ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
"САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ"
(СПбГУ)

Образовательная программа магистратуры "Современная математика"

Отчёт по курсу «Методы и алгоритмы эвристического поиска»: проект «Декомпозиция при поиске суб-оптимальных решений (R*, MRA*)»

Команда **BetterThan**:
Искандер Азангулов,
Евгений Вагин

Руководитель:
Яковлев Константин Сергеевич

Санкт-Петербург
2021 год

Содержание

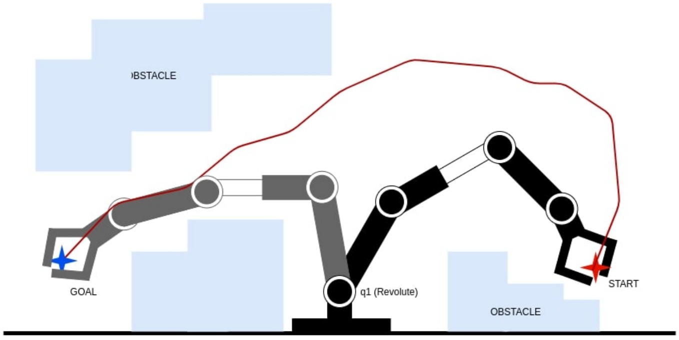


Рис. 1: Пример задачи поиска пути. Требуется переместить манипулятор из *start* в *goal*

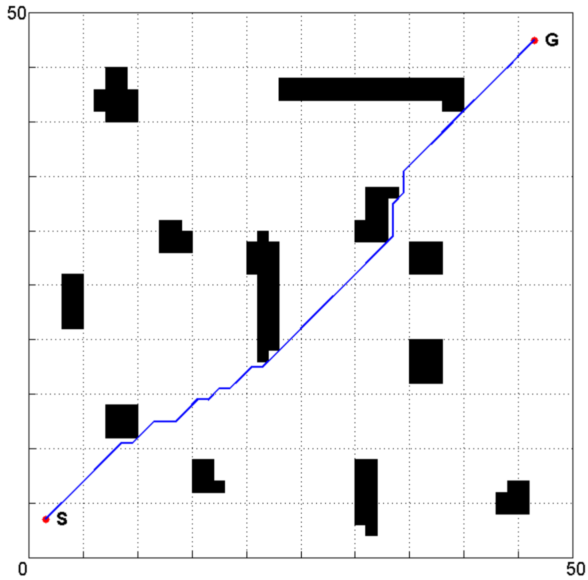


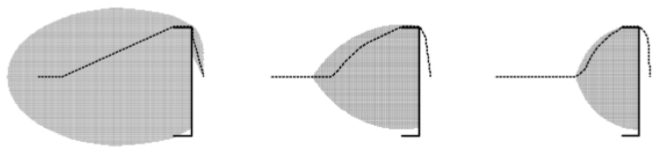
Рис. 2: Пример оптимального пути на клетчатой решетке.

Задача планирования траекторий

Задача планирование траекторий — это вычислительная задача поиска последовательности допустимых конфигураций, которая перемещает объект от источника к месту назначения. Как подзадача данная задача встречается при программированиии роборотов, в компьютерных играх, в вычислительной геометрии.

Задача планирования при помощи дискретизации пространства конфигураций, сводится к задаче поиска оптимального пути во взвещенном графе $G = (V, E)$. Множество вершин V соответствует всевозможным конфигурациям, а множество ребер E трудозатратам при переходе между конфигурациями. Соответственно задача о поиске конфигурации сводится к задаче о поиске пути во взвешенном графе из изначальной позиции v_{start} к искомой v_{goal}. Обычно построенный граф является не случайным и мы априори знаем оценку $h(u, v)$ на расстояния между вершинами. Данную оценка называется эвристической функцией. Соответственно задача сводится к нахождению кратчайшего пути $path_{opt} = (v_0 = v_{start}, v_1, \ldots, v_n = v_{goal})$, где $len(path_{opt}) = \min_{path} len(path)$

Мы ограничимся случаем, когда построенный граф является подграфом регулярной решеткой в $\mathbb{Z}^d$. В этом случае в качестве эвристической функции обычно используется какая-нибудь стандартная метрика, например евклидова.



(a) A*   (b) wA*, $w = 2$ (c) wA*, $w = 5$

Рис. 3: WA* алгоритм. Зависимость количества раскрытых клеток от w

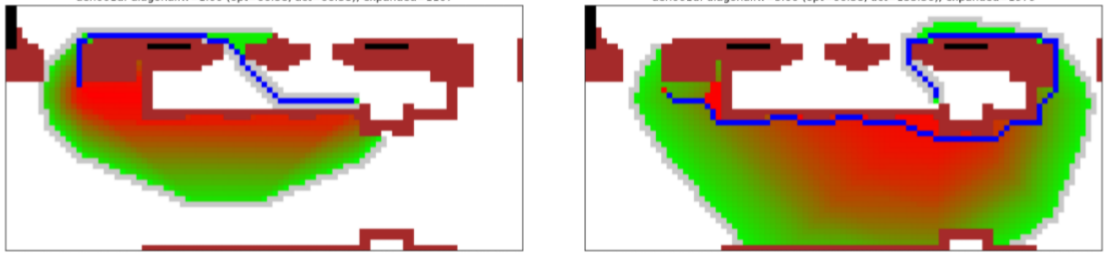


Рис. 4: WA* алгоритм. Увеличение w привело к дольшей работе алгоритма.

1 Суб-оптимальные алгоритмы.

Отметим, что не во всех случаях необходимо находить именно кратчайший путь между вершинами v_{start} и v_{goal}. Во многих случаях достаточно найти путь который не более чем в w раз хуже чем оптимальный. т.е. $len(path) \leq w(path_{opt})$. Так как это накладывает куда меньшие ограничения на путь, соответственно, существует надежда найти его куда быстрее. Алгоритмы находящие такие пути будем называть суб-оптимальными.

WA*

Классическим примером суб-оптимального алгоритма является WA*. Его описание может быть найдено на Википедии.

Отметим, несколько проблем данного алгоритма:

- Количество иследуемых вершин хоть и уменьшается все равно остается квадратичным от размера поля.(см. 3)

- Большая зависимость от эвристической функции. (см. 4)

Декомпозиция

Для преодоления данных проблем можно использовать методы основанные на декомпозиции. Идея данных методов заключается в построении вспомогательного графа $G' = (V', E')$. Вершины V' – это подмножество вершин изначального графа, а ребра E' это некоторые пути между вершинами в изначальном графе. Путь между v_{goal} и v_{start} ищется уже в новом графе. Очевидно, что по найденнному пути легко можно востановить путь в исходном графе, если мы знаем соответствие между ребрами в E' и путями в G.

Данный подход исправляет недостатки перечисленные выше. Во-первых, так как ребра это пути, мы за один раз перемещаемся сразу на несколько клеток. Во-вторых, так как мы сами выбираем пути в E', влияние эвристики уменьшается.

Мы сравним два алгоритма основанных на WA* и использующих декомпозицию. R* использующий декомпозицию построенную случайным образом и MRA* строящий де-

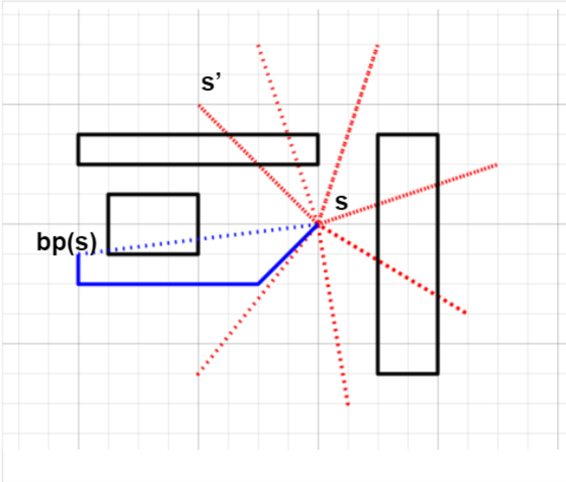


Рис. 5: R*, к потомкам вершины s ведут красные ребра, синий путь был найден при помощи WA* с ограничениями.

композицию регулярным образом.

Ниже используется стандартные обозначения используемые при описании A* алгоритма. Как и в прошлый раз мы отсылаем читателей к Википедии.

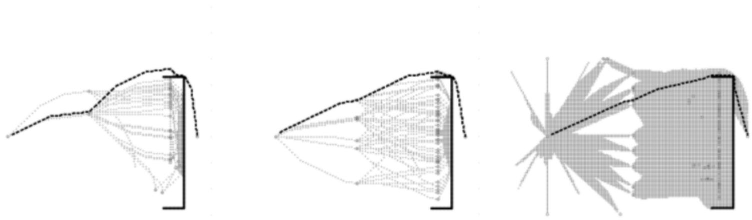
2 R*

R*[1], как говорилось выше, основан на WA*. Однако, упрощая, он вместо того чтобы раскрывать непосредственных соседей, соседями считает случайным образом выбранные вершины, на некотором отдалении.

Точнее алгоритм работает следующим образом:

- Во-первых, вместо добавления в $OPEN$ соседних вершин на очередном шаге, при раскрытии вершины s в $OPEN$ мы добавляем K случайных вершин на расстоянии Δ. Путь от s до s' оценивается как $c(s, s') = w \cdot h(s', s)$. И наконец, вершина s' добавляется в $OPEN$ с приоритетом $f(s) = g(s) + c(s, s')$, а её предком объявляется $bp(s') = s$.
- Во-вторых, непосредственно путь между s и $bp(s)$ вычисляется лениво. То есть, только при необходимости раскрыть вершину s. Сам путь между вычисляется при помощи WA* с ограничением на количество раскрытий. Основываясь на длине найденного пути или минимальной оценке его длины, если путь не найден мы обновляем $c(bp(s), s)$. Если путь найден, мы добавляем её обратно в $OPEN$ с приоритетом $f(s) = g(s) + c(bp(s), s)$. Если же путь не найден, среди вершин у которых есть наследник s, ищется другой предок s' минимизирующий $g(s') + c(s', s)$, s переподвешивается к этой вершине и добавляется в $OPEN$ с приоритетом соответствующим новому предку.
- В-третьих приоритетом вершины в $OPEN$ является пара $(AVOID, f(s))$. Где $AVOID$ принимает значения 0 (False) или 1(True). Это означает, что если вершина помечена $AVOID$, то мы её раскрываем в последнюю очередь. Вершина помечается $AVOID$ в двух случаях, если оказывается, что $f(s) > w \cdot h(s_{start}, s_{goal})$ или если путь от $bp(s)$ до s не удается найти.

Параметры алгоритма:



(a) R*, $w = 2$ (b) R*, $w = 1.5$ (c) R*, $w = 1$

Рис. 6: R*. Серым отмечены вершины которые были использованы в ходе работы алгоритма.

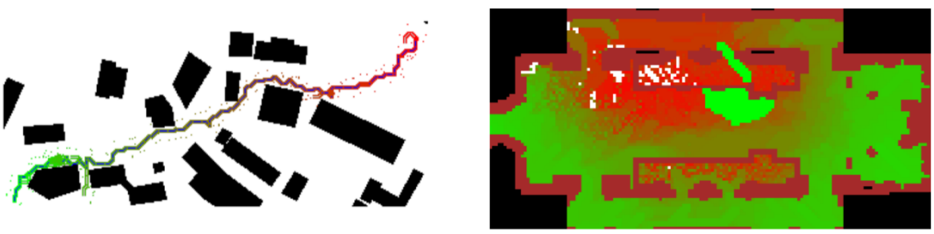


Рис. 7: При маленьком Δ(слева), алгоритм практически не имеет выигрыша. При большом Δ(справа) стены могут оказаться слишком тонкими и алгоритм раскрывает всю карту.

- K – количество соседей вершины s;
- Δ – расстояние между соседями;
- C – количество раскрытий во вспомогательном WA*;
- w – фактор субоптимальности;

Преимущества: кроме преимуществ присущих алгоритмам основанным на декомпозиции мы выделим также, меньший объем используемой памяти, так как нам не нужно хранить промежуточные шаги; доказуемая, почти наверное, субоптимальность, но уже с фактором w^2.

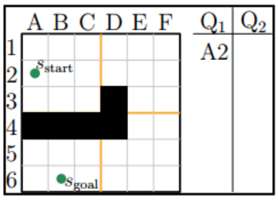
Недостатки: большая чувствительность к гиперпараметрам, особенно к параметру Δ; работает не на всех картах, если стены оказались слишком тонкими, то алгоритм может пометить v_{goal} как $AVOID$. В этом случае алгоритм исследует сначала всю карту, вместо того чтобы искать кратчайший путь; в наших экспериментах оказался худшим алгоритмом

3 MRA* (Multi-Resolution A*)

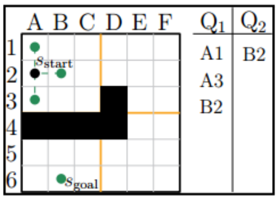
При поиске пути алгоритм зачастую на преодоление открытых пространств тратит столько же времени, сколько и на преодоление закрытых. В свете этого разумно как-то дискретизировать пространство поиска. При этом:

- на малом разрешении поиск более "манёвренный но менее производительный;
- на большом разрешении поиск быстрее исследует карту, но менее манёврен и стабилен.

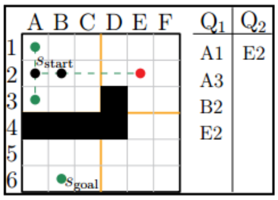
Алгоритм MRA* [2] совмещает поиски на разных разрешениях, тем самым позволяя избавиться от недостатков обоих. Для этого мы делаем несколько WA* поисков в разных разрешениях, поддерживая для каждого свои очереди $OPEN_i$ и $CLOSED_i$, а также ещё



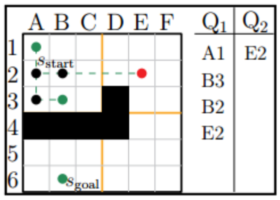
(a) MRA* is initialized. State s_{start} (A2) is inserted into Q_1.



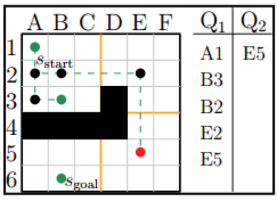
(b) State A2 is expanded by high resolution search. Since state B2 lies at the center of a coarse cell, it is also inserted into Q_2.



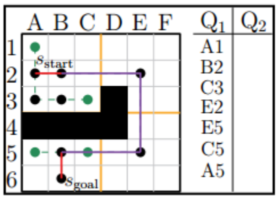
(c) State B2 is expanded by low resolution search. The successor E2 is inserted into both queues.



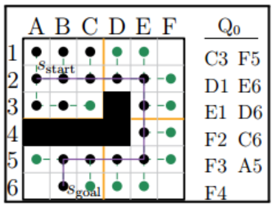
(d) State A3 is expanded by high resolution search.



(e) State E2 is expanded by low resolution search and the successor E5 is inserted into both queues.



(f) The last step when s_{goal} is expanded by high resolution search. A solution (solid line segments) is found with 8 expansions in total.



(g) The final status of a high resolution search. The same solution (purple) is found with 17 expansions.

Рис. 8: Пример работы алгоритма MRA*

один дополнительный (якорный) WA* для обмена состояний между $OPEN_i$. Например, если в качестве разрешений выбрать $\{1, 3\}$, то WA*(resolution=3) будет обмениваться состояниями с WA*(resolution=1) в каждой девятой клетке (центр квадрата 3×3). Размеры разрешений должны выбираться нечётными, иначе непонятно, какая клетка должна выбираться в качестве центра.

Таким образом, на каждой итерации алгоритм MRA*:

1. Выбирает очередь i, из которой будет выбрана вершина для раскрытия,
2. Делает раскрытие выбранной вершины,
3. Добавляет в другие очереди $OPEN_j$ раскрытые вершины.

Выбор очереди можно делать разными способами, например, методами обучения с подкреплением, случайно или просто по очереди. В данной работе использовался только последний метод.

Параметры алгоритма:

- w_1 – вес поисков WA*,
- w_2 – субоптимальность решения относительно A*.

4 Эксперименты

Мы сравнивали MRA* и R*, используя в качестве baseline A* и WA*. Эксперименты проводились на 8-связных 2D картах из датасета MovingAI: https://movingai.com/benchmarks/grids.html. Были выбраны 4 сравнительно разные карты.

Оценивались 3 параметра:

- **mean success rate**: показывает, сколько в среднем алгоритм за определённый лимит времени успел найти маршрутов между **start** и **goal**;
- количество раскрытых нод по длине маршрута: показывает, сколько было раскрыто в среднем узлов на различных длинах маршрута;
- коэффициент субоптимальности по длине маршрута: показывает, во сколько раз найденная длина пути превышает длину оптимального пути.

Код экспериментов доступен в репозитории.

Сравнение алгоритмов

Во всех случаях сравнивались 4 алгоритма:

1. A*

Algorithm 1 Multi-Resolution A*

procedure MAIN
  $g(s_{\text{start}}) = 0$; $g(s_{\text{goal}}) = \infty$
  $bp(s_{\text{start}}) = bp(s_{\text{goal}}) =$ **null**
  for $i = 0, \ldots, n$ **do**
    OPEN$_i \longleftarrow \emptyset$
    CLOSED$_i \longleftarrow \emptyset$
    if $i \in$ GETSPACEINDICES(s_{start}) **then**
      Insert s_{start} in OPEN$_i$ with KEY(s, i)
  while OPEN$_i \neq \emptyset$ for each $i \in \{0, \ldots, n\}$ **do**
    $i \longleftarrow$ CHOOSEQUEUE( )
    if OPEN$_i$.MINKEY() $\leq \omega_2$ * OPEN$_0$.MINKEY() **then**
      if $g(s_{\text{goal}}) \leq OPEN_i$.MINKEY() **then**
        Return path pointed by $bp(g(s_{\text{goal}}))$
      else
        s = OPEN$_i$.Pop()
        EXPANDSTATE(s, i)
        Insert s into CLOSED$_i$
    else
      if $g(s_{\text{goal}}) \leq \omega_2 * OPEN_0$.MINKEY() **then**
        Return path pointed by $bp(g(s_{\text{goal}}))$
      else
        s = OPEN$_0$.Pop()
        EXPANDSTATE$(s, 0)$
        Insert s into CLOSED$_0$

(a)

Algorithm 2 ExpandState

procedure KEY(s, i)
  if $i = 0$ **then**
    return $g(s) + h(s)$
  else
    return $g(s) + \omega_1 h(s)$
procedure EXPANDSTATE(s, i)
  for all $s' \in$ SUCCS(s, i) **do**
    if s' was never generated **then**
      $g(s') = \infty$; $bp(s') =$ **null**;
    if $g(s') > g(s) + c(s, s')$ **then**
      $g(s') = g(s) + c(s, s')$; $bp(s') = s$
      for each $i \in$ GETSPACEINDICES(s') **do**
        if $s' \notin$ CLOSED$_i$ **then**
          Insert/Update s' in OPEN$_i$ with KEY(s', i)

(b)

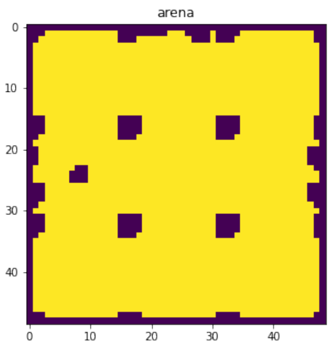
Рис. 9: Псевдокод MRA*

2. WA*(weight=3)
3. MRA*(resolutions=(1, 3), w_1=3, w_2=3)
4. R*(D=3, K=10, w=3, exp$_{\text{coeff}}$=3)

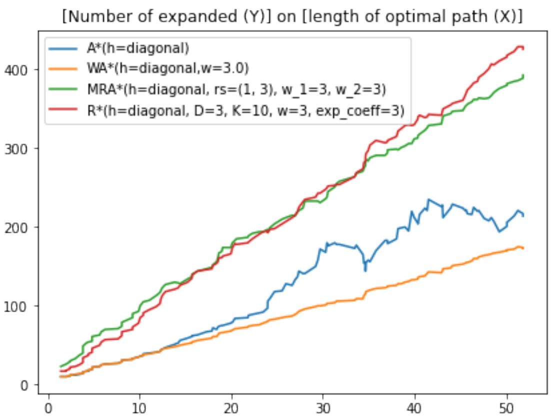
В качестве эвристики использовалась диагональная метрика: $h(dx, dy) = \sqrt{2} \cdot \min(dx, dy) + |dx - dy|$, где $dx = |x_1 - x_2|, dy = |y_1 - y_2|$.

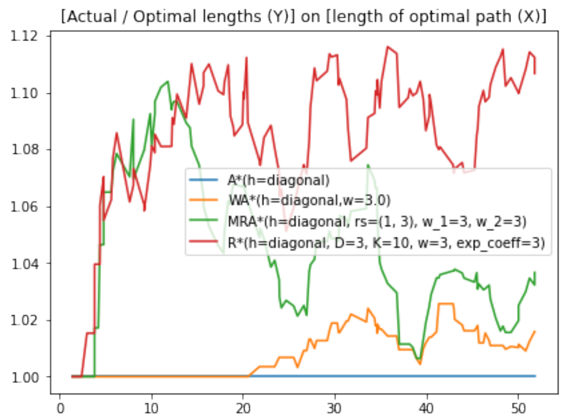
ARENA

Небольшая закрытая карта.



Результаты:

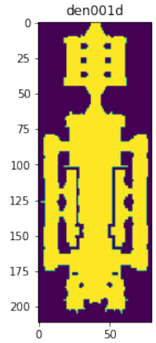




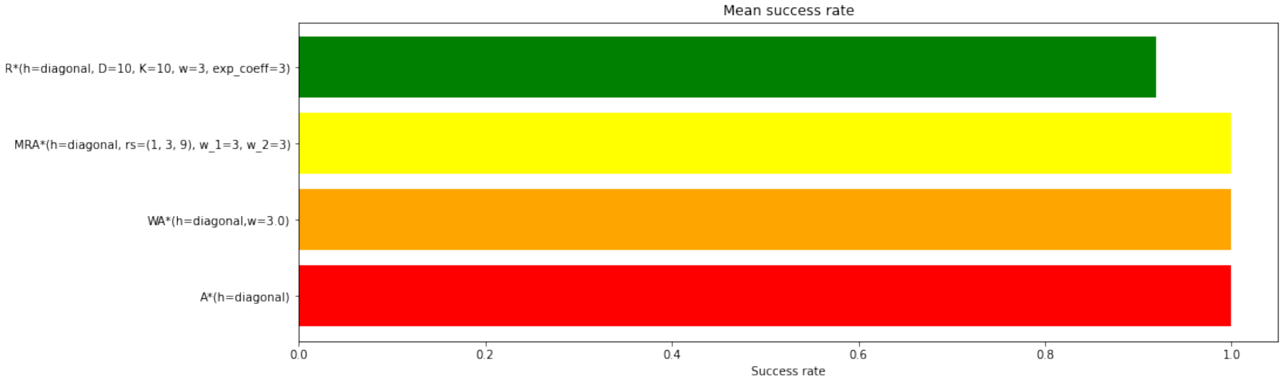
Все тесты прошли успешно, так что **Mean success rate** не приводится. Ожидаемо, WA* отрабатывает лучше чем R* и MRA*.

DEN001D

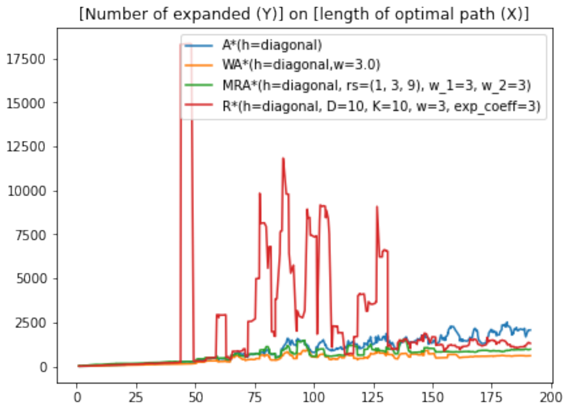
Ещё одна небольшая закрытая карта, структурно несколько более сложная чем ARENA.

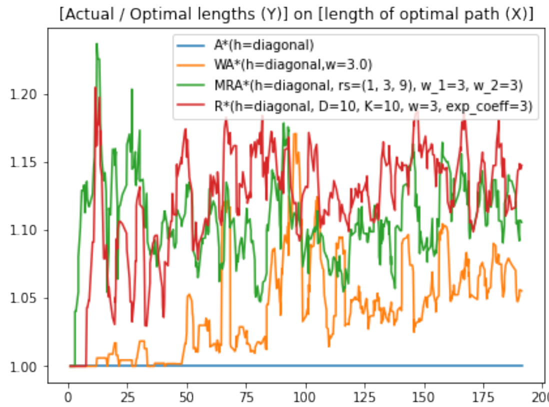


Mean success rate:



Результаты:

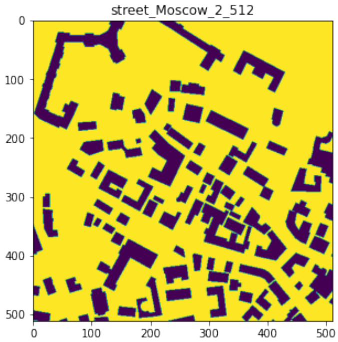




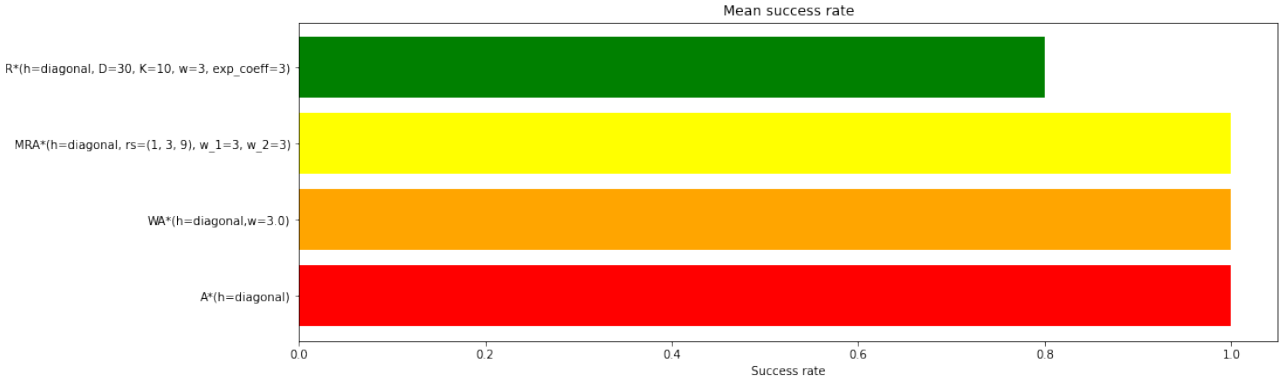
Результаты как и для ARENA, не считая того, что R* стал менее стабильно работать из-за препятствий.

MOSCOW

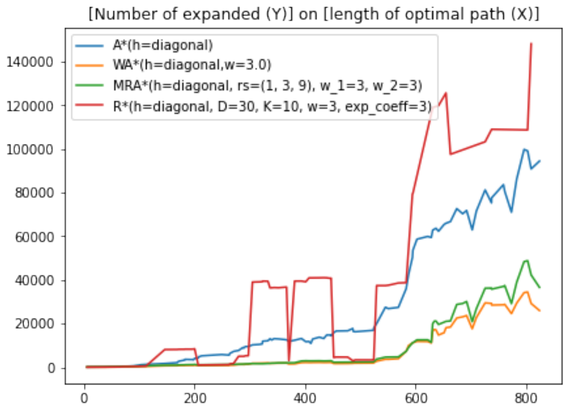
Большая открытая карта. Лимит по времени – 10 секунд.

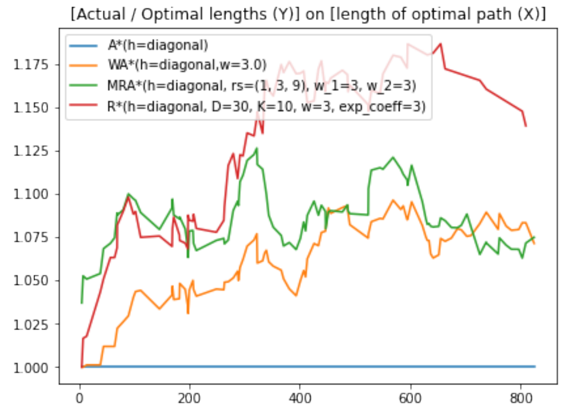


Mean success rate:



Результаты:

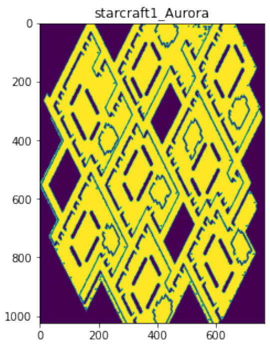




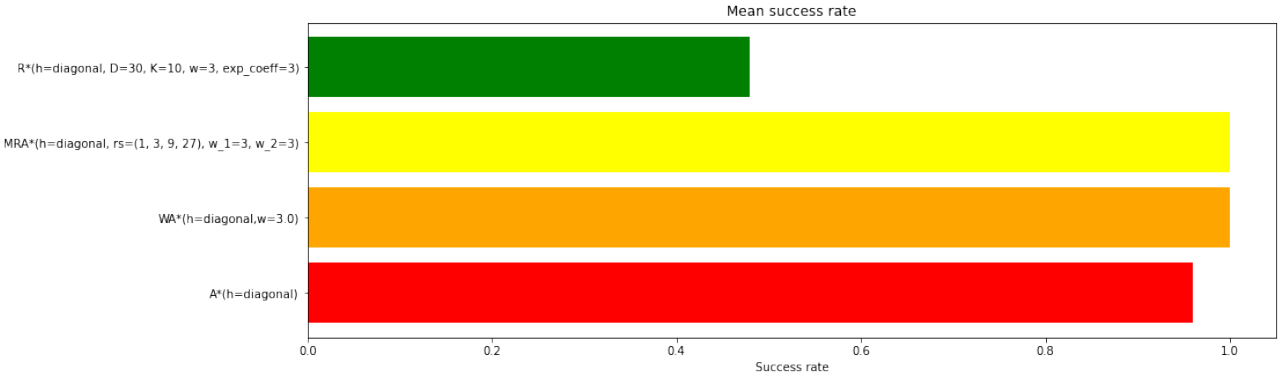
Здесь мы видим, что MRA* работает как и WA*. Судя по всему, выбор очереди с помощью обучения с подкреплением позволил бы MRA* превзойти WA*. При этом мы видим, что MRA* для больших расстояний находит более короткий путь, чем WA*.

STARCTRAFT

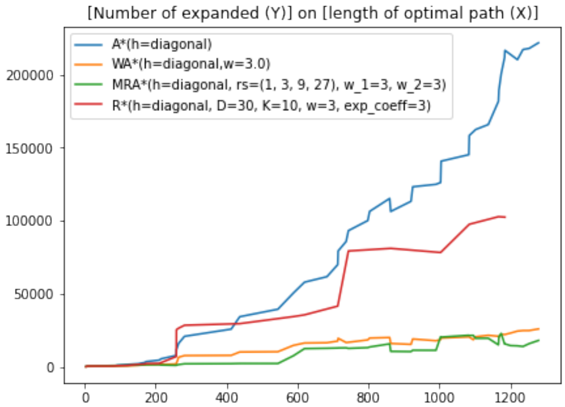
Большая закрытая карта. Лимит по времени также 10 секунд.

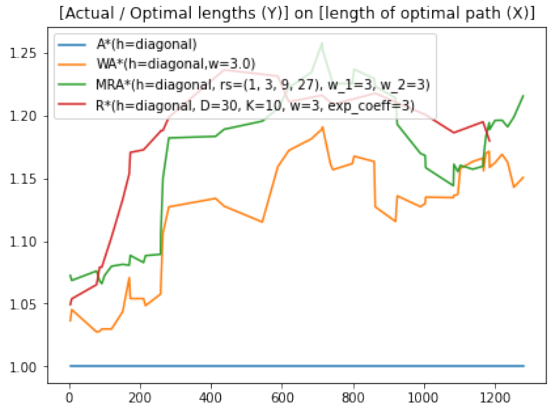


Mean success rate:



Результаты:





MRA* по сравнению с WA* работает несколько хуже, чем в предыдущем эксперименте, что ожидаемо – на закрытых картах реже можно делать большие размашистые шаги.

Зависимость от гиперпараметров

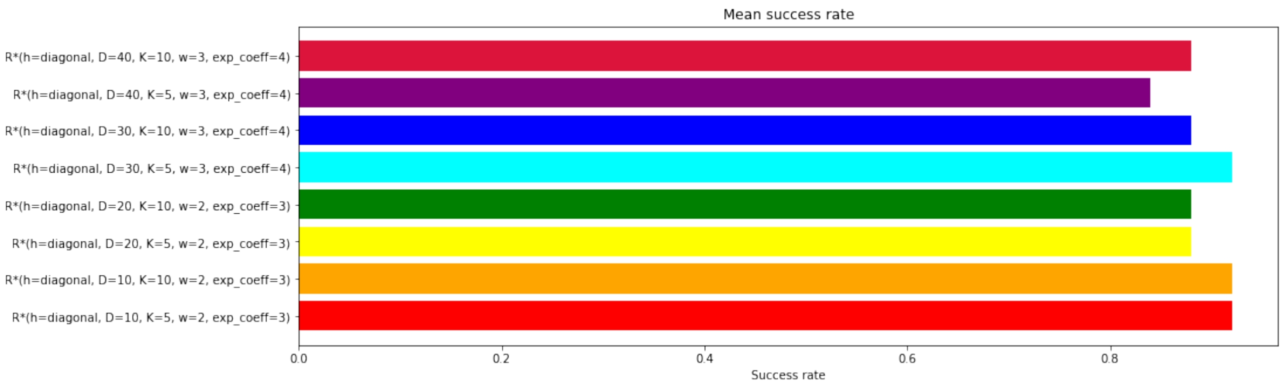
Гиперпараметры алгоритма проверялись на карте MOSCOW с лимитом по времени в 6 секундд.

R*

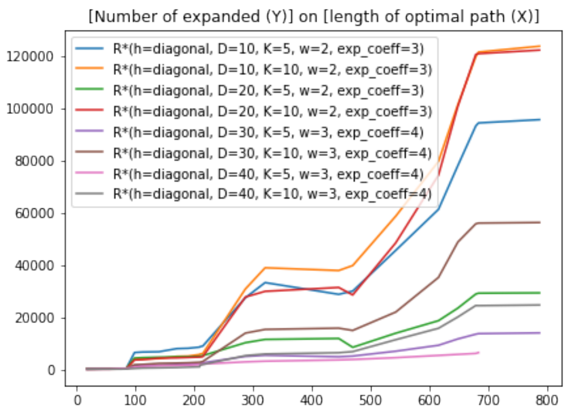
Проверяемые параметры:

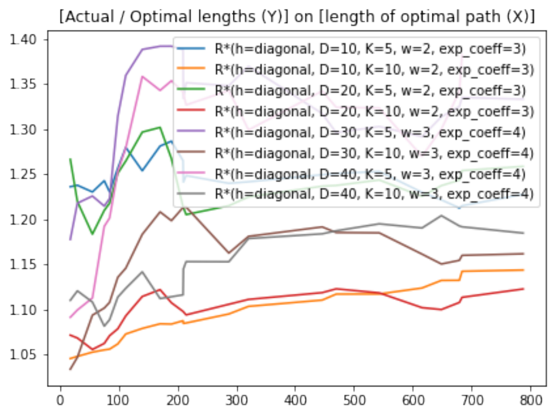
D	K	w	exp_coeff
10	5	2	3
10	10	2	3
20	5	2	3
20	10	2	3
30	5	3	4
30	10	3	4
40	5	3	4
40	10	3	4

Mean success rate:



Результаты:



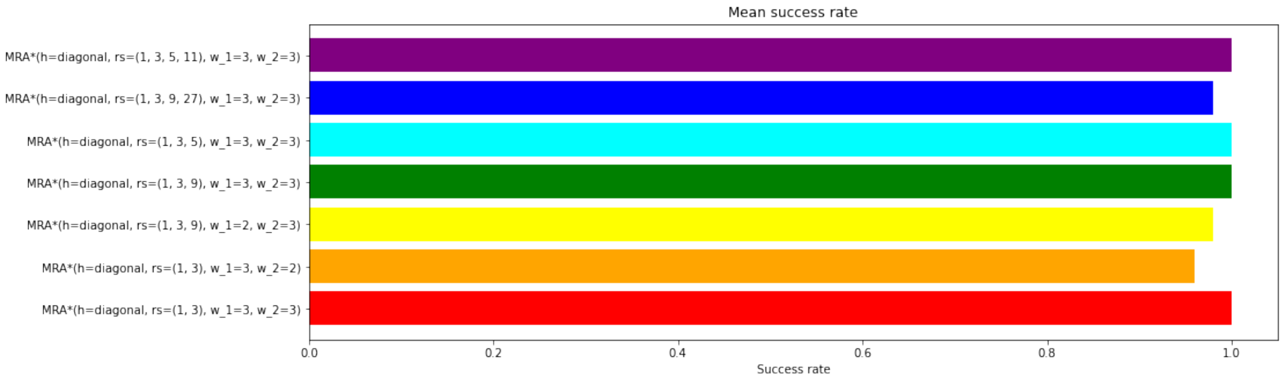


MRA*

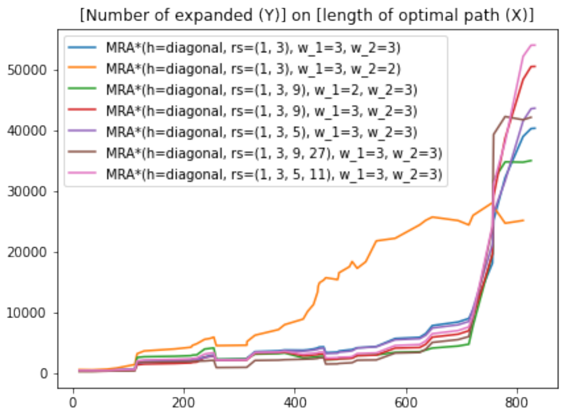
Проверяемые параметры:

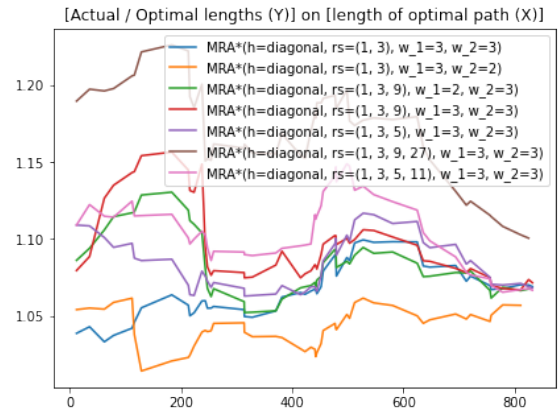
resolutions	w_1	w_2
{1, 3}	3	3
{1, 3}	3	2
{1, 3, 9}	2	3
{1, 3, 9}	3	2
{1, 3, 5}	3	2
{1, 3, 9, 27}	3	2
{1, 3, 5, 11}	3	2

Mean success rate:



Результаты:

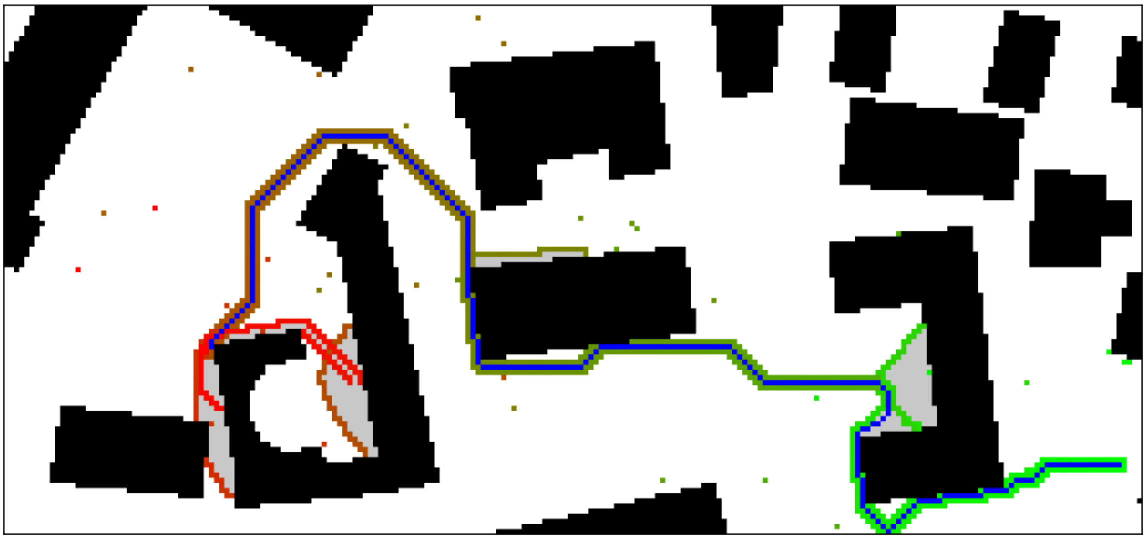




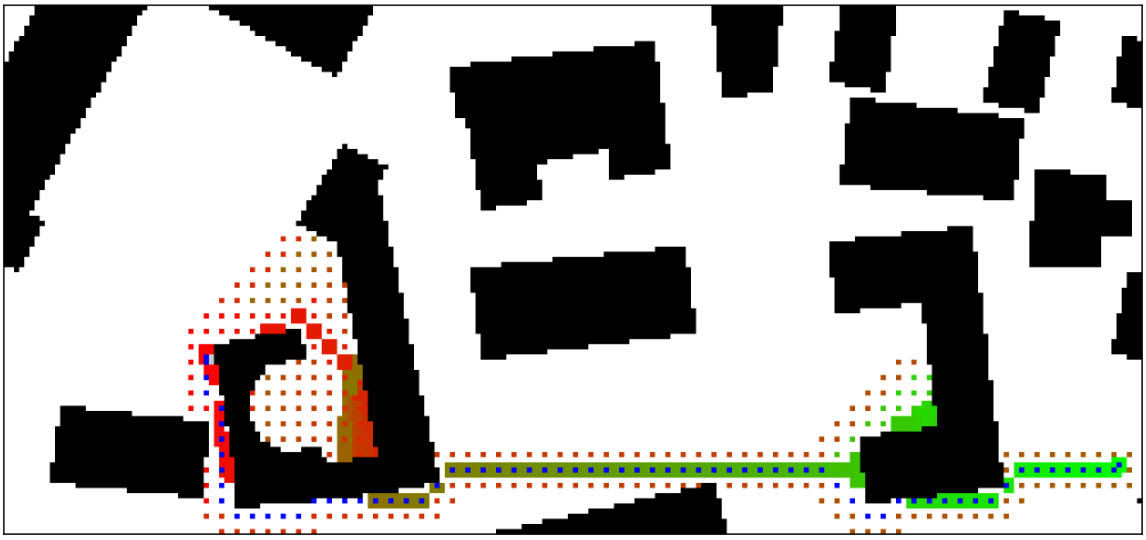
Визуализация работы алгоритмов

На визуализации
- чёрным отмечены препятствия;
- белым – проходимые области карты;
- синим – итоговый выбранный путь;
- серым – CLOSED;
- зелёным и красным – вершины из OPEN:
  - ближе к красному – добавленные ближе к началу поиска,
  - ближе к зелёному – добавленные ближе к концу поиска.

R*(h=diagonal, D=30, K=5, w=3, exp_coeff=4): op=739, cl=916, subopt=1.33



MRA*(h=diagonal, rs=(1, 3), w_1=3, w_2=3): op=359, cl=1763, subopt=1.02



На визуализации можно заметить, что алгоритм
- сначала размашисто исследует карту, быстро приблизившись к **goal**;
- уточняет путь, покуда не выполнена оценка на оптимальность.

5 Выводы

На небольших расстояниях WA* работает лучше всех и оптимально, что ожидаемо – если препятствий между **start** и **goal** почти нет, эффективнее всего просто максимально жадно двигаться к цели. MRA* при этом тратит дополнительное время на обмен состояний между очередями.

На больших расстояниях MRA* работает лучше всех, R* работает хуже всех: начинает сказываться «умение» MRA* двигаться по карте в нескольких режимах (быстром и детальном). Что касается R*, то нам не удалось добиться от него сравнимой с остальными алгоритмами производительности.

Дополнительные комментарии к R*:

- R* довольно «дисперсионный», то есть результат достаточно сильно зависит от случайного выбора алгоритмом направления;
- гиперпараметры R* требуют достаточно тонкой настройки.

Дополнительные комментарии к MRA*:

- MRA* хорошо работает на открытых картах, где часто полезны размашистые шаги на большое количество клеток;
- разрешения надо выбирать таким образом, чтобы как можно больше центров пересекалось – иначе очереди будут реже обмениваться состояниями, что плохо скажется на производительности;

Список литературы

[1] Likhachev Maxim, Stentz Anthony. R* Search. — Vol. 1. — 2008. — 07. — P. 344–350. — URL: https://www.cs.cmu.edu/~maxim/files/rstar_aaai08.pdf.

[2] Du Wei, Islam Fahad, Likhachev Maxim. Multi-Resolution A*. — 2020. — 04. — URL: https://arxiv.org/abs/2004.06684.