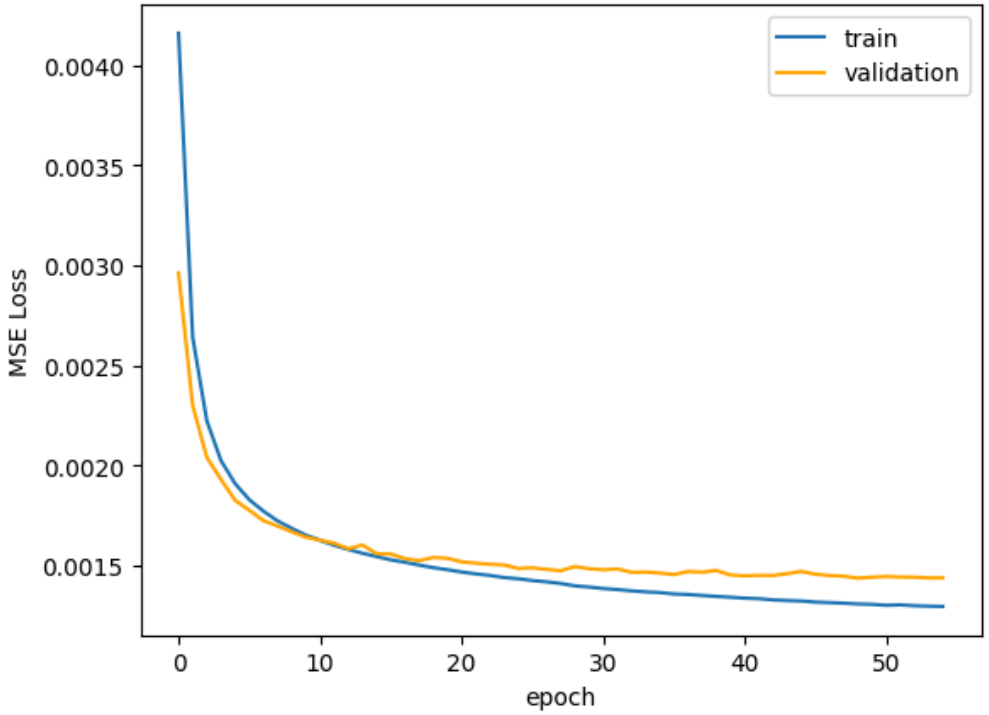

Рис. 19: График обучения модели PGM_64. Уменьшение функции потерь на обучающей выборке (синяя кривая) и валидационной (оранжевая).

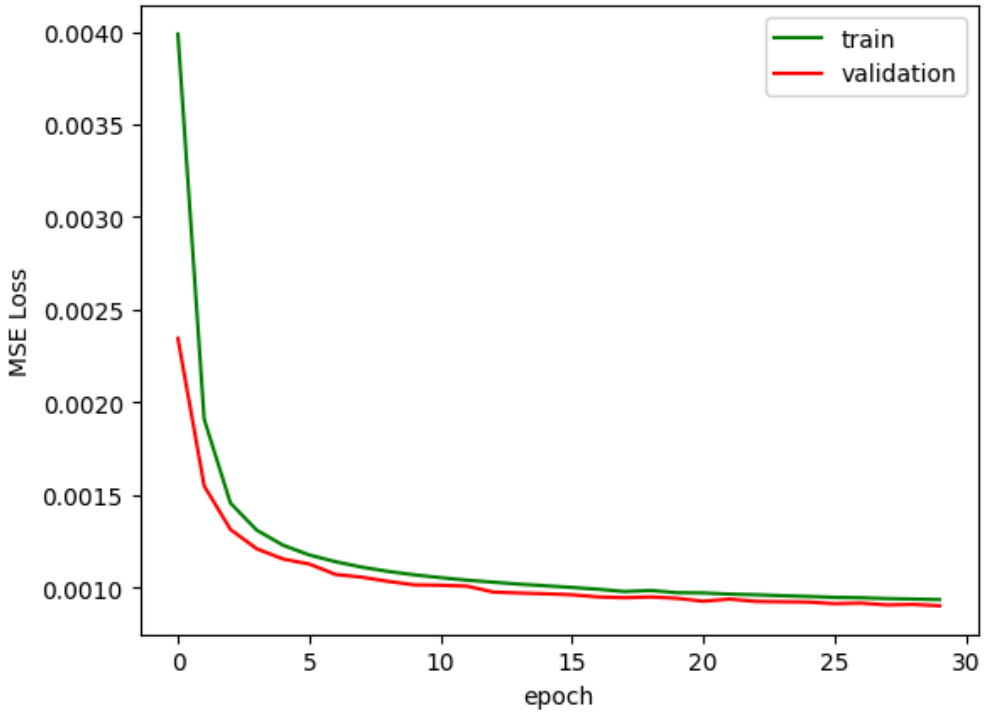

Рис. 20: График обучения модели PGM_128. Уменьшение функции потерь на обучающей выборке (зеленая кривая) и валидационной (красная).

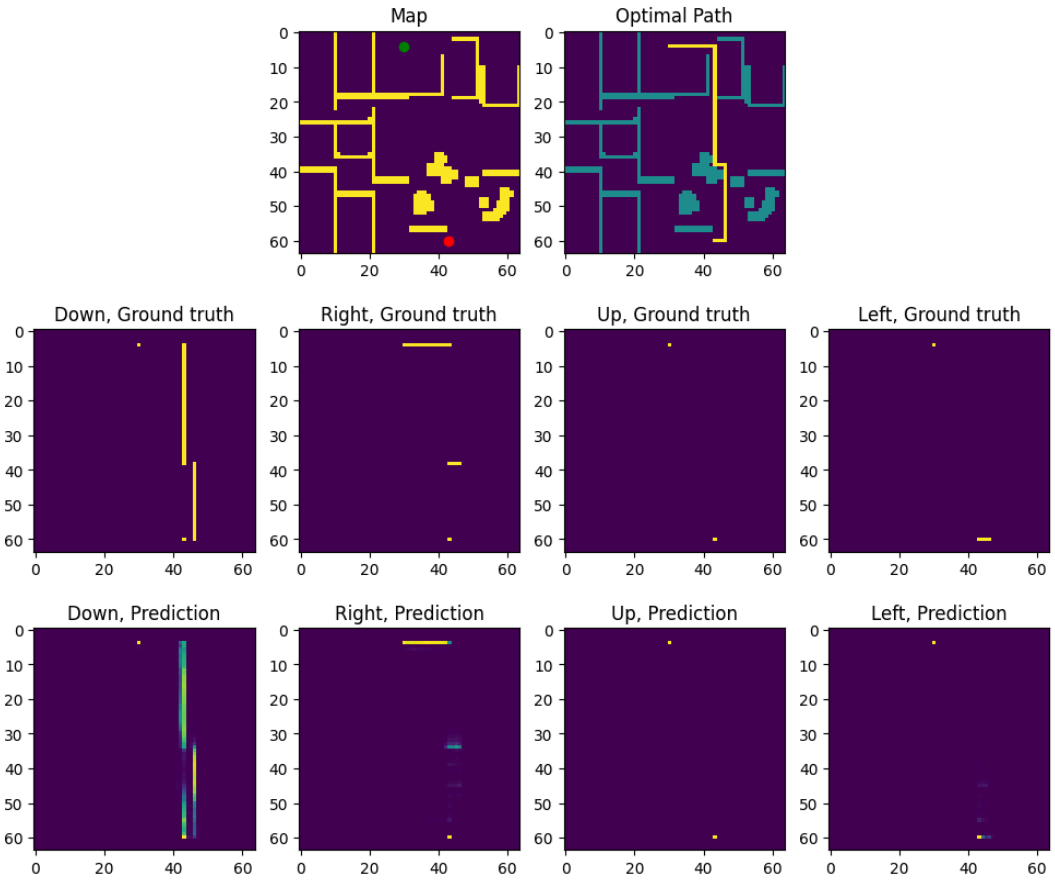

Рис. 21: Пример предсказаний PGM_64 на конкретной карте из валидации.

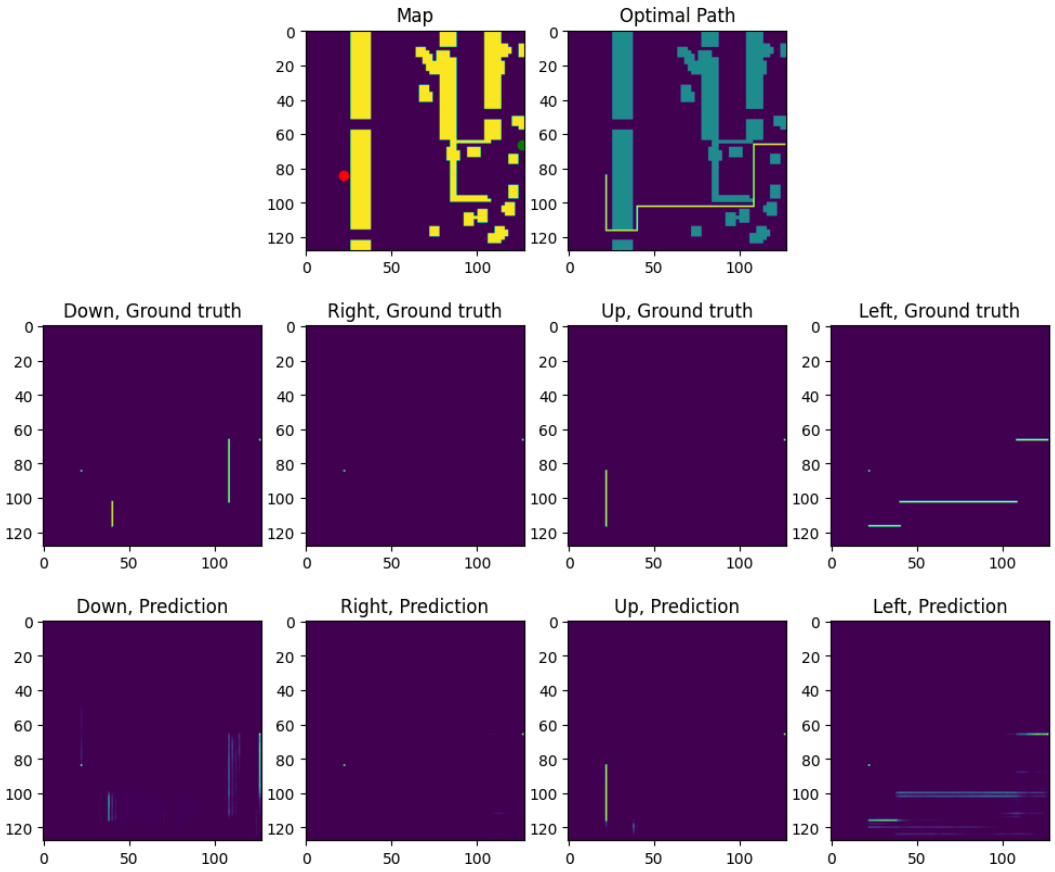

Рис. 22: Пример предсказаний PGM_128 на конкретной карте из валидации.

5. Эксперименты

Анализ результатов будет проводиться на валидационном датасете. Будем рассматривать следующие алгоритмы: TurnA*, WTurnA*, FocalTurnA*, GBFS. Алгоритмы WTurnA* и FocalTurnA* были протестированы со следующими весами: [1.01, 1.1, 1.5, 2, 3, 4, 5, 10]. А в качестве второстепенной эвристики для алгоритмов FocalTurnA* и GBFS были протестированы как предсказанные моделью PGM, так и их gt значения. Сделано это для того, чтобы отследить, насколько отличается работа планировщиков с предсказаниями и истинными значения. Во время тестирования отслеживались следующие метрики:

1) Стоимость найденного пути от старта до цели;
2) Число шагов алгоритма, то есть итераций основного цикла;
3) Число созданных вершин, а именно суммарный размер списка $OPEN$ и $CLOSED$ к концу алгоритма;

В качестве эталона выбран алгоритм TurnA*, поэтому все метрики будут выражаться в процентах относительно эталона. Также, подсчитаем процент найденных оптимальных решений на валидационной выборке для каждого алгоритма.

Во время экспериментов, все задачи были разделены на *легкие, средние и сложные* по следующему принципу: с помощью TurnA* находился оптимальный путь π* и считалось число шагов алгоритма $steps$. Затем считалось отношение $steps\ /\ cost(\pi^*)$. Если это отношение $\leq$ 10, то задача считалась легкой, если > 20, то сложной, иначе средней. Это объясняется тем, что в идеальном случае планировщик каждую итерацию алгоритма будет раскрывать вершину, лежащую на оптимальном пути, тем самым отношение $steps\ /\ cost(\pi^*)$ будет равно 1. Чем это отношение больше, тем труднее для планировщика было найти путь.

По результатам, в картах 64x64 отношение легких, средних и сложных задач составило 5:25:70, а в картах 128x128 это отношение равно 1:7:92.

Сначала были проведены эксперименты на картах размера 64x64:

Легкие задачи:

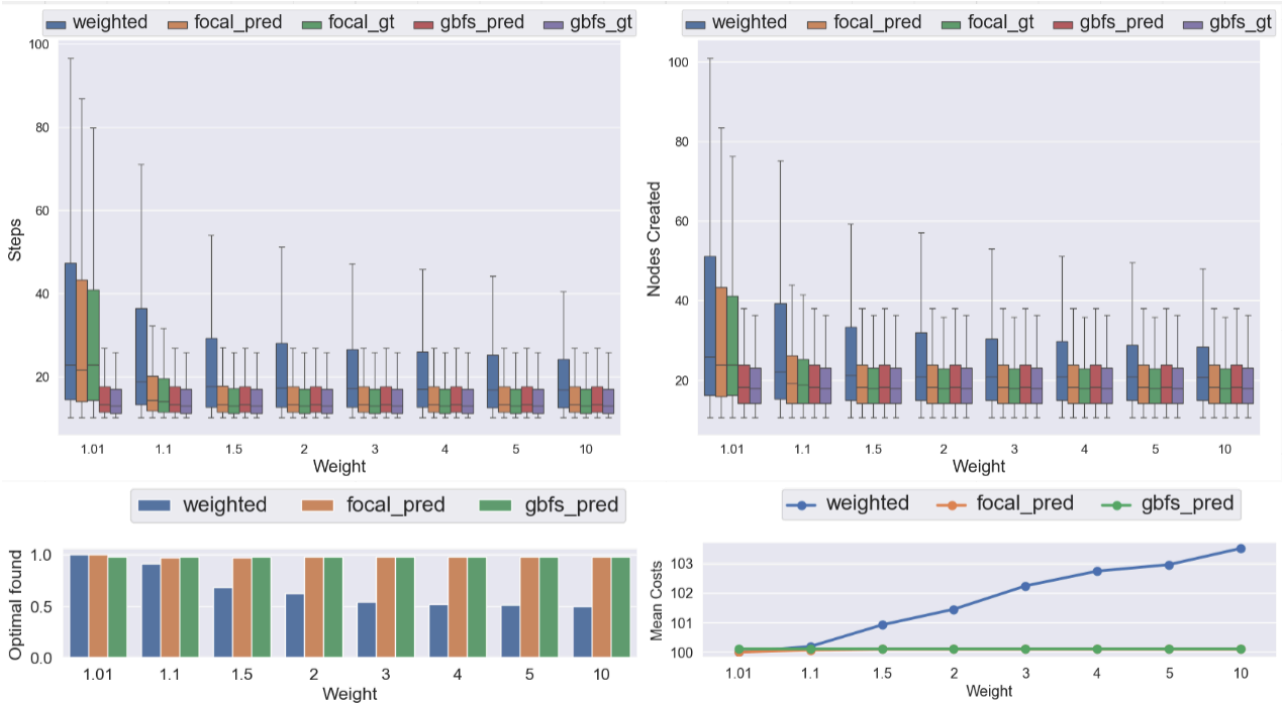

Рис. 23: Карты 64x64. Легкие задачи. Первая строка – диаграммы размаха для шагов алгоритма и число созданных вершин. Вторая строка – процент найденных оптимальных решений и средняя стоимость найденных путей.

Алгоритм	Шаги, %	Вершины, %	Стоимости, %	Оптимальность, %
WTurnA*, $w = 2$	24.48 ± 19.39	25.48 ± 14.23	101.45 ± 3.37	61.74
FocalTurnA*, предсказания, $w = 2$	15.82 ± 7.64	20.58 ± 9.31	100.10 ± 1.00	97.56
GBFS, предсказания	15.84 ± 7.67	20.61 ± 9.36	100.10 ± 1.00	97.56
FocalTurnA*, gt, $w = 2$	15.42 ± 7.36	20.01 ± 8.70	100	100
GBFS, gt	15.42 ± 7.36	20.02 ± 8.70	100	100

Таблица 1: Карты 64x64. Легкие задачи. Средние значения метрик ± разброс.

Средние задачи:

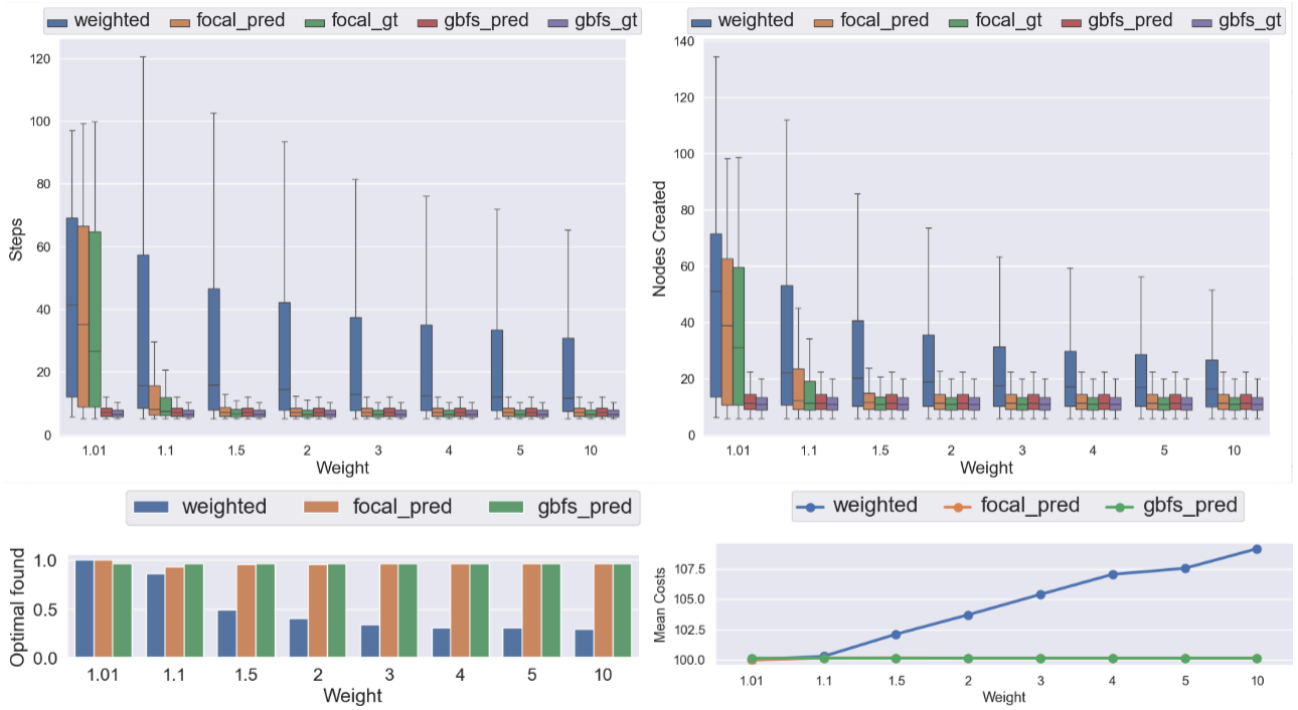

Рис. 24: Карты 64x64. Средние задачи. Первая строка – диаграммы размаха для шагов алгоритма и число созданных вершин. Вторая строка – процент найденных оптимальных решений и средняя стоимость найденных путей.

Алгоритм	Шаги, %	Вершины, %	Стоимости, %	Оптимальность, %
WTurnA*, $w = 2$	26.93 ± 26.45	24.91 ± 18.41	103.71 ± 6.05	40.02
FocalTurnA*, предсказания, $w = 2$	7.93 ± 5.49	12.65 ± 6.61	100.13 ± 0.74	95.41
GBFS, предсказания	7.43 ± 2.12	12.22 ± 4.62	100.13 ± 0.72	95.57
FocalTurnA*, gt, $w = 2$	7.02 ± 2.68	11.36 ± 3.80	100	100
GBFS, gt	6.86 ± 1.36	11.23 ± 3.20	100	100

Таблица 2: Карты 64x64. Средние задачи. Средние значения метрик ± разброс.

Сложные задачи:

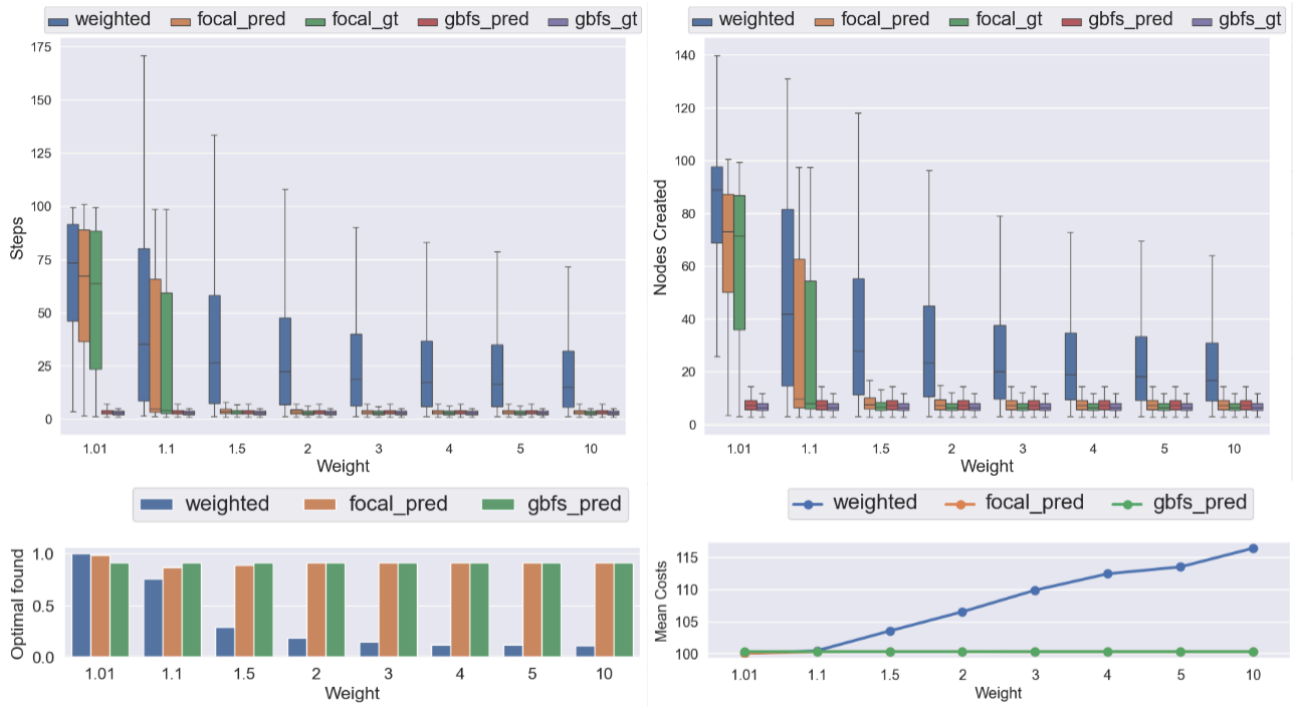

Рис. 25: Карты 64x64. Сложные задачи. Первая строка – диаграммы размаха для шагов алгоритма и число созданных вершин. Вторая строка – процент найденных оптимальных решений и средняя стоимость найденных путей.

Алгоритм	Шаги, %	Вершины, %	Стоимости, %	Оптимальность, %
WTurnA*, $w = 2$	29.90 ± 25.86	29.76 ± 22.38	106.50 ± 7.33	19.43
FocalTurnA*, предсказания, $w = 2$	5.12 ± 9.1	9.33 ± 9.75	100.27 ± 1.46	90.84
GBFS, предсказания	3.60 ± 2.39	7.82 ± 4.09	100.26 ± 1.49	91.48
FocalTurnA*, gt, $w = 2$	3.61 ± 3.89	7.18 ± 4.09	100	100
GBFS, gt	3.06 ± 0.99	6.69 ± 1.97	100	100

Таблица 3: Карты 64x64. Сложные задачи. Средние значения метрик ± разброс.

Во время экспериментов, какие-то задания решались заметно хуже остальных, что приводило к наличию в эмпирическом распределении хвостов, сильно увеличивающих разбросы отслеживаемых метрик. В связи с этим, для сложных задач были рассчитаны доверительные интервалы для процента шагов, созданных вершин и стоимостей с уровнем доверия 95%:

Алгоритм	Шаги, %	Вершины, %	Стоимости, %
WTurnA*, $w = 2$	(29.43, 30.38)	(29.35, 30.18)	(106.37, 106.64)
FocalTurnA*, предсказания, $w = 2$	(4.96, 5.3)	(9.16, 9.52)	(100.25, 100.30)
GBFS, предсказания	(3.54, 3.68)	(7.11, 7.25)	(100.23, 100.28)
FocalTurnA*, gt, $w = 2$	(3.57, 3.66)	(7.18, 7.33)	100
GBFS, gt	(3.05, 3.08)	(6.65, 6.73)	100

Таблица 4: Карты 64x64. Сложные задачи. Доверительные интервалы для метрик.

Теперь представим результаты экспериментов на картах размера 128x128:

Легкие задачи:

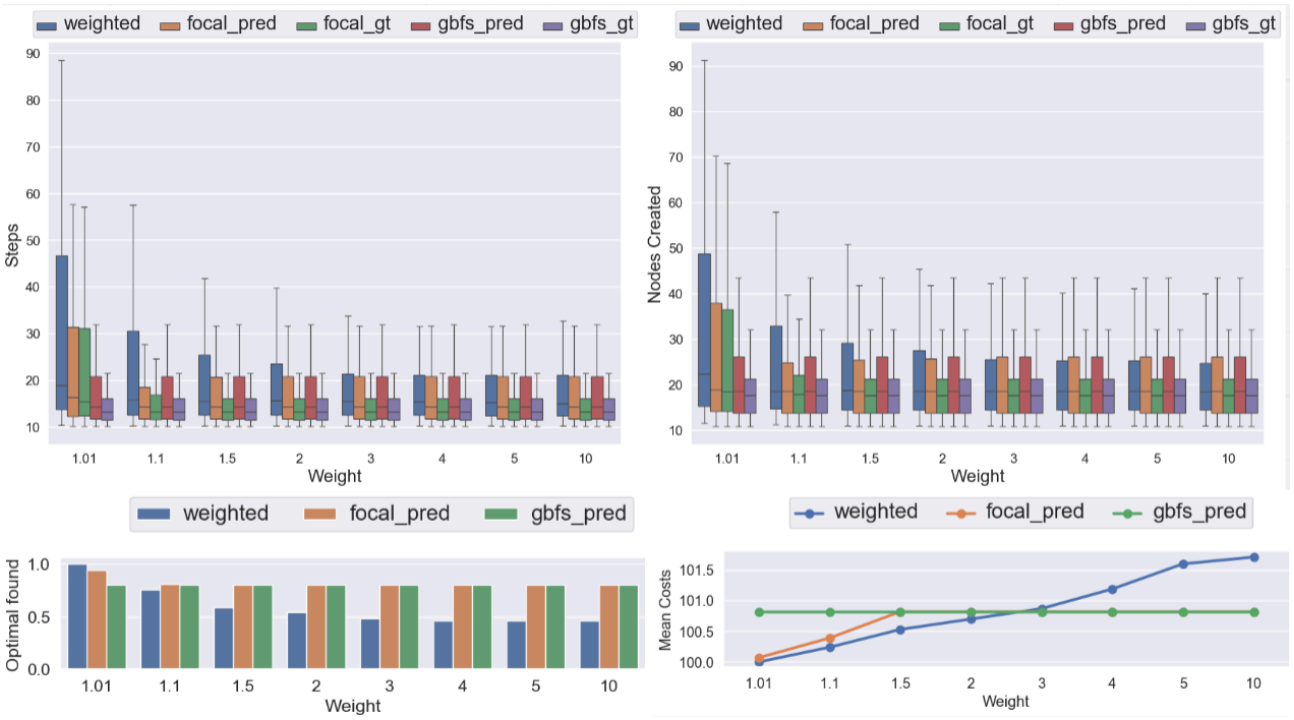

Рис. 26: Карты 128x128. Легкие задачи. Первая строка – диаграммы размаха для шагов алгоритма и число созданных вершин. Вторая строка – процент найденных оптимальных решений и средняя стоимость найденных путей.

Алгоритм	Шаги, %	Вершины, %	Стоимости, %	Оптимальность, %
WTurnA*, $w = 2$	22.90 ± 21.53	23.89 ± 15.80	100.70 ± 1.51	53.66
FocalTurnA*, предсказания, $w = 2$	18.97 ± 13.89	24.35 ± 17.89	100.82 ± 2.45	79.67
GBFS, предсказания	19.13 ± 14.48	24.55 ± 18.41	100.82 ± 2.45	79.67
FocalTurnA*, gt, $w = 2$	15.55 ± 9.35	19.69 ± 10.36	100	100
GBFS, gt	15.55 ± 9.35	19.69 ± 10.36	100	100

Таблица 5: Карты 128x128. Легкие задачи. Средние значения метрик ± разброс.

Средние задачи:

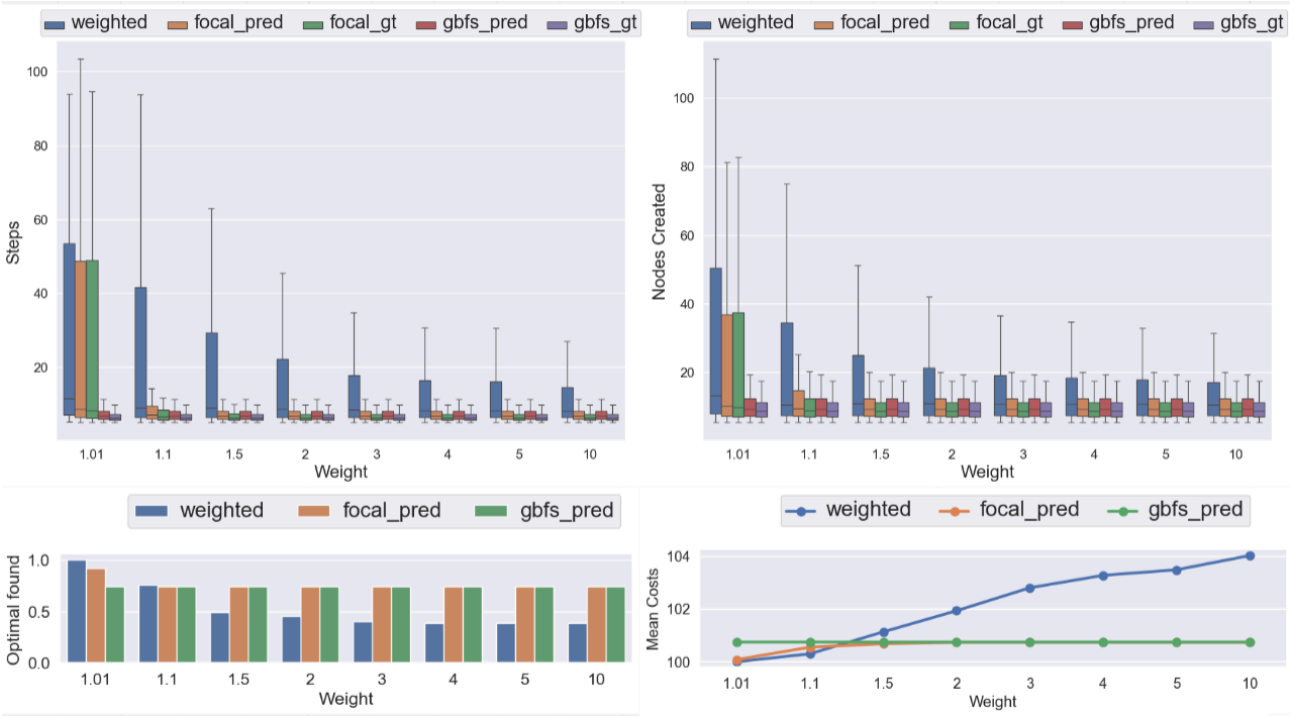

Рис. 27: Карты 128x128. Средние задачи. Первая строка – диаграммы размаха для шагов алгоритма и число созданных вершин. Вторая строка – процент найденных оптимальных решений и средняя стоимость найденных путей.

Алгоритм	Шаги, %	Вершины, %	Стоимости, %	Оптимальность, %
WTurnA*, $w = 2$	19.29 ± 23.55	16.78 ± 13.87	101.93 ± 3.97	45.05
FocalTurnA*, предсказания, $w = 2$	7.60 ± 3.42	10.95 ± 6.57	100.73 ± 2.27	74.23
GBFS, предсказания	7.65 ± 3.57	11.06 ± 6.99	100.73 ± 2.27	74.23
FocalTurnA*, gt, $w = 2$	6.67 ± 1.28	9.49 ± 3.06	100	100
GBFS, gt	6.67 ± 1.28	9.49 ± 3.06	100	100

Таблица 6: Карты 128x128. Средние задачи. Средние значения метрик ± разброс.

Сложные задачи:

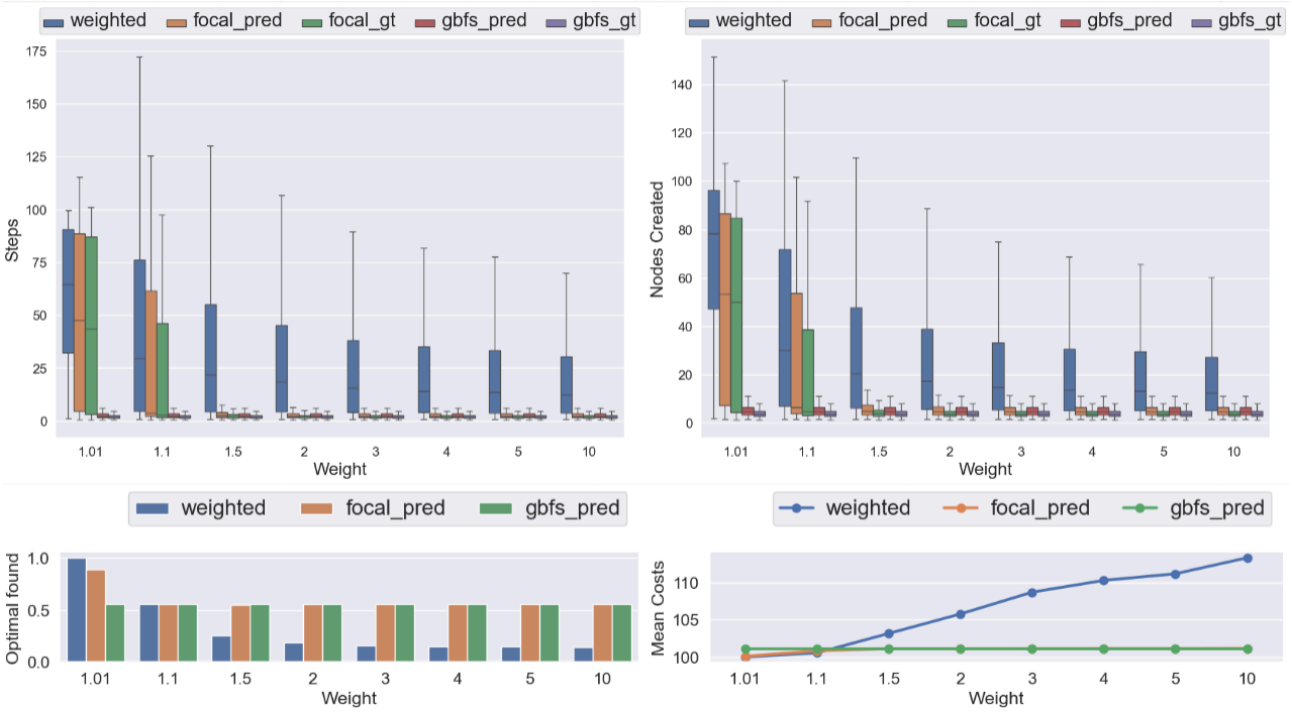

Рис. 28: Карты 128x128. Сложные задачи. Первая строка – диаграммы размаха для шагов алгоритма и число созданных вершин. Вторая строка – процент найденных оптимальных решений и средняя стоимость найденных путей.

Алгоритм	Шаги, %	Вершины, %	Стоимости, %	Оптимальность, %
WTurnA*, $w = 2$	27.71 ± 27.33	25.21 ± 22.70	105.76 ± 7.18	19.23
FocalTurnA*, предсказания, $w = 2$	4.52 ± 10.27	7.38 ± 10.66	101.12 ± 2.63	56.00
GBFS, предсказания	2.95 ± 3.42	5.99 ± 5.87	101.15 ± 2.78	56.09
FocalTurnA*, gt, $w = 2$	2.47 ± 3.47	4.54 ± 3.56	100	100
GBFS, gt	2.05 ± 1.00	4.16 ± 1.72	100	100

Таблица 7: Карты 128x128. Сложные задачи. Средние значения метрик ± разброс.

Также рассчитаны доверительные интервалы для сложных задач 128x128:

Алгоритм	Шаги, %	Вершины, %	Стоимости, %
WTurnA*, $w = 2$	(27.11, 28.31)	(24.71, 25.71)	(105.61, 105.92)
FocalTurnA*, предсказания, $w = 2$	(4.31, 4.75)	(7.16, 7.62)	(101.07, 101.18)
GBFS, предсказания	(2.4, 2.55)	(4.47, 4.62)	(100.91, 101.13)
FocalTurnA*, gt, $w = 2$	(2.43, 2.58)	(4.60, 4.74)	100
GBFS, gt	(2.03, 2.07)	(4.12, 4.19)	100

Таблица 8: Карты 128x128. Сложные задачи. Доверительные интервалы для метрик.

Помимо основных мерик, было подсчитано среднее время, за которое алгоритмы находят решения для всех задач из батча из 128 сэмплов. При этом здесь разделение задач на сложности не учитывалось.

Карты 64x64:

Алгоритм	Среднее время, секунды
TurnA*	8.03
WTurnA*, w=2	2.41
FocalTurnA*, предсказания, w=2	3.06
GBFS, предсказания	2.13
FocalTurnA*, gt, w=2	2.38
GBFS, gt	2.07

Таблица 9: Карты 64x64. Среднее время решения батча из 128 задач.

Заметим, что в алгоритмах FocalTurnA* и GBFS также учитывалось время, потраченное моделью на предсказание PGM для всего батча, что в среднем составило 1.8 секунд.

Карты 128x128:

Алгоритм	Среднее время, секунды
TurnA*	32.36
WTurnA*, w=2	9.06
FocalTurnA*, предсказания, w=2	13.95
GBFS, предсказания	3.85
FocalTurnA*, gt, w=2	4.52
GBFS, gt	3.43

Таблица 10: Карты 128x128. Среднее время решения батча из 128 задач.

Среднее время, потраченное моделью на предсказание PGM для всего батча составило 2.8 секунды.

Дополнительно, для модели PGM64 были проведены эксперименты на трех картах, которые по своему строению сильно отличаются от карт, представленных в основном датасете: городская местность, лабиринт и компьютерная игра:

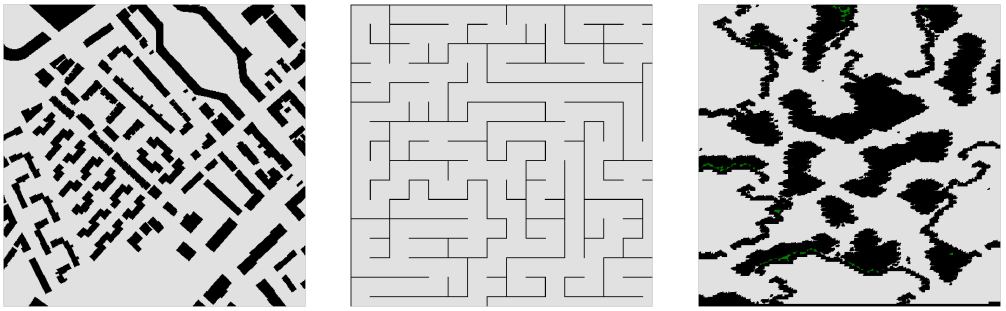

Рис. 29: Слева направо: Boston_0_512, maze512-32-1, EbonLakes.

Карты *Boston_0_512*, *maze512-32-1* и *EbonLakes* были взяты из работы MovinAI Benchmarks [30] и приведены к размеру 64x64. Для каждой карты было сгенерировано 1024 задачи, и для каждой из них подсчитаны истинные значения PGM. Ниже представлены результаты для каждой карты без разделения на сложности:

Алгоритм	Шаги, %	Вершины, %	Стоимости, %	Оптимальность, %
WTurnA*, $w = 2$	31.01 ± 24.6	33.58 ± 22.89	107.31 ± 5.01	4.9
FocalTurnA*, предсказания, $w = 2$	23.26 ± 30.03	32.08 ± 28.51	113.64 ± 12.71	1.5
GBFS, предсказания	24.68 ± 38.06	36.39 ± 37.75	115.18 ± 17.38	1.5
FocalTurnA*, gt, $w = 2$	2.78 ± 4.14	6.13 ± 4.15	100	100
GBFS, gt	2.18 ± 0.76	5.58 ± 1.63	100	100

Таблица 11: Карта Boston_0_512. Средние значения метрик ± разброс.

Алгоритм	Среднее время, секунды
TurnA*	13.67
WTurnA*, $w = 2$	4.51
FocalTurnA*, предсказания, $w = 2$	17.55
GBFS, предсказания	2.53
FocalTurnA*, gt, $w = 2$	6.26
GBFS, gt	2.16

Таблица 12: Карта Boston_0_512. Среднее время решения батча из 128 задач.

Алгоритм	Шаги, %	Вершины, %	Стоимости, %	Оптимальность, %
WTurnA*, $w = 2$	85.59 ± 9.0	85.92 ± 8.6	100.77 ± 0.57	20.1
FocalTurnA*, предсказания, $w = 2$	77.98 ± 13.82	80.22 ± 13.37	113.43 ± 5.77	0.0
GBFS, предсказания	71.82 ± 18.28	77.89 ± 18.18	114.21 ± 6.05	0.2
FocalTurnA*, gt, $w = 2$	39.33 ± 23.87	42.26 ± 22.39	100	100
GBFS, gt	3.36 ± 0.79	9.35 ± 2.13	100	100

Таблица 13: Карта maze512-32-1. Средние значения метрик ± разброс.

Алгоритм	Среднее время, секунды
TurnA*	20.04
WTurnA*, $w = 2$	17.39
FocalTurnA*, предсказания, $w = 2$	41.47
GBFS, предсказания	21.51
FocalTurnA*, gt, $w = 2$	19.89
GBFS, gt	2.81

Таблица 14: Карта maze512-32-1. Среднее время решения батча из 128 задач.

Алгоритм	Шаги, %	Вершины, %	Стоимости, %	Оптимальность, %
WTurnA*, $w = 2$	32.96 ± 19.13	36.12 ± 17.44	108.31 ± 6.91	10.4
FocalTurnA*, предсказания, $w = 2$	10.09 ± 11.39	18.76 ± 12.9	104.36 ± 5.41	18.4
GBFS, предсказания	10.07 ± 12.48	19.85 ± 16.02	104.59 ± 6.29	18.4
FocalTurnA*, gt, $w = 2$	2.85 ± 1.52	7.23 ± 2.35	100	100
GBFS, gt	2.79 ± 0.92	7.18 ± 2.13	100	100

Таблица 15: Карта EbonLakes. Средние значения метрик ± разброс.

Алгоритм	Среднее время, секунды
TurnA*	10.58
WTurnA*, $w = 2$	3.63
FocalTurnA*, предсказания, $w = 2$	6.06
GBFS, предсказания	2.46
FocalTurnA*, gt, $w = 2$	3.34
GBFS, gt	2.14

Таблица 16: Карта EbonLakes. Среднее время решения батча из 128 задач.

6. Выводы по результатам экспериментального исследования

Данные выводы сделаны как для карт размера 64x64, так и 128x128.

При увеличении веса в алгоритме WTurnA* падает число шагов алгоритма и число созданных вершин, однако сильно увеличивается средняя стоимость найденных путей, что приводит к более редкому нахождению оптимальных решений. При этом, для каждого веса алгоритм WTurnA* оказывается хуже, чем FocalTurnA* с тем же весом.

В алгоритме FocalTurnA* как для предсказаний, так и для истинных значений PGM, хуже всего себя показывают маленькие веса 1.01 и 1.1, при которых алгоритм тратит больше шагов и создает больше вершин, по сравнению с большими весами. При этом стоит отметить, что, начиная с веса 1.5, все показатели у FocalTurnA* отличаются крайне незначительно.

Алгоритм GBFS, который не зависит от веса, показывает результаты, близкие или немного лучше, чем FocalTurnA* на легких и средних задачах, и значительно улучшает число шагов и созданных вершин на сложных задач, выигрывая как по средним показаниям, так и по меньшему разбросу. Также отметим, что в среднем, GBFS решает батч из 128 задач быстрее, чем WTurnA* и FocalTurnA* .

При увеличении сложности задач: проценты шагов и созданных вершин у FocalTurnA* и GBFS становятся только меньше; процент найденных оптимальных путей у FocalTurnA* и GBFS также падает, больше всего опускаясь на сложных задачах 128x128 до 56%, но при этом стоимости найденных решений остаются близкими к оптимальным; алгоритмы FocalTurnA* и GBFS лучше WTurnA* по всем отслеживаемым метрикам во всех случаях, кроме легких задач 128x128, где WTurnA* в среднем создает чуть меньше вершин и находит пути, в среднем чуть меньшие по стоимости.

Учитывая показания отслеживаемых метрик для всех случаев и среднее время работы, алгоритм GBFS показал лучшие результаты.

По экспериментам на картах, сильно отличающихся от карт датасета, можно сделать следующие выводы о модели PGM_64:

На карты с очень сложными топологиями модель обобщается плохо, что заметно по карте *maze* со строением лабиринта, и что вполне ожидаемо. Хорошие результаты модель показала на карте *Boston*, немного уступая алгоритму WTurnA* в стоимости найденных решений, и сильные результаты на карте *EbonLakes*. Это может говорить нам о возможностях и перспективах модели PGM_64 к обобщению и хорошим показателям на картах, отличающихся от датасета, на котором она обучалась.

Заключение

В данной работе была исследована задача планирования траектории мобильного агента, и изучены современные подходы к ее решению. Формально поставлена задача о построении оптимального пути мобильного агента с поворотами, и созданы алгоритмы TurnA* и FocalTurnA* – модификации алгоритмов A* и Focal Search соответственно.

Был создан датасет, включающий в себя карты с разными топологиями местности, пары стартов и целей, а также истинные значения Path Guidance Maps – второстепенных эвристик для алгоритма FocalTurnA*.

Были созданы и обучены модели PGM_64 и PGM_128, с использованием архитектуры трансформера, по предсказанию PGM для карт размеров 64x64 и 128x128.

По результатам экспериментов были сделаны следующие выводы:

Алгоритмы FocalTurnA* и GBFS, использующие предсказания моделей в качестве второстепенной эвристики, показали существенные улучшения по числу шагов, числу раскрытых вершин и времени поиска кратчайшего пути, однако в картах 128x128 процент найденных оптимальных решений не слишком высок. Для более эффективной работы алгоритма FocalTurnA* следует выбирать вес, не меньший 1,5. На основании отслеживаемых метрик и времени решения батча из 128 задач, алгоритм GBFS в среднем показывает себя лучше, чем FocalTurnA*. Планировщики, использующие предсказания моделей, добились метрик, относительно близких к своим идеальным версиям, использующим истинные значения, но качество этих предсказаний все еще можно улучшить.

Github: https://github.com/MarkPermyak/TurnAstar

Список литературы

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

[2] Hart, P., Nilsson, N., & Raphael, B. (1968). “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107

[3] Pohl, I. 1970. “Heuristic search viewed as path finding in a graph”. Artificial intelligence, 1(3-4): 193–204

[4] Pearl, J.; and Kim, J. H. 1982. “Studies in semi-admissible heuristics”. IEEE transactions on pattern analysis and machine intelligence, (4): 392–399.

[5] Pohl, I. 1969. “Bidirectional Heuristic Search in Path Problems”. Ph.D. Dissertation, Stanford University

[6] Richard E. Korf. “Depth-first iterative-deepening: An optimal admissible tree search”, Artificial Intelligence, Volume 27, Issue 1, 1985,

[7] Daniel, K. and Nash, A. and Koenig, S. and Felner, A. 2010. “Theta*: Any-Angle Path Planning on Grids”. Journal of Artificial Intelligence Research, Volume 39.

[8] D. Harabor and A. Grastien. 2011. “Online Graph Pruning for Pathfinding on Grid Maps”. In Proceedings of the 25th National Conference on Artificial Intelligence (AAAI), San Francisco, USA.

[9] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” TRA, 12(4): 566–580, 1996

[10] S. M. LaValle and J. J. Kuffner Jr.. “Randomized kinodynamic planning,” IJRR, 20(5): 378–400, 2001.

[11] S. Karaman and E. Frazzoli. “Sampling-based algorithms for optimal motion planning,” IJRR, 30(7): 846–894, 2011.

[12] Gammell, Jonathan D. and Srinivasa, Siddhartha S. and Barfoot, Timothy D.. “Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs”, 2015 IEEE International Conference on Robotics and Automation (ICRA 2015), pp. 3067-3074.

[13] Takahashi, T.; Sun, H.; Tian, D.; and Wang, Y. 2019. “Learning heuristic functions for mobile robot path planning using deep neural networks”. In Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS 2019), 764–772.

[14] Binghong Chen, Chengtao Li, Hanjun Dai, Le Song 2020. “Retro*: Learning Retrosynthetic Planning with Neural Guided A* Search”, presented at ICML 2020.

[15] Yonetani, R.; Taniai, T.; Barekatain, M.; Nishimura, M.; and Kanezaki, A. 2021. “Path planning using neural A* search”. In Proceedings of the 38th International Conference on Machine Learning (ICML 2021), 12029–12039.

[16] Li, Z.; Chen, Q.; and Koltun, V. 2018. “Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search”.

[17] Bhardwaj, M.; Choudhury, S.; and Scherer, S. 2017. “Learning heuristic search via imitation”. In Proceedings of the 1st Conference on Robot Learning (CoRL 2017), 271–280.

[18] Sanjiban Choudhury, Mohak Bhardwaj, Sankalp Arora, Ashish Kapoor, Gireeja Ranade, Sebastian Scherer, Debadeepta Dey, 2017. “Data-driven Planning via Imitation Learning”.

[19] Edward Groshev, Maxwell Goldstein, Aviv Tamar, Siddharth Srivastava, Pieter Abbeel, 2017. “Learning Generalized Reactive Policies using Deep Neural Networks”.

[20] Leah Chrestien, Tomas Pevny, Antonin Komenda, Stefan Edelkamp, 2021. “Heuristic Search Planning with Deep Neural Networks using Imitation, Attention and Curriculum Learning”.

[21] Michal Pándy, Weikang Qiu, Gabriele Corso, Petar Veličković, Rex Ying, Jure Leskovec, Pietro Liò, 2022. “Learning Graph Search Heuristics”.

[22] Daniil Kirilenko, Anton Andreychuk, Aleksandr Panov, Konstantin Yakovlev. “TransPath: Learning Heuristics For Grid-Based Pathfinding via Transformers”. Proceedings of the AAAI Conference on Artificial Intelligence. 37, 10 (Jun. 2023), 12436-12443

[23] Fukushima, Kunihiko (1980). "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". *Biological Cybernetics*. 36 (4): 193–202.

[24] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. “Deep Residual Learning for Image Recognition”. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[25] Yuxin Wu and Kaiming He, “Group Normalization”, 2018.

[26] Prajit Ramachandran and Barret Zoph and Quoc V. Le, “Searching for Activation Functions”, 2017.

[27] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. “Attention is all you need”. Advances in neural information processing systems, 30.

[28] Jimmy Lei Ba and Jamie Ryan Kiros and Geoffrey E. Hinton, “Layer Normalization”, 2016.

[29] Diederik P. Kingma, Jimmy Ba. “Adam: A Method for Stochastic Optimization”, 2014

[30] Sturtevant, N. R. 2012. Benchmarks for Grid-Based Pathfinding. IEEE Transactions on Computational Intelligence and AI in Games, 4(2): 144–148.