OVERVIEW OF ALGORITHMS FOR FINDING THE OPTIMAL ROUTE FOR VEHICLES

DOI: 10.36724/2072-8735-2020-14-2-52-56

Julia V. Mahovikova,
Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, Russia, ***mahovikova1994@gmail.com***

Svetlana N. Mironenko,
Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, Russia, ***sn-mironenko@mail.ru***

Aleksandr V. Devjatkov,
Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, Russia, ***dav009@bk.ru***

Jaroslav I. Shamlitskiy,
Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, Russia, ***2538357@mail.ru***

Keywords: *suboptimal algorithms, A* algorithm, Theta* algorithm, route planning, vehicle.*

Cargo transportation management is impossible without quality planning, which should be aimed at efficient use of vehicles. The key tasks of transportation management are cargo routing and distribution of vehicles along routes, provided that the transportation plan is fulfilled in accordance with the selected optimization criteria. The modern development of information and communication technologies allows us to significantly improve the quality of planning and monitoring the execution of orders for the transportation of goods by land transport. Equipping drivers of cargo vehicles with satellite navigation devices and terminals with Internet access provides a technical opportunity for information interaction with dispatchers in real time, which determines new requirements for intelligent transport resource planning systems taking into account the human factor [9]. When creating simulators that involve moving different types of vehicles over large areas, taking into account the current tactical situation, there are problems with choosing the optimal path search algorithm, since its use is subject to restrictions. There are a large number of algorithms that allow you to determine the route by which you can get from one point to another. The main problem with the path search problem is that there is no universal algorithm for solving it. An overview of algorithms for finding the optimal path for vehicles (the Algorithm A* and its modifications, in particular Beam search; Iterative deepening; Dynamic weighing; Bidirectional search; Bandwidth search; Jump Point Search; Theta*). It is concluded that it is advisable to use different algorithms at the stages of building a preliminary route variant and optimizing it.

Information about authors:
Julia V. Mahovikova, *Reshetnev Siberian State University of Science and Technology, graduate student, Krasnoyarsk, Russia*
Svetlana N. Mironenko, *Reshetnev Siberian State University of Science and Technology, graduate student, Krasnoyarsk, Russia*
Aleksandr V. Devjatkov, *Reshetnev Siberian State University of Science and Technology, graduate student, Krasnoyarsk, Russia*
Jaroslav I. Shamlitskiy, *Reshetnev Siberian State University of Science and Technology, Candidate of Engineering Sciences (Ph.D), associate professor Department of Information Management Systems, Krasnoyarsk, Russia*

Для цитирования:
Маховикова Ю.В., Мироненко С.Н., Девятков А.В., Шамлицкий Я.И. Обзор алгоритмов поиска оптимального маршрута транспортных средств // T-Comm: Телекоммуникации и транспорт. 2020. Том 14. №2. С. 52-56.

For citation:
Mahovikova J.V., Mironenko S.N., Devjatkov A.V., Shamlitskiy J.I. (2020) Overview of algorithms for finding the optimal route for vehicles. *T-Comm*, vol. 14, no.2, pp. 52-56. (*in Russian*)

Introduction

Suboptimal algorithms work on the principle of step-by-step improvement of the current result. Some heuristic function is selected. you can use it to select a grid cell at each step, the distance from which to the end point will have the minimum value (based on the value of the heuristic function). The advantage of such algorithms is a smaller number of resources used compared to algorithms for determining the optimal path. Many heuristic algorithms allow you to give out the point that is presumably closest to the end point and the route to it, which is important for real-time modeling algorithms, when you need to get some part of it instead of the end path [5].

When creating simulators that involve moving different types of objects over large territories, taking into account the current tactical situation, there are problems with choosing the optimal path search algorithm, since its use is subject to restrictions caused by the following factors:

• a large amount of data from real maps of the area, exceeding the amount of RAM, so in most cases it is not possible to store full information about the intermediate state of the route in memory;

• complexity of the representation of the territory where objects are moving, this requires minimizing the number of requests to determine the patency of a certain section of the path;

• large variation in the complexity of the resulting path: the optimal solution may be either a straight line or a strongly broken line [5].

1. Path search algorithms for a vehicle

In this paper, based on a test analysis, it was proposed to include such path search algorithms in the library:

• Algorithm A*;
• Beam search (beam search);
• Iterative deepening (interactive immersion);
• Dynamic weighing (using variable weights);
• Bidirectional search (bidirectional search);
• Bandwidth search (search bandwidth);
• Jump Point Search (the search for the transition point);
• Theta*.

A well-designed route and control automation will significantly save fuel resources and, consequently, reduce environmental damage, as well as minimize accidents and other accidents [7]. More and more different research centers around the world are developing this direction.

The task of finding a path is reduced to analyzing the obstacles that may occur on the trajectory of the movement, and their rational circumvention. For this task, many algorithms have been developed that are actively used in laying networks, distributing printed circuit boards, and moving objects in computer games. Creating simulators that allow you to simulate the behavior of a vehicle in a changing environment reduces the need for expensive experiments [7, 8].

A big problem is the speed with which traffic calculations and route selection analysis will be performed, which imposes a certain limit on the choice of the optimal path search algorithm. The task of implementing a path search using the minimum possible amount of memory and calculation time remains relevant.

Analysis of the direction of movement of the vehicle implies the presence of data about its location and the surrounding environment. The vehicle has a clearly fixed set of options to continue driving: increase speed, decrease speed, continue driving, reverse, turn right/left.

The analysis of existing algorithms showed that not all of them are suitable for this type of problem. Real-time systems are limited in response speed by the acceptable threshold of 100 m/s [2]. It is also necessary to take into account the fact that the vehicle can't instantly change course. The speed of the "reaction" to a change in the position of the steering mechanism depends on the mass of the vehicle, its adhesion to the surface, and other factors.

Based on the above, the task of finding a path can be divided into 2 stages:

– building an optimal route using an algorithm that satisfies the necessary restrictions on the use of resources and time;

– post-processing the results of the first stage to get realistic indicators.

Depending on the choice of algorithm, these two steps can be combined into one if data optimization is provided during the path search. The second stage is necessary because the vehicle cannot change its trajectory instantly. And, therefore, sharp turns in the route provided by the first stage must be avoided.

2. Description of path search algorithms

Algorithm A*

The algorithm A* is one of the most well-known suboptimal path search algorithms [3]. It finds the route from the initial vertex to the final one with the lowest cost. The order of traversal is determined by the «distance + cost» heuristic function: $f(x) = g(x) + h(x)$. The function $h(x)$ must be a valid heuristic estimate, that is, it must not overestimate the distance to the target vertex. The function can represent the distance to the target in a straight line, since it is the smallest distance between two points [4].

The A* algorithm looks through all the paths from the initial vertex to the final one step by step until it finds the minimum one. Like all informed search algorithms, it first searches the routes that appear to be most likely leading towards the goal. From the greedy algorithm (the algorithm for finding the shortest distance by selecting the shortest, not the selected edge, provided that it does not form a cycle with already selected edges, which is also the search algorithm best-first match) it is different in that when you select a vertex it takes into account, among others, all passed before her path (component $g(x)$ is the cost of the path from the initial vertex, and not from the previous one, as in a greedy algorithm) [5].

The algorithm A* is complete, that is, it always finds a solution if it exists. It is also optimally efficient for a given $h(x)$ heuristic. This means that any other algorithm explores at least as many nodes as the A* algorithm (except when there are several particular solutions with the same heuristics that exactly match the cost of the optimal path) [5].

While the A* algorithm is optimal for "randomly" defined graphs, there is no guarantee that it will do its job better than simpler, but also more informed algorithms about the problem area. For example, in some maze, you may need to first go in the direction of the exit, and only then turn back. In this case, the initial survey of those vertices that are located closer to the exit (in a straight line) will be a waste of time [5].

Analysis of known modifications of the algorithm.

In cases of routing for real systems, this algorithm has a number of disadvantages.

First, if the field is large enough to occupy thousands of cells, there may be a problem with the amount of memory used by lists.

Second, the time to search for a path may exceed the maximum allowed response time.

Third, the algorithm does not find the shortest path.

To eliminate these shortcomings, the following modifications of the A* algorithm and other suboptimal search heuristic algorithms have been developed [6].

Beam search.

In classic A*, all vertices that may be required to perform a search are saved to the open list. This modification imposes restrictions on the size of this list. That is, after the open list is completely filled in to add the next cells to it, the ones that are least likely to be used are deleted first. Using this modification, you can avoid problems with the amount of memory used.

In the main A* loop, the OPEN set stores all the nodes that may need to be searched to find a path. The Beam Search is a variation of A* that places a limit on the size of the OPEN set. If the set becomes too large, the node with the worst chances of giving a good path is dropped. One drawback is that have to keep your set sorted to do this, which limits the kinds of data structures you'd choose [4].

Iterative deepening.

Iterative Deepening is an approach used in many AI algorithms to start with an approximate answer, and then make it more accurate. The name comes from the search for a decision tree from game theory, where you need to look at a certain number of moves ahead. You can try to deepen the tree by looking ahead more moves. Once your answer doesn't change or improve much, you assume that you have a pretty good answer, and it won't improve when you try to make it more accurate again. In IDA*, the «depth» is a cutoff for f values. When the f value is too large, the node won't even be considered (i.e., it won't be added to the OPEN set). The first time through process very few nodes. Each subsequent pass, you increase the number of nodes visit. If find that the path improves, then you continue to increase the cutoff; otherwise, you can stop.

The idea of this method is to check the next few steps of the algorithm and, if there is no improvement in the result, stop searching in this direction. Continuing the search until there is an improvement is made by the value f, which sets the value of the dive. Thus, a small number of nodes will be carried out during the first pass, and their number will increase during subsequent passes. Therefore, if there are no obstacles or their circumvention does not exceed the value of the immersion f, the number of cells involved in the analysis will be less than in the usual algorithm A*. This modification in certain cases will give a significant gain in time and the amount of memory used.

Dynamic weighting.

This modification assumes that at the beginning of the search, you need to reach the area containing the final cell of the path as quickly as possible. At the end of the search, reaching a specific cell is more important. The following modification of the weight function is proposed:

$$f(p) = g(p) + w(p) * h(p) \quad (1)$$

where $w(n)$ is the weight assigned to node n.

When approaching the final cell, the weight decreases. This reduces the importance of the heuristic and increases the relative importance of the actual path cost.

Bidirectional search.

Instead of searching from the start to the finish, you can start two searches in parallel-one from start to finish, and one from finish to start. When they meet, you should have a good path

It's a good idea that will help in some situations. The idea behind bidirectional searches is that searching results in a «tree» that fans out over the map. A big tree is much worse than two small trees, so it's better to have two small search trees.

The front-to-front variation links the two searches together. Instead of choosing the best forward-search node – $g(start, x)$ + h(x, goal) – or the best backward-search node-$g(y, goal)$ + $h(start, y)$ – this algorithm chooses a pair of nodes with the best $g(start, x) + h(x, y) + g(y, goal)$.

The retargeting approach abandons simultaneous searches in the forward and backward directions. Instead, it performs a forward search for a short time, chooses the best forward candidate, and then performs a backward searchÏ not to the starting point, but to that candidate. After a while, it chooses a best backward candidate and performs a forward search from the best forward candidate to the best backward candidate. This process continues until the two candidates are the same point.

Bandwidth of search.

There are two properties about Bandwidth Search that some people may find useful. This variation assumes that h is an overestimate, but that it doesn't overestimate by more than some number e. If this is the case in your search, then the path you get will have a cost that doesn't exceed the best path's cost by more than e. Once again, the better you make your heuristic, the better your solution will be.

Another property you get is that if you can drop some nodes in the OPEN set. Whenever $h+d$ is greater then the true cost of the path (for some d), you can drop any node that has an f value that's at least $e+d$ higher than the f value of the best node in OPEN. This is a strange property. You have a «band» of good values for f; everything outside this band can be dropped, because there is a guarantee that it will not be on the best path.

Curiously, you can use different heuristics for the two properties, and things still work out. You can use one heuristic to guarantee that your path isn't too bad, and another one to determine what to drop in the OPEN set.

Jump Point Search.

Many of the techniques for speeding up A* are really about reducing the number of nodes. In a square grid with uniform costs it's quite a waste to look at all the individual grid spaces one at a time. One approach is to build a graph of key points (such as corners) and use that for path finding. However, you don't want to precompute a waypoint graph, look at Jump Point Search, a variant of A* that can skip ahead on square grids. When considering children of the current node for possible inclusion in the OPEN set, Jump Point Search skips ahead to faraway nodes that are visible from the current node.

Each step is more expensive but there are fewer of them, reducing the number of nodes in the OPEN set.

Theta*

The main problem of finding a path is the distance of the selected route from the optimal obstacle avoidance. The difference between this modification is that Theta* allows you to select any cell as the «parent» for each cell, unlike A*, where only the neighboring cell can be the" parent". Using this modification will allow you to get more realistic indicators of path search, but it requires more time and additional memory costs [5].

A* will work faster and create better paths if you give a graph of key points (such as corners) instead of a grid. However, if you don't want to pre-calculate the angle graph, you can use Theta*, a variant of A* that works on square grids, to find paths that don't strictly follow the grid. When building parent pointers, Theta* will point directly to the ancestor if there is a line of sight to that node, and skips nodes between them. Unlike path smoothing, which aligns paths after they are found by A*, Theta* can analyze these paths as part of the A*process. This can result in shorter paths than post-processing the grid path into a path at any angle.

Comparison of algorithms

Many algorithms have a similar problem: the paths they produce look unrealistic. To solve this problem, you must either use subsequent path optimization, or use an algorithm that already uses functions that allow you to get a realistic picture [5].

For Fig. 1 shows an example of finding a path between two points using the A* and Theta* algorithms. The picture shows that the path obtained using Theta* is shorter and also looks more realistic. However, the Theta* algorithm is quite heavy due to the large number of calls to the terrain. If used in the detection of obstacles on the straight line between two points to apply to the result of the algorithm A*, you get the path of realism is close to the result of the algorithm Theta*, while the overhead will be much smaller [5].

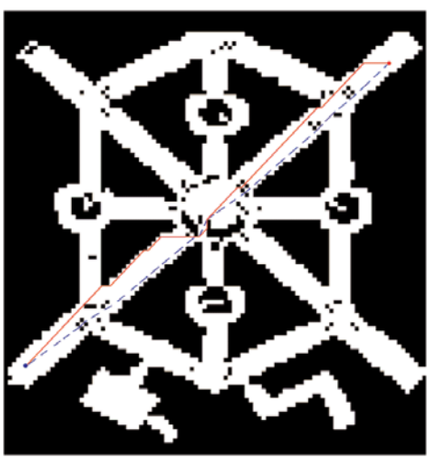

Fig. 1. Comparing path A* (red line) to path Theta* (blue line)

Conclusion

The main problem with the path search problem is that there is no universal algorithm for solving it. At the same time, based on the conducted research, it can be concluded that in the case of searching for a path on geographical maps, one of the following methods of solving the problem should be chosen. If you want to get the most realistic-looking suboptimal path, we recommend using the Theta* algorithm. When the cost of accessing the terrain is critical, we recommend using the following combination of algorithms:

- using the A* algorithm to get the route;
- delete points that lie on the same line;
- for each pair of a small set of received key points, we use an algorithm to check the presence of a straight path. You can apply this algorithm only to neighboring path segments, splitting them by inserting dummy points.

In General, you need to build a system of algorithms that have similar input and output data, which will allow you to exchange data at separate steps, combining different approaches to solving the problem. In addition, the introduction of a hierarchy will be important: by combining individual terrain areas, you can first lay paths between large areas, then using other algorithms - on separate sections, and then by analogy [1].

References

1. Botea A., Muller M., Schaeffer J. (2004). Near Optimal Hierarchical Path-Finding. *Journal of Game Development*. Vol. 1, issue 1, pp. 7-28.

2. Eric Hansen, Terry Huntsberger and Les Elkins. (2006). Autonomous maritime navigation : developing autonomy skill sets for USVs. Proc. SPIE 6230, 62300U.

3. Nash A. Any-Angle Path Planning. Dis. … Doctor of Philosophy (Computer Science). University of South California. August 2012.

4. Variants of A*, Amit Patel's Home Page. http://theory.stanford.edu/~amitp/GameProgramming/Variations.html (accessed 15.12.2019).

5. Basarab M.A., Domracheva A.B., Kupljakov V.M. (2013). Algoritmy reshenija zadachi bystrogo poiska puti na geograficheskih kartah. *Inzhenernyj zhurnal: nauka i innovacii*. No. 11. URL: http://engjournal.ru/ catalog/it/hidden/1054.html (accessed 20.12.2019). (*in Russian*)

6. Kravchenko M.K., Krivosheev S.V., Mal'cheva R.V. (2018). Realizacija algoritma poiska puti dlja transportnogo sredstva. *Sovremennye tendencii razvitija i perspektivy vnedrenija innovacionnyh tehnologij v mashinostroenii, obrazovanii i jekonomike*. Azov. Vol. 4. No. 1(3), pp. 107-111. (*in Russian*)

7. Krivosheev S.V. (2012). Issledovanie jeffektivnosti parallel'nyh arhitektur vychislitel'nyh sistem dlja rascheta parametrov dvizhenija transportnogo sredstva. *Nauchnye trudy Doneckogo nacional'nogo tehnicheskogo universiteta*. No. 1 (10) - 2 (11). Serija «Problemy modelirovanija i avtomatizacii proektirovanija». Doneck, DonNTU, pp. 207-214. (*in Russian*)

8. Mal'cheva R.V., Krivosheev S.V., Zavadskaja T.V. (2015). Razrabotka simuljatorov transportnyh sredstv s ispol'zovaniem operacionnoj sistemy Android. *Informatika i kibernetika*. No. 2, pp. 76-81. (*in Russian*)

9. Pejsahovich, D.G. Upravlenie interaktivnoj dispetcherizaciej v edinom informacionnom prostranstve posrednicheskogo transportnogo operatora : dissertacija kandidata tehnicheskih nauk : 05.13.10. Mesto zashhity: Penzenskij gosudarstvennyj universitet . Penza, 2014, pp. 151. (*in Russian*)

ОБЗОР АЛГОРИТМОВ ПОИСКА ОПТИМАЛЬНОГО МАРШРУТА ТРАНСПОРТНЫХ СРЕДСТВ

Маховикова Юлия Викторовна, *Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, г. Красноярск, Россия,* ***mahovikova1994@gmail.com***
Мироненко Светлана Николаевна, *Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, г. Красноярск, Россия,* ***sn-mironenko@mail.ru***
Девятков Александр Валерьевич, *Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, г. Красноярск, Россия,* ***dav009@bk.ru***
Шамлицкий Ярослав Иванович, *Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, г. Красноярск, Россия,* ***2538357@mail.ru***

Аннотация
Управление перевозками грузов невозможно без качественного планирования, которое должно быть направлено на эффективное использование транспортных средств. Ключевыми задачами управления перевозками являются маршрутизация грузоперевозок и распределение транспортных средств по маршрутам при условии выполнения плана перевозок в соответствии с выбранным критерием оптимизации. Современное развитие информационно-коммуникационных технологий позволяет существенно повысить качество планирования и контроля исполнения заказов на перевозку грузов наземным транспортом. Оснащение водителей грузовых транспортных средств устройствами спутниковой навигации и терминалами с выходом в Интернет обеспечивает техническую возможность информационного взаимодействия с диспетчерами в режиме реального времени, что определяет новые требования к интеллектуальным системам планирования транспортных ресурсов с учетом человеческого фактора [9]. При создании симуляторов, подразумевающих перемещение различных типов транспортных средств по большим территориям с учетом текущей тактической обстановки, возникают проблемы с выбором алгоритма поиска оптимального пути, так как на его использование накладываются ограничения. Существует большое количество алгоритмов, позволяющих определить маршрут, по которому можно попасть из одной точки в другую. Главная проблема задачи поиска пути заключается в том, что не существует какого-либо универсального алгоритма ее решения. Проведен обзор алгоритмов поиска оптимального пути для транспортных средств (Алгоритм A* и его модификации, в частности Beam search (поиск по лучу); Iterative deepening (итеративное погружение); Dynamic weighting (использование переменных весов); Bidirectional search (двунаправленный поиск); Bandwidth search (Поиск полосы пропускания); Jump Point Search (поиск точки перехода); Theta*). Сделан вывод о целесообразности использования различных алгоритмов на этапах построения предварительного варианта маршрута и его оптимизации.

Ключевые слова: *субоптимальные алгоритмы, алгоритм A*, алгоритм Theta*, планирование маршрута, транспортное средство.*

Литература

1. *Botea A., Muller M., Schaeffer J.* Near Optimal Hierarchical Path-Finding. Journal of Game Development, 2004, vol. 1, issue 1, pp. 7-28.
2. *Eric Hansen, Terry Huntsberger and Les Elkins.* Autonomous maritime navigation : developing autonomy skill sets for USVs / Proc. SPIE 6230, 62300U (2006).
3. *Nash A.* Any-Angle Path Planning. Dis. … Doctor of Philosophy (Computer Science). University of South California. August 2012.
4. Variants of A*, Amit Patel's Home Page. http://theory.stanford.edu/~amitp/GameProgramming/Variations.html (дата обращения 15.12.2019).
5. *Басараб М.А., Домрачева А.Б., Купляков В.М.* Алгоритмы решения задачи быстрого поиска пути на географических картах // Инженерный журнал: наука и инновации, 2013. № 11. URL: http://engjournal.ru/ catalog/it/hidden/1054.html (дата обращения 20.12.2019).
6. *Кравченко М.К., Кривошеев С.В., Мальчева Р.В.* Реализация алгоритма поиска пути для транспортного средства. Современные тенденции развития и перспективы внедрения инновационных технологий в машиностроении, образовании и экономике. Азов, 2018. Т4. № 1 (3). С. 107-111.
7. *Кривошеев С.В.* Исследование эффективности параллельных архитектур вычислительных систем для расчета параметров движения транспортного средства // Научные труды Донецкого национального технического университета. Выпуск № 1 (10) - 2 (11). Серия "Проблемы моделирования и автоматизации проектирования". Донецк, ДонНТУ, 2012. С. 207-214.
8. *Мальчева Р.В., Кривошеев С.В., Завадская Т.В.* Разработка симуляторов транспортных средств с использованием операционной системы Android // Информатика и кибернетика. 2015. № 2. С. 76-81.
9. *Пейсахович Д.Г. Управление* интерактивной диспетчеризацией в едином информационном пространстве посреднического транспортного оператора : диссертация кандидата технических наук : 05.13.10. Место защиты: Пензенский государственный университет. Пенза, 2014. 151 с.

Информация об авторах:
Маховикова Юлия Викторовна, *Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, аспирант, г Красноярск, Россия*
Мироненко Светлана Николаевна, *Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, аспирант, г Красноярск, Россия*
Девятков Александр Валерьевич, *Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, аспирант, г. Красноярск, Россия*
Шамлицкий Ярослав Иванович, *Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, к.т.н., доцент кафедры информационно-управляющих систем, г Красноярск, Россия*