RESEARCH PAPER | Computer Science | Volume : 4 | Issue : 4 | Apr 2014 | ISSN - 2249-555X

A Survey on Routing Algorithms for Efficient and Optimised Railway Scheduling

KEYWORDS: Metro, Routing, Pathfinding, Algorithms

Karan Mitra	Nishikant Mokashi	Prasad Patil
KJ College of Engineering and Management Research, Pune	KJ College of Engineering and Management Research, Pune	KJ College of Engineering and Management Research, Pune

Rohan Navgire	Manali Vashi
KJ College of Engineering and Management Research, Pune	KJ College of Engineering and Management Research, Pune

ABSTRACT *Metro Rail Systems are getting more and more popular in most countries these days. Almost all developing country is working on setting up major metro systems in all their best cities. Advantages of the metro systems are enormous, lead to a substantial increase in the economic as well as financial up gradation of the city and most of all provides a fast, cheap and reliable method of transportation to the residents of the city. With all these advantages the high initial investments and the years of construction are totally covered up. During the inception and establishment of such a system comes a crucial time of deciding the schedules as well as the number of stations and trains that shall be running and their timings. Routing algorithms came into being due to some of these such railway scheduling problems. In this paper, we survey the various formats and types of routing algorithms to decide the most suitable and optimal paths.*

[1] INTRODUCTION

Train companies face many algorithmically challenging questions. The decisions about the number of stations, trains as well as their timings are crucial since the entire functionality of the systems depends upon it. In this paper, we shall consider some of the existing algorithms that have been used before for routing as well as scheduling trains as well as other transportation services to enlist and compare the parameters that shall be important for the urban railways a.k.a. Metros in specific.

Since the advent of the railway systems, many routing algorithms have been used to schedule the trains and decide the paths in the cities where the trains shall run. Along with that, the scheduling is also done which primarily decides the running time of the trains to the dot. Metro systems have a slight advantage in this sector of planning over the traditional rail systems since the metros have pre-decided running systems, i.e. once trains, their routes and their timings are set up, they are rarely changed.[1]

Metro Systems, though also hold some disadvantages over regular train systems in the way that it provides flexibility to the commuter to micro-plan his/her travel, i.e. the person travelling is allowed to choose the path that they wish to take while travelling from a source to a destination. The expectation of a traveller from a routing interface is to provide all the possible routes once these two end points are clearly identified. This is not the case in intercity as well as interstate connecting railway systems in which mostly the trains have very fixed routes and the maximum amount of flexibility that is provided to the traveller is for them to check the stops in the middle and in that way, choose the correct and appropriate train. Thus, in metro systems, the routes have to be dynamically decided by the application which is housing the routing algorithm.[1][2]

[2] PRE-EXISTING SYSTEMS

After establishing the fact that even though metros use a static system of time-tabling as well as planning of trains and stations, the dynamicity of the working of the routing algorithm comes into play when the actual real-time route of a commuter needs to be calculated and shown back on the interface so as to enable the user to make an informed decision. This provision of flexibility adds a level of complexity that is vital in the routing algorithms.[2]

The simplest approach towards routing and/or planning paths would be the path algorithms such as Djkistra's Shortest Path as well as path algorithms may befit the context of planning urban railway routing but it does not practically hold true since in shortest path algorithms the most cutting advantage is that the most important parameter is singular, which is time or distance, both of which are directly proportional. We shall now take a look at some of the popular path algorithms and routing algorithms in detail. We have considered some algorithms which works on nodes and paths which can be analogous to routing of packets between given routers which can be taken as a general case of trains moving between stations since in those cases also there is a source and a sink between the trains and the path logic still holds.

1. Djikstra's shortest path algorithm -

This algorithm provides a simple logical method of checking all possible paths leading from one end point to the other. The algorithm starts calculating shortest paths by stepping from one level of intermediate stops to the other till the time it reaches the other end point, at each level, calculating the time and distance travelled and comparing it with alternative routes that were missed in the earlier steps. It is a general case of the BFS algorithm. It works on the concept of CSF i.e. Cost-so-far value a.k.a. G-value. It assigns costs to every adjacent vertex starting from the initial node and then moves on to the next by accumulating the cost at each stop. A vertex is assigned a new cost if it is re-visited and if the new cost is lower than the previous cost. [5]

2. The travelling salesman problem (TSP) -

In this case a single vehicle has to visit a set of nodes exactly once before returning to its starting position. Such problems implicitly assume that the sum total of demand for services at the nodes is less than the capacity of the vehicle, or alternatively the capacity of the vehicle is not material to the problem. In this case optimality of a route is measured in terms of minimum route length. Practical examples of the TSP include planning the route for a courier who typically has to visit certain homes / houses in an area; other examples include that of developing a repairman's route, or that of a

Volume : 4 | Issue : 4 | Apr 2014 | ISSN - 2249-555X

doctor making house calls. More importantly the TSP often forms a sub-problem of other vehicle routing problems. [8]

3. Distance Vector Algorithm -
Distance Vector Protocol broadcasts its complete routing table periodically. Examples of Distance Vector Protocols are RIP, BGP [Border Gateway Protocol], IGRP, EIGRP [Enhanced IGRP].

A distance-vector routing protocol is one of the two major classes of routing protocols used in packet-switched networks for computer communications, the other major class being the link state protocol. A distance vector routing protocol uses the Bellman-Ford algorithm to calculate paths.[4] [5]

A distance vector routing protocol requires that a router informs its neighbours of topology changes periodically and, in some cases, when a change is detected in the topology of a network.

The three key features for this routing is -
i. Sharing knowledge about the entire Network:
Each router sends all of its collected knowledge about the network to its neighbours.

ii. Sharing only with neighbours:
Each router sends its collected knowledge about the network to its neighbour routers which directly connected. It sends whatever it has knowledge about the network through all of its ports.

iii.Sharing at regular intervals:
Each router periodically shares its knowledge about the entire network with its neighbours.

Sharing Information:
A router can share its knowledge about network to its neighbours.

The knowledge may be collected by itself or otherwise shared from other routers.
Routing Table:
Distance vector routing information may be, Network ID, cost and NextHop. These three essentials need to form a Distance vector's routing table.

4. Link State Algorithm -
A Link-state routing is a concept used in routing of packet-switched networks in computer communications. Link-state routing works by having the routers tell every router on the network about its closest neighbours. The entire routing table is not distributed from any router, only the part of the table containing its neighbours.

The basic concept of link-state routing is that every node constructs a map of the connectivity of the network, in the form of graph. Using that map of connectivity graph, each node independently calculates the best next hop from it for every possible destination in the network. The collection of best next hops forms the Routing Table for the node.

Contrast with Distance Vector:
Distance vector, which work by having each node share its routing table with its neighbours. But, in link state protocol, the only information passed between the nodes is information used to construct the connectivity maps.

Optimized link state routing protocol is its extended version which is used with wireless mesh networks.

The key features of this routing include all the nodes knowing the state of the other nodes which is usually done by one of the nodes via a process called flooding in which one router sends all of its data to every other router, like a broadcast. Flooding is also done every time the data changes or is modified.

5. K shortest path routing -
There are many cases in which it is vital to have more than one path between two nodes in a given network. Sometimes different paths need to be found due to some additional constraints. One simple method is to use any basic path finding algorithm and then extend it to include other alternative paths. The K Shortest Path routing algorithm is a generalisation of the shortest path problem. Its primary task, like the other algorithms is to find the shortest path, but, in addition to that, it also finds a user-specified K number of shortest paths in the order of increasing cost. The problem can be restricted to have the K shortest path without loops (loopless K shortest path) or with loop. The running time complexity is O(Kn(m + n log n)). m represents the number of edges and n is the number of vertices. [5]

6. A * path finding algorithm -
A* is a computer path finding algorithm which is a combination of two different types of algorithms, namely, incremental and heuristic search algorithms. Incremental searches work in increasing steps in which the output of the previous step is used as the input to the next step. Heuristic algorithms are those that learn from previous iterations and attempt to identify paths that should not be taken in the proceeding steps. This ensures that the obvious paths that are definitely incorrect are discarded from consideration early on and computation time is not wasted on them later on. The domain of usage of such algorithms are those in which the entire scenario may not be known before hand. Heuristic searches have been studied since the late 1960s. There are certain search problems, known as dynamic path planning problems, in which paths are to be found repeatedly since the graph, costs and vertices may change over time. The time complexity of A* depends on the heuristic. In the worst case, the number of nodes expanded is exponential in the length of the solution (the shortest path), but it is polynomial when the search space is a tree, there is a single goal state, and the heuristic function h meets the following condition:

$|h(x) - h^*(x)| = O(\log h^*(x))$
where h* is the optimal heuristic, the exact cost to get from x to the goal. In other words, the error of h will not grow faster than the logarithm of the "perfect heuristic" h* that returns the true distance from x to the goal. [4][5]

7. Iterative Deepening A* Algorithm -
The Iterative Deepening A* (IDA) was presented in 1985. It is a memory optimal algorithm and an acceptable alternative to the A* algorithm. It is quite similar to the Depth First Search (DFS), and the algorithm itself can simply be represented as a series of DFSs from the start node and with an increasing depth till a particular threshold value. These searched are carried out recursively which ensures that every search enhances the effectiveness of each search. The algorithm is limited by using an initial threshold value and all the nodes are searched by always checking the threshold value before the search is applied. This ensures that no path will be found more than a specified minimum heuristically predicted cost of the search. It is an effective algorithm since it is easy to implement and to understand since it does not involve any major complex steps, just simple depth first searches. Disadvantages include the fact that it is effective only in a limited scope of problems and thus, is not a more generalised algorithm. [4]

8. Simplified Memory-Bounded Algorithm -
The SMA algorithm is presented as an improvement to the Memory bounded search algorithm. In the MA algorithm, a fixed buffer is used to store favourable nodes which ensures that node re-exploration is not required. The drawback of this method, even though it saves up on a lot of search time, is that when the algorithm runs out of buffer space, the least favourable node is taken away from the buffer, which some-

Volume : 4 | Issue : 4 | Apr 2014 | ISSN - 2249-555X

times leads to memory losses. To reduce the degree of re-exploration, a node estimated backup system is implemented to try and counted this loss of information. The SMA* backup system for nodes propagates the estimated cost of the successor to the successor's parent and recursively till the root. If a successor's parent has a favourable cost backup value, then it is kept in the list rather than taken away which facilitates node re-exploration. With this advantage in mind, there appears to be an obvious drawback in this algorithm in the measure of complexity since now every backup list item is stored as a binary tree of binary trees. Thus, time and space complexity both increase since even the time to go through such a structure will be more as compared to a simple list as in the case of A*. [4]

9. Fringe Search -
Similar to the similar memory-bounded algorithm is cancelling the node re-exploration limitations of the of MA, the fringe search is made as an enhancement to the IDA*. The basic difference between the two that defines the working of the fringe search lies in the working of the IDA. In IDA, the recursive DFS iterations require that the subsequent DFS starts from the root or the initial node which brings about the drawback of certain nodes always being re-explored. Also, the number of nodes re-explored constantly keeps increasing as the iterations advance.

In order to overcome this limitation, the fringe search allows the storage of the end of each recursive cycle such that the next cycle will start from this previously stored stage. This ensures that almost no nodes are re-explored. Fringe search works in a very simple manner, keeping track of two lists, namely the "now" and the "later" lists. As the name suggests, the former list keeps tracks of the nodes that are to be explored during the current search and the latter list of the nodes that will be explored in the subsequent searches. Both the lists are updated during each iteration, thereby giving way to a threshold limited faster way of searching using DFSs. [4]

10. Lifelong Planning A* Algorithm -
D* lite algorithm, which can be considered as the basis of most dynamic environment scoped search algorithms, is a modified, backwards version of LPA*.

LPA* is an incremental version of the A* algorithm that reuses information from previous searches across search iterations, also known as the heuristic method of algorithm operation. Heuristic algorithms are used when we require the algorithms to function repeatedly changing environment since such algorithms keep track of the favourable and non-favourable nodes that are encountered in the first few workings of the algorithm. Thus, in the subsequent usages, unfavourable nodes are avoided and favourable paths are considered.

The backwards concept of the LPA* algorithm can be understood by considering the local consistencies of a node. If a node is locally consistent when its G value if equal to its RHS value. This means that the cost required to reach that node since the starting of the iteration should be the minimum cost required to reach that node from the start node. Thus, it comes to say that if all the nodes in a working of algorithm are locally consistent, then one can simply find any path by working these G values backwards from the destination node till we reach the source node and adding up all the G values on the way overlapping the common paths thereby giving us the shortest cost path. This is the overall working of the Lifelong Planning Algorithm. [4][5]

[3] ROUTING MEASUREMENT PARAMETERS -
Parameters that may be used for assessing the efficiency of a routing algorithm may be many in number but they are overlapping as well as contradictory in a lot of situations. Thus, in order to correctly analyse an algorithm, we need to ensure the completion of two tasks -

Based upon the study of the environment and implementation, assess the parameters which shall be used in the required situation. Also, check the inter-relationships between the parameters, since as mentioned before, there are parameters which are contradictory to each other. Only those parameters which result from these steps should be considered should be taken in the analysis, as it will give a clear idea about the usage of the algorithm and its overall effectiveness can be clearly considered and also compared to other similar algorithms.

By referring paper [5] , the following parameters may be considered for assessing a routing algorithm:

- Time Complexity Time complexity takes into account the number of repetitions, time taken to execute certain complex statements etc. Time complexity can be taken out in different manners such as lower bounds, upper bounds also in order to analyse the minimum or maximum time taken by an algorithm to execute.

- Space Complexity - Space Complexity of an algorithm is total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input. It takes into account the data structures, their complexities and storage mechanisms required for accessing and manipulating data items. It is important in almost all situations since an excessive space complexity puts load on the system as well as the processor, thereby increasing cost and maintenance.

- Re-exploration of Nodes - As seen in the above comparative analysis of the nodes, specific to path finding algorithms, algorithms require for some nodes to be processed again in order to check for a different path or a lower cost route. This re-exploration requires that a memory space is also dedicated to it, like a buffer, so that a track of the nodes can be kept in order to determine which ones are favourable and which are not.

- Speed of path recalculation In order to increase the effectiveness of an algorithms, they are designed in a manner in which they can handle this recalculation faster. The measure of this increase in speed for re-exploration is maintained by this parameter.

- Type of Movement through graph - Algorithms may work very differently. Their types of movements are checked to compare algorithms which are working a basically different concepts. Some may compare neighbouring nodes and try to find a path till then, others work on edge basis or angle basis.

- Ease of Implementation - Implementation simply refers to converting the algorithm into a application form using some programming language. Implementation also refers to the ability for the algorithm to adapt to most situations and environments. There are some algorithms which are easily implementable to some environments where unsuitable for others.

- Scalability - Specific to path finding algorithms, its generalisation can be determined by observing its scalability. Scalability refers to the ability of an algorithm to bear with changes to the nodes. If we were to increase or decrease the nodes in a given scenario, the adaptive mechanisms in an algorithm come into play. Increasing of nodes can be referred to as up-scaling and decreasing of nodes is known as down-scaling. This scaling is considered up to a certain threshold limit, i.e. we do not expect any algorithm to ever be infinitely scalable. Only practically applicable values of scalability are considered. If an algorithm can morph into a form which can manage to handle these changes, it is referred to as a scalable algorithm, otherwise not.
- Failure of Nodes - Failure of nodes is a more specialised

Volume : 4 | Issue : 4 | Apr 2014 | ISSN - 2249-555X

concept of scalability. If one or more nodes tend to not be part of the equation anymore, some algorithms tend to fail at that point rather than adjust to the changes. The difference between down-scaling and failure of nodes is that in down-scaling the nodes are removed from consideration itself, but in failure the nodes are there in the system but cannot be used as part of the solution

[4] COMPARITIVE ANALYSIS -

The choice of an algorithm depends a lot on the architecture and the circumstances on which it is applied. Djkistra's algorithm is the easiest to implement and to apply since it requires minimal coding and is applicable to most situations as it is a very basic application algorithm. However, it does not take into account failure of one of the nodes or paths and is impervious to scalability changes. The travelling salesperson problem is a very specific algorithm which has a lot of predefined conditions such as the cyclic formation of the travel as well as the hardbound condition that all the nodes have to be visited. This limits its application scope by a significant amount. Moreover, the time complexity of this algorithm is of the kind $O(n2^n)$ which means that if in any case, more stations are added then the time taken will increase by a huge amount.

Distance vector–based routing protocols are simple router advertisement processes that are easy to understand. Between the size of the routing table and the high overhead, distance vector–based routing protocols do not scale well to large and very large internetworks. Link state protocols on the other hand, require more memory and processor power than distance vector protocols.

Thus, we can conclude that when the number of stations and the paths are pre-decided and scalability is not an issue, Djikstra's or TSP can be used to a certain level. When fail-safe systems and a more dynamic level of planning is required, distance vector and link state routing is used. Within them, link state routing is the more efficient algorithm since its only limitation is the fact that it requires more resources in the form of memory and processing power which can be easily managed and does not prove to be a hindrance in the overall efficiency of the algorithm.[2][4][5]

Considering properties applicable to adaptive meshes, re-planning, re-exploration as well as cost changes, D* algorithm stands out as the simplest and most effective algorithm.

Parameter/ Algorithm	DJKISTRA	TSP	DISTANCE VECTOR	LINK STATE
Implementation	Easy	Easy	Relatively tougher	Relatively tougher
Scalability	Not scalable	Very limited scaling	Easily scalable	Easily scalable
Failure of Nodes	Not considered	Causes overall failure	Re-routing is required	Re-routing is required
Information of Topology	Not known to all nodes	Known to traveller but not to each node	Not known to all nodes	Known to all nodes before routing commences
Time Complexity	$O(n^2)$	$O(n2^n)$	O(d=Network diameter)	

TABLE 1 Comparison between simple path finding methods

Method	Time Complexity	Space Complexity	Portion of path recalculated	Speed of path recalculation	Type of Movement through graph
Djikstra [5]	$O((E+V)\log V)$	$O(V^2)$	Entire path	Slow	Edge-based
A*[6]	$O((E+V)\log V)$	$O(V^2)$	Entire path	Slow	Edge-based
LPA*[7]	$O((E+V)\log V)$	$O(V^2)$	Current node to end	Moderate	Edge-based
D* Lite[8]	$O((E+V)\log V)$	$O(V^2)$	End to current node	Fast	Edge-based
Theta*[9]	$O(V^2)$	O (V)	Entire path	Slow	Any-angle
Field D*[11]	O(V log V)	O (V)	End to current node	Fast	Any-angle
K-shortest path routing	$O(Kn(m+n\log n))$	O(Vk) k=No of alternative paths	Entire path	Moderate	Edge-based

TABLE 2 Comparisons between path finding algorithms [5]

ALGORITHM	ADVANTAGES	DISADVANTAGES	LIMITATIONS (if any)
Djikstra's	• Simple and Easy to Implement • Complete and Optimal • Moderate Time & Space Complexity	▪ Excess node re-exploration ▪ Slow recalculation ▪ Inability to resume after failure of nodes	Not scalable. Complexities increase drastically upon increasing nodes
A*	• Heuristic Algorithm • Moderate Time & Space Complexity • Iterative and Incremental	▪ Excess node re-exploration ▪ Slow recalculation ▪ Inability to resume after failure of nodes	
D* Lite	• Simpler and easier than A* or D* • Backwards planning • Adaptable to Dynamic environments	▪ Limited to situations ▪ Moderate speed	
LPA*	• Iterative and Incremental • Quick path recalculation • Previous as well as look-ahead costs are considered	▪ Repeated steps ▪ Recalculation and Re-exploration	It uses DFSs to search, so scope of DFS is carried forward
SMA	▪ Decreased node re-exploration ▪ Faster working than all predecessors ▪ Efficient memory management	▪ Excessive data structure keeping ▪ Increased space complexity	
Fringe Search	• High level of node backup • Very less re-exploration • Relatively faster than most searches		

TABLE 3 : Advantages, Disadvantages and Limitations of Path Finding and Routing Algorithms[4][5][6][8][9]

[5] CATEGORISATION OF ALGORITHMS -

Using the measurement parameters as the base, we can categorise the path finding algorithms into certain types. These types help us choose the correct algorithm for the correct type of usage and implementation.

The following categories can be considered -

- Iterative / Non Iterative Algorithms - These are algorithms which run on loops, iterations and propagate information through every iteration which helps in deciding favourable and unfavourable paths which eventually leads to optimal solutions. Eg. Fringe Search, Iterative Deepening Algorithm etc.

- Incremental / Non Incremental Algorithms Incremental algorithms are those which accumulate information with each run or iteration of the algorithm. Accumulation of information leads to effective comparison between all the data leading to optimum solutions. Eg. LPA* etc.

Volume : 4 | Issue : 4 | Apr 2014 | ISSN - 2249-555X

- Heuristic / Non Heuristic Algorithms - Heuristic algorithms are used when we require the algorithms to function repeatedly changing environment since such algorithms keep track of the favourable and non-favourable nodes that are encountered in the first few workings of the algorithm. Thus, in the subsequent usages, unfavourable nodes are avoided and favourable paths are considered. Thus, in other words, it can be said that heuristic algorithms learn from their previous iterations and improve themselves in successive iterations. Eg. A* etc.

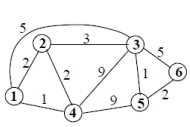
Heuristic algorithms can further be of 3 types:
- Manhattan
- Euclidean
- Chebyshev

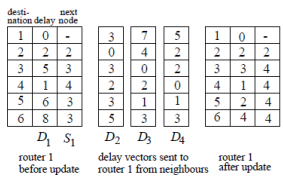
- Discrete Search / Continuous Search Algorithms - Search algorithms can work in two kinds of environments, one in which the entire path and scenario is known to the algorithm and an optimal path has to be found given these boundary conditions as input. For such environments, discrete search algorithms are applied since they work on a complete set of inputs only. On the other hand, continuous search algorithms are given a set of initial conditions and thresholds and all the other inputs are entered or are encountered (usually by sensors) and the algorithm works with this new set. This calls for a high level of adaptability and readjustment mechanisms that need to be available in the algorithm since they will always be in play and will control the working and optimality of the algorithm. Examples of discrete search algorithms include A*, D* etc. whereas the examples of continuous search algorithms are LPA*, D* Lite etc.

- Edge - Based / Any - Angle Based Algorithm - Basic algorithms such as Djikstra's or A* algorithms all fall under the edge based algorithm category since they work on edges joining nodes. Paths are found all on the basis of addition of costs of optimal edges between the nodes in the chosen optimal path. Any Angle path planning algorithms are those which consider the angle between the start and the end node in a 2 dimensional block environment and work towards keeping that angle optimal so that minimum number of nodes are used thereby decreasing the cost of the path searching and planning. Djikstra's and A* fall under edge-base category and Theta* and Field D* fall under Any-Angle path planning type.

[6] RESULTS -

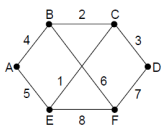
Distance Vector Routing

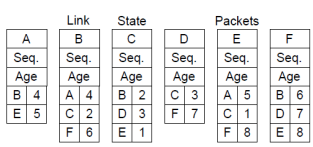




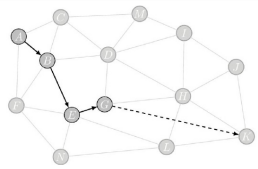
[9] (Reference)

Link State Routing

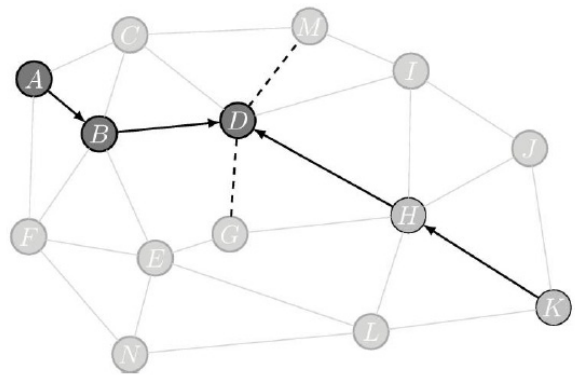




[9] (Reference)

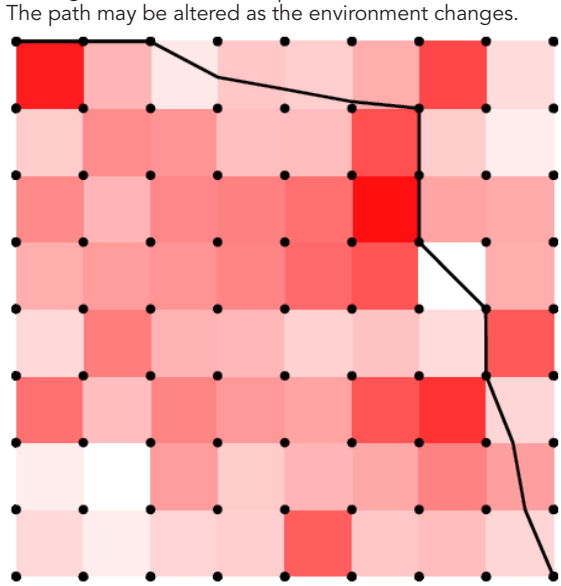


An intermediate apth generated by A*. The dotted line represents the heuristic estimating the distance of the goal; in this case, the Euclidian distance heuristic.



[5] (Reference)

An example of D* lite iterative search. Having found a path KHDBA, the path has been traversed to D, where the neighbouring vertices and D are processed for new information. The path may be altered as the environment changes.



[5] (Reference)

An example of a path created by Field D*, displaying any-angle movement through individual grid blocks

[7] CONCLUSION -

Djkistra's is not suitable for more number of nodes. A* suffers from slow recalculation whereas in LPA* greater recalculation is required. There is need for an algorithm with works for more numbers of nodes effectively with less time complexity and space complexities and provides accurate results.

REFERENCE [1] IEEE 2012 Android Suburban Railway Ticketing with GPS as Ticket Checker (ICACCCT). | [2] Emerging Trend in Using Smartphone Technology for Transportation Research | [3] Android App Development with Java Essential Training –www.lynda.com | [4] Video Game Pathfinding and Improvements to Discrete Search on Grid-Based Maps by Bobby Anguelov | [5] Pathfinding Algorithms, A Literature Survey by Daanyaal du Toit | [6] Scheduling Freight Trains Traveling on Complex Networks by SHI MU and MAGED DESSOUKY | [7] Train Routing Algorithms: Concepts, Design Choices, and Practical by Luzi Andereggy Stephan Eidenbenz | [8] A Survey on Shortest Path Routing Algorithms for Public Transport Travel by S.Meena Kumari & Dr.N.Geethanjali | [9] Computer Networks, Electronics and Computer Science Department, University of Southampton