ФОРМИРОВАНИЕ ОПТИМАЛЬНОГО ПУТИ ДЛЯ БОЛЬШОГО КОЛИЧЕСТВА ПОГРУЗЧИКОВ НА ПРОИЗВОДСТВЕ

FORMATION OF THE OPTIMAL PATH FOR A LARGE NUMBER OF LOADERS IN PRODUCTION

R. Avkhadiev

Summary. While there is a large number of algorithms for shortest route search in a graph, all of them have a major drawback: exponential growth of operation time as the number of graph nodes and the route length increases. For practical use in production scheduling, an algorithm is required that would allow for quick estimation of the actual distance to be traveled by an object when moving from one point to another. The calculation time should be short due to vast number of cases enumerated by the system during schedule optimization. Classic algorithms involve a graph with a huge number of nodes, so long as it is possible to move in any direction within a production site (as contrasted with a highway graph). Hence, calculation time required to generate routes will be unacceptably large (by the moment of scheduling completion, the schedule will become obsolete). Therefore, in order to solve this problem, we will not use graphs, but introduce the concept of obstacles, and create a "passability" matrix (using the spatial partitioning). The route will be generated in two steps: first, go in sequence through all obstacles that prevent achieving the goal, and then optimize the generated route by "cutting the corners". A final route will be worse (longer) than ideal routes (generated by classic algorithms) by 3–16%, but the time used for its generation will be 85–920 times shorter. Of course, the accuracy is rather poor, and in most cases such route is not suitable for practical use. However, routes themselves are not needed for production scheduling, it is important just to estimate roughly the route length and displacement time by selecting and comparing a large number of cases. If routes will still be required, the considered algorithm can be used to determinate start and end points, and then the generated route may be improved using the classic algorithm.

Keywords: production scheduling, logistics, shortest path search.

Авхадиев Рустем Ахмедович
Аспирант, Казанский Национальный Исследовательский Технический Университет им. А. Н. Туполева — КАИ
psilon@inbox.ru

Аннотация. на сегодняшний день разработано значительное количество алгоритмов, позволяющих найти кратчайший маршрут в графе. При этом для большинства из них характерным является один недостаток — экспоненциальный рост периода работы при возрастании количества вершин графа и протяженности пути. Для практического применения в производственном планировании требуется такой алгоритм, который в кратчайшие сроки позволит оценить фактическое расстояние, которое необходимо будет преодолеть объекту при перемещении из одной точки в другую. Время расчета должно быть небольшим ввиду огромного количества вариантов, перебираемых системой при построении оптимального плана. Если применять классические алгоритмы, то придется строить граф с огромным количеством вершин, т.к. внутри производственной площадки возможно перемещение в любом направлении (в отличие от графа автомобильных дорог). Соответственно, время вычислений, которое потребуется на формирование маршрутов, будет неприемлемо большим (к моменту построения плана он будет уже неактуален). Поэтому, для решения этой задачи мы не будем использовать графы, а введем понятие препятствий и построим матрицу проходимости (используя пространственное разбиение). Формирование маршруту будем производить в два этапа: сначала последовательное обойдем все препятствия, мешающие достижению цели, а затем оптимизируем построенный маршрут "срезая углы". В результате, мы получим маршрут, который будет хуже (длиннее) идеальных маршрутов (построенных классическими алгоритмами) на 3–16%, но время, затраченное на построение будет меньше в 85–920 раз. Конечно, погрешность достаточно высокая и, в большинстве случаев, такой маршрут будет непригоден для реального использования. Но для производственного планирования сами маршруты и не нужны, нужно лишь при подборе и сравнении большого количества вариантов примерно оценить длину маршрута и время, которое потребуется на перемещение. Если сами маршруты все же потребуются, можно применить рассматриваемый алгоритм для выбора точек старта и финиша, а затем уже выбранный маршрут улучшить классическим алгоритмом.

Ключевые слова: производственное планирование, логистика, поиск кратчайшего пути.

Введение

При формировании графиков движения Погрузчиков возникает задача поиска кратчайшего пути между объектами производства [1], а именно:

1. Доставка материальных ресурсов до Рабочих Центров (со складов до станков, оборудования и т.д.);
2. Перемещение полуфабрикатов между Рабочими Центрами;
3. Доставка готовой продукции до склада реализации, вывоз брака и т.д.

Для обеспечения снабжения «точно в срок» [8] необходимо учитывать фактическое время перемещения между объектами. При этом, редко на каком производстве возможно перемещение «по прямой», следует

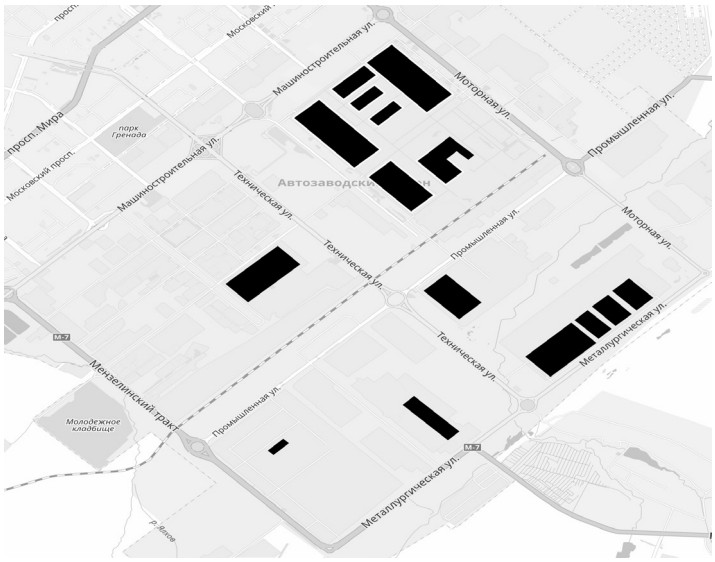

Рис 1. Пример разбиения по зонам (зоны обозначены черным)

сформировать маршрут с учетом обхода препятствий. Несмотря на то, что эта задача давно решена с использованием классической теории графов, на практике имеются ограничения быстродействия задействованных вычислительных мощностей.

Возьмем простой пример для очень маленького производства: есть 10 рабочих центров, которым за смену необходимо 10 единиц ресурсов, которые находятся на 10 складах. Перевозку осуществляет 10 погрузчиков, для упрощения будем считать, что перевозка осуществляется только по челночной схеме и только 1 единицы ресурса за раз. Для перебора всех возможных вариантов потребуется построить 10 * 10 * 10 * 10 = 10~<000 маршрутов. На реальном машиностроительном производстве десятки погрузчиков, сотни рабочих центров, тысячи складов и десятки тысяч видов ресурсов. При этом за один рейс погрузчик объезжает сразу несколько складов и несколько рабочих центров, загружаясь и выгружаясь по пути. Подробнее об этом можно прочитать в статье: Формирование плана доставки ресурсов до производственных рабочих центров предприятия. (Информационные технологии и вычислительные системы, 2 квартал 2017 г. http://www.jitcs.ru/index.php?option=com_content&view=article&id=568)

Таким образом, для планирования одного дня производства в масштабах крупного машиностроительного предприятия необходимо сформировать миллионы маршрутов на графе [9], имеющем десятки тысяч вершин. Один из наиболее эффективных алгоритмов поиска пути, алгоритм Дейкстры в его модификации A*[4], на современном сервере будет решать эту задачу несколько часов[7]. Похожий по постановке задачи, алгоритм Ли [13] (волновой алгоритм) будет работать еще дольше, т.к. даже в идеальных условиях (когда препятствий вообще нет) он переберет N^2 клеток, где

N = MAX (Расстояние между стартом и финишем по оси X, Расстояние между стартом и финишем по оси Y).

В то время, как алгоритм Дейкстры A* осуществляет направленный поиск.

Поэтому, необходимо создание более эффективного специализированного *метода поиска кратчайшего маршрута*, который будет состоять из последовательного выполнения следующих этапов:

1. формулировки математической модели;
2. определения порядка решения задач;
3. решения задачи построения прямого маршрута;
4. решения задачи обхода препятствий;
5. решения задачи оптимизации построенного маршрута.

Далее рассмотрим каждый из этапов.

Формулировка математической модели

Введем двухмерную систему координат (X, Y). Дополнительно, у каждой ячейки $a(X, Y)$; $b(X, Y)$ могут быть такие метки:

1. Проходимая (по такой метке можно двигаться в любое время без ограничений) $F^{+}(t)$.
2. Непроходимая (по этой метке невозможно проехать никогда) $F^{-}(t)$.

3. Условно проходимая (ворота, проходная и т.д.) $F^{+/-}(t)$.

Требуется сформировать минимальный по длине маршрут из ячейки $a(X, Y)$ в ячейку $b(X, Y)$ по проходимым и условно проходимым ячейкам:

$$a(X, Y) \rightarrow b(X, Y) = d_{min}(F^{+}(t); F^{+/-}(t)).$$

Порядок решения задач метода

В начале, рассматриваемый прямоугольник делится на зоны, соответствующие производственным площадкам (заводам, корпусам) (Рисунок 1). Если ячейки $a(X, Y)$ и $b(X, Y)$ размещены внутри одной зоны (A *или* B), то поиск маршрута будет производиться внутри рассматриваемой зоны.

Если $A(X, Y)$ и $B(X, Y)$ размещены в различных зонах (A *и* B), то целевой маршрут разбивается на 3 участка:

1. От ячейки $a(X, Y)$ к выходу из зоны $A(X, Y)$;
2. От выхода из зоны $A(X, Y)$ ко входу в зону $B(X, Y)$;
3. От входа в зону $B(X, Y)$ к ячейке $b(X, Y)$.

Участки 1 и 3 маршрута формируются по высокопроизводительному методу, описанному ниже.

Участок 2 оптимально формируется по методу Дейкстры[4], поскольку вершин в графе [10], соединяющим зоны будет немного, что по аналогии соответствует перекресткам и поворотам на автодороге. Кроме того, если маршрутов, соединяющих зоны, будет немного, то можно будет рассчитать заранее и кешировать.

Если входов/выходов будет больше одной пары, то нужно перебрать все комбинации входов/выходов, построить все 3 части целевого маршрута, и выбрать кратчайший маршрут.

В результате такого разбиения, скорость поиска маршрута значительно увеличивается.

В случае, если ячейки a и b находятся в одной зоне, то найденный маршрут может быть не самым оптимальным, поскольку существует вероятность того, что выгоднее покинуть свою зону и затем снова в нее вернуться. Но вероятность эта крайне мала, поэтому такой маршрут рассматривать не будем.

Задача выбора прямого маршрута

Формализуем прямой маршрут, который пролегает из ячейки a в ячейку b, минуя преграды. Алгоритм [5] состоит из следующих шагов.

Шаг 1. Определяется текущая ячейка $c = a$.

Шаг 2. Рассчитывается угол α между векторами (c_x, c_y) (c_x, b_y) и (c_x, c_y) (b_x, b_y):

$$\alpha = arctg\,(|b_x - c_x| / |b_y - c_y|),$$

либо через псевдоскалярное произведение векторов (c_x, c_y) (c_x, b_y) и (c_x, c_y) (b_x, b_y).

Поскольку данные вычисления займут значительное процессорное время, то данные вычисления производятся заранее и кешируются в двумерный массив.

Размер массива будет составлять: $max\,|b_x — a_x| * max\,|b_y — a_y|$.

Шаг 3. По углу α определяется ячейка c_N, в которую в направлении b следует осуществить «шаг» (в одну из 8 ячеек, смежных с c).

Например, если $\alpha \in [22.5, 67.5]$, тогда смещение последующего шага относительно c будет равно (1,1), если $\alpha \in [67.5, 112.5]$, то (1,0), если $\alpha \in [112.5, 157.5]$, то (1,-1) и т.д. Если $c_N = b$, то переход на *Шаг4*, иначе $\in$ присвоить $c = c_N$ и перейти на *Шаг 2*.

Шаг 4. В итоге будет сформирован прямой маршрут, который представляется в виде перечня ячеек. Если в процессе построения данного маршрута не будет выявлено ни одной преграды, то можно считать, что наилучший маршрут обнаружен. Если же препятствия выявлены, для дальнейшего конструирования маршрута следует выполнить обход преград.

Задача обхода преград

Для обхода препятствии используем следующий *алгоритм [6]*:

Шаг 1. Определение стартовой проходимой ячейки s, которая предшествовала столкновению с преградой в сформированном прямом маршруте. Обозначается ячейка столкновения с препятствием буквой O.

Шаг 2. Вычисляется «угол столкновения» α, который равен углу между векторами (s_x, s_y) (s_x, O_y) и (s_x, s_y) (O_x, O_y). Вычисления произведем аналогично *Шагу 2* алгоритма из п. 4.

Шаг 3. Вычитается 45° из α. По полученному углу α определяется смежная с s ячейка (аналогично *Шагу3* алгоритма из п. 4).

Шаг 4. Если данная ячейка является проходимой, то делается следующий «шаг» в нее. Переход на *Шаг 6.*

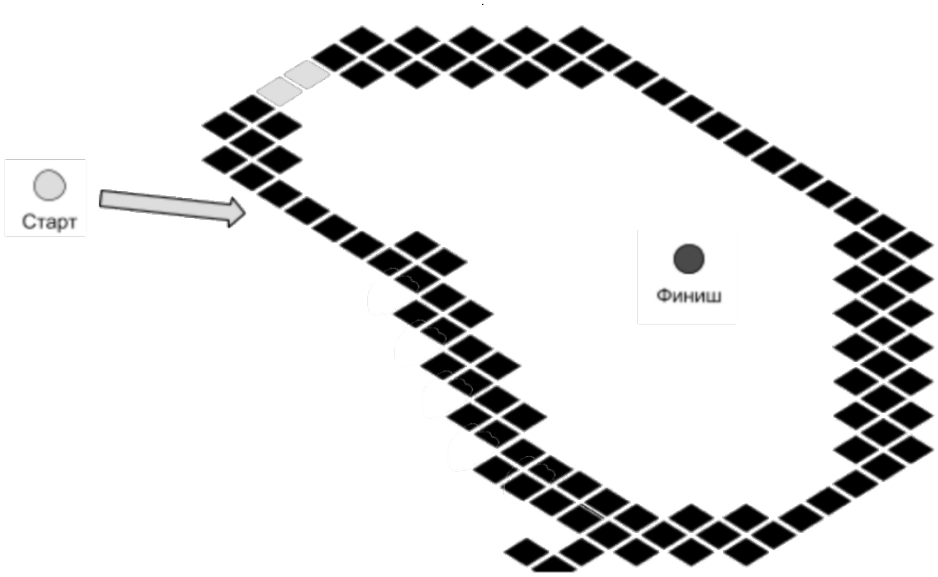

Рис 2. Препятствие, которое невозможно обойти. Желтыми квадратами обозначены ворота, которые в момент поиска маршрута закрыты

Шаг5. Если α совершен полный оборот, а проходимая ячейка так и не установлена, маршрут не может быть построен. Алгоритм в таком случае завершается.

Иначе, переход на *Шаг 3*.

Шаг 6. s заменяется на найденную на *Шаге 4* проходимую ячейку, которая записывается в скорректированный маршрут.

Шаг 7. Если соблюдено условие достижения «отрыва» от преграды (рассмотрено далеее), то Алгоритм завершается.

Шаг 8. Если *s* принял стартовое значение (то же, что на *Шаге 1*), значит случилось зацикливание и обойти препятствие невозможно, Алгоритм завершен.

Шаг 9. Угол α корректируется так, чтобы «шаг» по направлению к нему снова спровоцировал столкновение с преградой (что необходимо во избежание случайного «отрыва» от препятствия). Вычисляется $\alpha = 90° + \alpha$ и проверяется смежная с *s* ячейка в направлении α на проходимость.

Шаг 10. Если ячейка принадлежит преграде, осуществляется переход к *Шагу 3*. Иначе, вычисляется $\alpha = 45° + \alpha$, и так действия повторяются до тех пор, пока ячейка преграды не будет установлена. После этого осуществляется переход к *Шагу3*.

Шаг 11. В результате будет установлен обход преграды «с левой стороны». Таким же образом необходимо обнаружить обход «с правой стороны». С этой целью необходимо инвертировать знак операций с углом α (на *Шаге 3* прибавлять, а на *Шаге10* вычитать). Из найденных обходов следует выбрать минимальное значение.

Существует ситуация, при которой препятствие невозможно обойти. Например, если проход к целевой ячейке возможен только через одни ворота, которые в момент поиска маршрута оказались закрыты. В этом случае найти маршрут к целевой ячейке невозможно.

Условия прекращения обхода препятствия

1. *Простое условие*. Если текущий «шаг» обхода ближе к конечной цели, чем предыдущий, то следует завершить обход. Данное условие будет корректно срабатывать для подавляющего большинства препятствий, но не для всех. Пример приведен на Рисунке 3.

2. Надежное условие. Если текущий «шаг» обхода сходится хотя бы с одной ячейкой, принадлежащей прямому маршруту, то следует завершить обход. Проверка этого условия более трудоемкая и потребуется либо существенный расход процессорного времени на перебор списка ячеек маршрута, либо расход памяти и некоторый расход процессорного времени для проецирования маршрута на служебную матрицу размером во всю площадь поиска [12]. Однако, данное условие будет корректно срабатывать для всех видов препятствий.

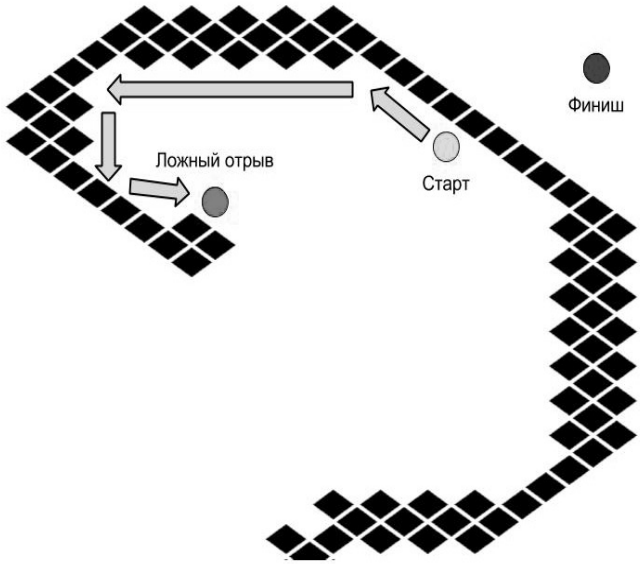

Рис 3. Преждевременный отрыв от препятствия при обходе

Задача оптимизации построенного маршрута

Для полученного маршрута будет характерен следующий недостаток: с начала он встретит препятствие, а потом будет обходить его.

Очевидно, что маршрут будет короче, если обход будет начинаться заблаговременно, до встречи с препятствием.

Для сокращения протяженности построенного маршрута применяется следующий *Алгоритм*:

Шаг1. Определяется стартовая ячейка c, равная первой ячейке построенного маршрута, а также проверяемая ячейка d, равная предпоследней ячейке полученного маршрута.

Шаг2. Прокладывается прямой маршрут от c до d по Алгоритму п. 4.

Шаг3. Если при выборе маршрута не встретилось преград, то следует заменить часть построенного маршрута (от c до d) прямым маршрутом. Приравнивается $c = d$, при этом d равна предпоследней ячейке построенного маршрута. Если c и d совпали, то переход на *Шаг6*, иначе переход к *Шагу2*.

Шаг4. Приравнивается d = предшествующая ей ячейка маршрута (с индексом, меньше на 1). Если c и d совпали, то c приравнивается следующей ячейке маршрута (с индексом, большим на 1).

Шаг5. Если c совпала с предпоследней ячейкой маршрута, то переход на *Шаг6*, иначе переход к *Шагу2*.

Шаг6. Фиксация сокращенного маршрута.

Предложенный алгоритм дает возможность «срезать углы» в маршруте, в результате, будет построен маршрут следующего вида (Рисунок 5).

Сравнение алгоритмов

Для оценивания целесообразности и эффективности использования предложенного алгоритма последовательного обхода преград, представляется целесообразным сравнить его с уже разработанными алгоритмами. Для сравнения используем наиболее распространенные и эффективные алгоритмы:

- Алгоритм A star [2]
- Алгоритм Дейкстры [4]
- Алгоритм Левита [3]

Методика проведения испытаний

Выберем случайным образом ячейки старта и финиша (A и B) и построим маршруты между ними, используя каждый из 4-х сравниваемых алгоритмов. Для чистоты эксперимента, все 4 алгоритма [11] строят маршруты между одними и теми же ячейками A и B. Затем, случайным образом выбираются новые A и B и снова подаются на вход всех 4-х алгоритмов. Полученные результаты сравним по 2-м параметрам:

- Время работы алгоритма
- Длина построенного маршрута

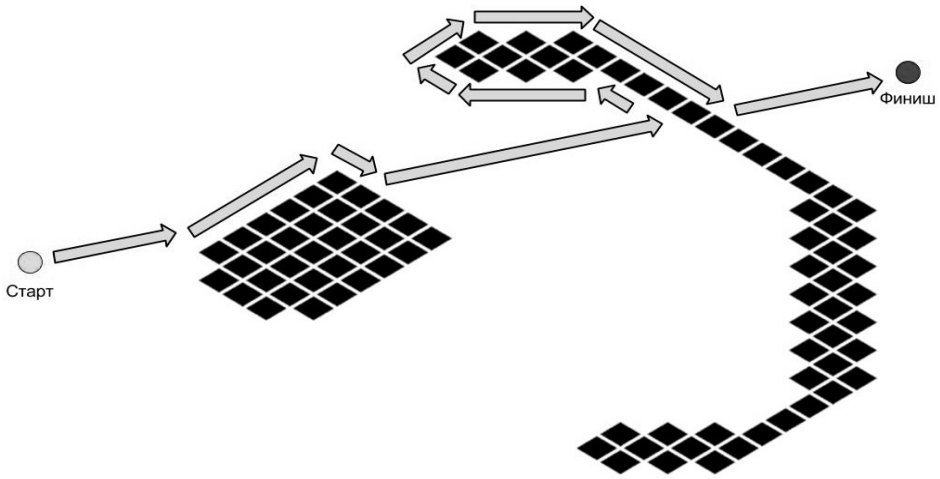

Рис 4. Маршрут не предусматривающий последующую обработку (оптимизацию)

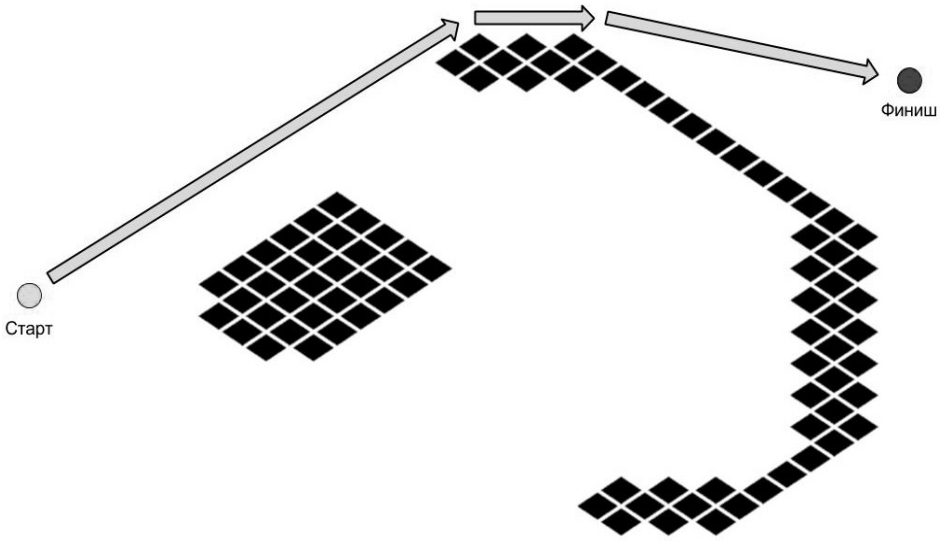

Рис 5. Оптимизированный маршрут

Генерацию ячеек старта и финиша выполним в 2-х вариантах:

- Внутри одной зоны. Расстояние между стартом и финишом "по прямой": 50.100, 100.200, 200.500, 500.1000 метров
- В разных зонах. Расстояние между стартом и финишом 2000–5000 м.

Количество построенных маршрутов: 40000 (по 10000 маршрутов на каждый алгоритм).

Используемое оборудование:

CPU: 4 GHz Intel Core i7 (задействовано 1 ядро, без многопоточности);
RAM: 8 Gb 1867 MHz DDR3

Результаты испытаний

Формирование маршрутов внутри одной зоны — см. таблицы 1—4 и рис.6.

Формирование маршрутов для разных зон — см. таблицу 5.

Пояснение: снижение эффективности алгоритма связано с необходимостью построения и сравнения маршрутов до всех выходов из зоны старта и от всех входов в зону финиша. Если этого не делать и строить только 2 варианта маршрута: до ближайших к старту/финишу выхода/входа и до входа и выхода, между которыми минимальное расстояние, то качество построенных маршру-

Таблица 1. Результаты для маршрутов 50–100 метров:

Алгоритм	Средняя протяженность маршрута (км)	Разница с лидером в длине	Время работы (с)	Разница с лидером по времени
Последовательный обход	0,1841	2.8%	0.001	0%
AStar	0,1804	0.7%	0.011	10~<000%
Дейкстра	0,1790	0%	0.79	78~<900%
Левит	0,1790	0%	0.93	92~<900%

Таблица 2. Результаты для маршрутов 100–200 метров:

Алгоритм	Средняя протяженность маршрута (км)	Разница с лидером в длине	Время работы (с)	Разница с лидером по времени
Последовательный обход	0,4389	8.2%	0.047	0%
AStar	0,4151	2.3%	1.98	4~<112%
Дейкстра	0,4057	0%	5.99	12~<644%
Левит	0,4057	0%	8.1	17~<134%

Таблица 3. Результаты для маршрутов 200–500 метров:

Алгоритм	Средняя протяженность маршрута (км)	Разница с лидером в длине	Время работы (с)	Разница с лидером по времени
Последовательный обход	0,9410	16.5%	0.17	0%
AStar	0,8484	5%	9.5	5~<488%
Дейкстра	0,8074	0%	22	12~<841%
Левит	0,8074	0%	28	16~<370%

Таблица 4. Результаты для маршрутов 500–1000 метров:

Алгоритм	Средняя протяженность маршрута (км)	Разница с лидером в длине	Время работы (с)	Разница с лидером по времени
Последовательный обход	1,1557	9%	0.28	0%
AStar	1,0732	1%	20	7~<043%
Дейкстра	1,0586	0%	38	13~<471%
Левит	1,0586	0%	46	16~<328%

Используемая карта проходимости:

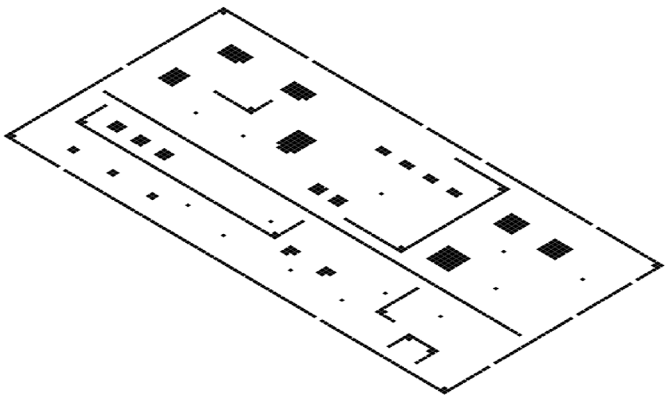

Рис. 6. Карта проходимости одной зоны

Таблица 5.Результаты для маршрутов 2000–5000 метров

Алгоритм	Средняя протяженность маршрута (км)	Разница с лидером в длине	Время работы (с)	Разница с лидером по времени
Последовательный обход	5,8391	4.7%	5	0%
AStar	5,6063	0.8%	42	740%
Дейкстра	5,5629	0%	432	8~<540%
Левит	5,5629	0%	481	9~<520%

тов упадет на несколько процентов, но скорость работы вырастет на порядок.

Заключение

Предложенный алгоритм построения маршрута с помощью последовательного обхода преград дает возможность за короткий промежуток времени (в среднем, в 100 раз быстрее, чем у традиционных алгоритмов) выбрать маршрут приемлемого качества (на 3% — 16% продолжительнее идеального).

Алгоритм более быстро и качественно функционирует на небольших дистанциях (т.к. большинство маршрутов выбираются вообще без учета встречи с преградой). При использовании на средних расстояниях эффективность алгоритма снижается, т.к. используются инструменты обхода препятствий и последующей корректировки маршрута. На значительных расстояниях сложность обычных алгоритмов возрастает экспоненциально в зависимости от числа обработанных вершин, а у предложенного алгоритма сложность увеличивается линейно. Таким образом, его можно использовать для выбора маршрута практически на неограниченные дистанции.

Такого качества маршрутов вполне достаточно для задач производственного планирования, т.к. требуется лишь сформировать график заданий для участников производственного процесса. Само фактическое перемещение и выбор маршрута будет производиться человеком-исполнителем задания. Система планирования должна лишь с некоторой точностью определить кто “ближе” и кому “выгоднее” дать то или иное задание.

ЛИТЕРАТУРА

1. Миротин Л.Б., Некрасов А. Г., «Логистика интегрированных цепочек поставок». М.: Издательство «Экзамен», 2003.
2. Левитин А. В. Глава 9. Жадные методы: Алгоритм Дейкстры // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. ISBN978–5–8459–0987–9.
3. Б. Ю. Левит. Алгоритмы поиска кратчайших путей на графе. Труды института гидродинамики СО АН СССР. Сб. «Моделирование процессов управления». Вып. 4. Новосибирск. 1971.
4. Dijkstra E. W. A note on two problems in connexion with graphs // Numer. Math — Springer Science+Business Media
5. Cucker, Felipe; Bürgisser, Peter (2013). “A.1 Big Oh, Little Oh, and Other Comparisons”. Condition: The Geometry of Numerical Algorithms. Berlin, Heidelberg: Springer. ISBN978–3–642–38896–5.
6. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001) [1990]. Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill. pp. 41–50. ISBN0–262–03293–7.
7. Black, Paul E. (11 March 2005). Black, Paul E., ed. “big-O notation”. Dictionary of Algorithms and Data Structures. U. S. National Institute of Standards and Technology. Retrieved December 16, 2006.
8. Сербин В. Д. Основы логистики. Учебное пособие. Таганрог: Изд-во ТРТУ, 2004.
9. Дистель Р. Теория графов Пер. с англ. — Новосибирск: Издательство института математики, 2002. — 336 с. ISBN5–86134–101-X.
10. Кормен Т. Х. и др. Часть VI. Алгоритмы для работы с графами // Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: Вильямс, 2006. —ISBN0–07–013151–1.
11. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ = Introduction to Algorithms. — 2-е изд. — М.: «Вильямс», 2006. —ISBN0–07–013151–1.
12. Steering Behaviors For Autonomous Characters.Режим доступа: http://www.red3d.com/cwr/steer/(дата обращения 17.04.17).
13. Lee, C. Y. (1961), “An Algorithm for Path Connections and Its Applications”, IRE Transactions on Electronic Computers, EC-10 (2): 346–365