IJASRM

International Journal of Advanced Scientific Research and Management, Volume 4 Issue 1, Jan 2019

www.ijasrm.com

ISSN 2455-6378

Fastest Shortest Path Finding Algorithms for Large Road Network

Aijaz Magray[1] and Chitaranjan Sharma[2]

[1,2]Department of Mathematics, Devi-AhiliyaVishwavidyalaya, Indore, M.P, India

Abstract

Shortest Path problems are amidst the most premeditated network flow optimisation glitches, with exciting applications in a range of fields. One such solicitation is in the arena of GPS routing systems. These systems important to speedily resolve enormous shortest path problems but stand classically inserted in devices with restricted memory and external storage. Conservative procedures for solving shortest paths within enormous networks cannot be used as they are either too slow or need enormous quantities of storage. In this paper we have reduce the runtime of the conventional techniques by exploiting physical structure of the road network and we are also using network pre-processing techniques. Our algorithms may guarantee optimal results but can offer major savings in terms of memory necessities and processing speed. Our effort uses heuristic estimates to sure the search and directs it towards the destination. We also associate a radius with each node that offers a measure of significance for roads in network. The further we acquire from either the origin or destination the extra selective we become about the roads we travel on, giving significance to roads with bigger significance (roads with larger radii).

By using these techniques we are able to reduce the runtime performance equated to conventional techniques while still maintaining an acceptable level of accuracy.

Keywords: *Dijkstra's Algorithm, shortest path, A* algorithm.*

1. Introduction

Due to nature of routing applications, we need flexible and effective shortest path techniques, both from a processing time point of view and also in terms of the memory requirements. Unluckily previous research does not offer a clear direction for picking an algorithm when one faces the problem of the computing shortest paths on real road networks [1,2].Previous research in testing different shortest path algorithms advises that Dijkstra's implementation with double buckets is the ultimate algorithm for networks with positive arc lengths. Still like most popular papers on Shortest Path algorithms, they have concentrated their motivation on algorithms that security optimality and have functioned on tuning data structures used in implementing these algorithms.

Meanwhile no "best" algorithm presently exists for every kind of transportation problem, research in this field has recently moved to the design and implementation of "heuristic" shortest path procedures, which are able to capture the individualities of the problem under consideration and mend the run time performance of a search, but at the cost of not guaranteeing optimality. As it is impossible to cover all search implementations, we use Dijkstra's algorithm as a building block to generate an effectual search algorithm that implements an artificial intelligence approach to the routing problem that may not guarantee optimal results but gives significant savings in terms of memory requirements and processing speed.

To examination these algorithms we used portions of the London road network (courtesy of Talon Technologies). The network troubled covers a physical area of about 22,500 km2, has about 140,000 nodes and 298,000 directed arcs. We absorbed on finding paths that minimalized the probable travel time in this network.

1.1 (ITS) Intelligent Transport Systems

To fully gain the merits of a search technique it is significant to recognize the marketable environment in which these methods are implemented. Many route finding systems are presently in development worldwide and the majority form part of much superior systems to succeed and operate the road network more competently. These management infrastructures are acknowledged as Intelligent Transport Systems (ITS) and differ in complexity and size. These systems reduction into two main categories centralised and decentralised systems. Centralised systems are connected to an information centre

which assembles and processes traffic and network information. Usually a driver requests a specific route from on-board electronics. The route is then relayed to a central location that transmits out all the processing of the route[3].

Decentralised systems on the other hand offer information to the driver which is computed onboard consuming local information sources. Normally such systems comprise road network information on the optical storage devices and electronics to feed a GPS. The Auckland based company Talon Technologies is developing a decentralised routing system; this effort forms part of their research effort.

2. Network Definitions

Before continuing let us announce some notation and legally define the shortest path problem. A network is a graph G = (N, A) containing of a unique indexed set of nodes N with n = |N| and a spanning set of directed arcs A with m = |A|. Each arc a is represented as an ordered pair of nodes, in the form "from node i to j", denoted by $a = (i, j)$each arc (i, j) has an associated numerical value Lij,which represents the distance, time or cost incurred by traversing the arc. Each node i has a set of successors$S(i)$ (i.e. the set of all nodes j: $(i, j \in A)$and predecessors P(i) (i.e. set of all nodes j: $(j, i) \in A$

3. Methodology

One possible approach to solving shortest path problems would be to pre-calculate and store the shortest path from each node to every possible other node, which would permit us to answer a shortest path enquiry in constant time. Unfortunately the essential storage size and computation time develops with the square of the no of nodes. With realistic road networks in awareness this processing would yield years if not decades and be almost impossible to store. Hence to stun this problem we need real time search techniques.

4. Search Algorithms

From earlier studies [1,2,4] we know that the application of labelling algorithms are fastest for one-to-one searches. Two features are particularly significant to the shortest path algorithms conversed in this:

1. The strategies used to choice the next node to be visited during a examine, and

2. The data structures consumed to maintain the set of earlier visited nodes.

A numeral of data structures can be used to operate the set of nodes in order to backing search strategies. These data structures contain arrays, singly and doubly associated lists, heaps, buckets, queues and stacks, detailed definitions and operations associated to these data structures are typical knowledge and are well documented. Previous research has focussed mainly on the issue of data structures, which can be operated and bounded to form clever methods in creating priority queues for choosing nodes to be scanned. A good example of this is the Dijkstra's execution with double buckets [1].

In a labelling algorithm, the no of visited nodes during the search is a good signal of the size of the search space. This means that a search approach which visits rarer nodes during a search is normally more efficient in terms of processing speed. The number of nodes visited depending on the depth d (i.e. the no of arcs on the optimal path) of the endpoint from the origin, and the branching factor b. For a 'finest first search' the number of nodes travelled during a search is of the order O(b^d) [3]. This exponential growth in the number of travelled nodes is identified as "combinatorial explosion" and is the main difficulty in computing shortest paths in large networks. ("even though Dijkstra's algorithm is polynomial in number of nodes n in the graph, and this bound is no restraint on how the number of nodes visited differs with d"). For general examination this exponential growth with depth marks many problems impossible on current hardware, as memory is soon pooped and a solution may take an unreasoning time to calculate. These effects can be lessened by using artificial intelligence (heuristic type) techniques which will be argued later. However let us first describe and implement Dijkstra's labelling algorithm.

4.1 Dijkstra's Naive Implementation

Dijkstra's labelling technique is a central process in shortest path algorithms. The output of the labelling technique is an out-tree from a source node s, to a set of nodes L. An out-tree is a tree creating from the source node to the other nodes to which shortest distance from source node is known. This out-tree is assembled iteratively, and shortest path from s to any destination node t in the tree is attained upon termination of the technique.

Three pieces of evidence are required for each node i in the labelling technique while building the shortest path tree:

• The distance label, $d(i)$

• The parent-node/predecessor $d(i)$

• The set of permanently labelled nodes L.

The distance label $d(i)$stores an upper bound on shortest path distance from s to i, while $P(i)$records the node that instantly precedes node i in the out-tree. If the node has not been yet added to the out-tree, it is considered 'unreached'. Normally the distance label of an unreached node is set to infinity. When we recognize that the shortest path from node s to node i, is also absolute shortest path, then node i is called permanent labelled. When additional improvement is expected to be made on the distance from the origin to the node i, then node i is consider only temporarily labelled. It monitors that $d(i)$is an upper bound on the shortest path distance to the node i if node i is temporarily labelled, and $d(i)$ represent the finishing optimal shortest path distance to node i if node is permanently labelled [1,2]. By iteratively addition a temporarily labelled node with the lowest distance label $d(i)$ to the set of perpetually labelled nodes L, Dijkstra's algorithm assurances optimality.

One benefit with Dijkstra's labelling algorithm is the algorithm can be terminated when the destination node gets permanently labelled. Most other algorithms assurance optimal shortest paths only upon the termination when the whole shortest path tree has been explored.

4.2. Symmetrical Dijkstra's Algorithm

Pohl modified Dijkstra's shortest path algorithm to decline the size of the search space. Pohl's algorithm was the leading to use a bi-directional search technique[1]. This algorithm contains of a forward search from a source node to the destination node and the backwards search from destination node to the source node. This was done in an effort to diminish the search complexity to O($b^{d/2}$) associated to O(b^d) as with Dijkstra's algorithm. This search technique assumes that two searches grow proportionally and will meet in certain middle area. Sometimes this may not be the case, and as the worst case scenario this may instead become two O(b^d) searches.

The Symmetrical\Bi-directional Dijkstra's algorithm by Pohl develops two search trees,

1. From the origin, giving a tree spanning a set of nodes L_F for which the minimum distance and time from the origin is known,
2. From the destination that gives a tree spanning a set of nodes $L_B$for which the minimum distance and time to the destination is known.

   We iteratively add one node to one or the otherL_F or L_B until there exists an arc crossing from $L_F$or L_B. Like Dijkstra's algorithm Pohl's bi-directional search selects the node with the smallest cost label to the label permanently. By choosing the new permanently labelled node from the either forward or backward phases we retain the Dijkstra's criterion required for optimality.

4.3 A* Search

So far we have examined search methods that can be comprehensive for any network (as long as it doesn't contain negative length cycles). However the physical nature of the real road networks inspires investigation into the possible practice of heuristic solutions that exploit the near-Euclidean network structure to decrease solution times while confidently obtaining near optimal paths. For maximum of these heuristics the goal is to bias a more focused search towards the destination. As we shall see, incorporating heuristic information into a search can affectedly reduce solution times.

When the essential network is Euclidean or approximately Euclidean as is the case of the road networks, then it is possible to recover the average case run time of the Dijkstra's and Symmetrical Dijkstra's algorithms. This is generally at the expense of the optimality; solutions are now not assured to be the best. Usually when solving such problems on networks the inherent geometric information is unnoticed by algorithms that are directly based or the variations on the Dijkstra's labelling algorithm.

The A* algorithm by Nilsson and Hart [2] formalised the idea of integrating a heuristic into a search technique. Instead of selecting the next node to the label permanently as that with the smallest cost (as measured from the initial node), the choice of the node is based on the cost from the initial node plus an estimation of proximity to the destination (a heuristic estimate) [4]. To shape a shortest path from the initial s to the destination t, we use the original distance from the s accumulated along edges (as in Dijkstra's shortest path algorithm) plus an estimate of distance to t. Thus we use the global information about our network to monitor the search for shortest path from s to t. This algorithm places more position on paths leading towards t than paths moving away from the t.

In essence with the A* algorithm associates two portions of information:

1. The current knowledge obtainable about the upper bounds (assumed by the distance labels$d(i)$),

2. An estimate of distance from leaf node of the search tree to destination.

There are several ways to estimation the lower bound from the leaf node in the search tree to destination node. These estimations are carried out

by so entitled "evaluation" functions [3]. The nearer this estimate is to a tight lower bound on the distance to destination, the better quality of the A* Search. Hence the merits of an A* search depends extremely on the evaluation function$h(i,j)$. There are two main calculation functions used in A* search. A true lower bound among two points is the length of a straight line among those two points (i.e. Euclidean distance):

$$h_E(i,t) = \sqrt{(x(i)-x(t))^2 + (y(i)-y(t))^2}$$

Where$x(i), y(i)$, and$x(t), y(t)$ are the coordinates for the node i and destination node t correspondingly. The other generally used estimation function is the Manhattan distanceh_M. In this situation the estimated lower bound distance is the addition of distance in the x and$y$ coordinates

$$h_M(i,t) = |x(i)-x(t)| + |y(i)-y(t)|$$

The Manhattan distance is not correct lower bound among two points and hence will normally return non-optimal results.

By using time as a measure of cost, the network converts near-Euclidean. This is because of fluctuating speeds of roads in network. Roads of similar lengths might have dissimilar times connected with using those roads. If the network is not severely Euclidean but near-Euclidean then our choice criteria for next node to the label permanently will not produce optimal results.

By using A* search, the shortest path tree would now develop towards t (distinct Dijkstra's algorithm where the tree develops approximately radially). As before, the search for shortest path is finished as soon as t is added to shortest path tree. Earlier we argued the problem of the combinatorial explosion with a blind search time complexity in order of $O(b^d)$ With A* search this is condensed to Ob_e^d. Where$b_e$is the effective branching issue. The A* search reduces search space by decreasing the number of node expansions. Although A* is still susceptible to a problem of combinatorial explosion, it decreases the effect by decreasing the size of the base in the complexity term.

4.4 Weighted A* Search

By selecting an appropriate multiplicative factor we can rise the contribution of the estimated component in calculating the label of a vertex (i.e. increase the contribution of evaluation function) [4]. From an instinctive standpoint this resembles to further biasing the onward search towards destination and the backward search towards origin. The heuristic is parameterised by multiplicative factor called the "overdo" parameter recycled to weight the evaluation function. This alteration will generally not produce optimal paths, but we would believe it to further decrease the search space. The objective is to find an "optimal" multiplicative or burn factor for which the running time is meaningfully improved while the solution superiority is still acceptable. Thus there will be an experimental time and performance trade-off as a function of the burn parameter.

4.5 Radius Search

To eradicate or to minimise the effects of combinatorial explosion we need to accept a search technique alike to the way human's method navigation problems. So far we haven't implemented any intelligence within the search which can filter out roads that are fewer likely to be travelled on. This type of intelligence involves some form of historical information about the network. Since the road network doesn't change very often it is possible to compute auxiliary information in a pre-processing step[3,5]. Perhaps the most evident way to classify the roads in the network is to recognise the class of each road (i.e. local roads, highways, motorways etc), and then to exploit these classes in a search. This is similar to the way human's method routing problems and is identified as Hierarchical Search.

Hierarchical methods offer the prospect of significantly reducing the size of any search by simplify the search through the series of simplified levels, where each of these levels is an concept of the previous level. These abstractions decrease the overall size of the search space that the algorithm addresses and thus the complexity of any search is compact. For route finding, hierarchical levels are created in which higher speed roads are positioned higher up in the hierarchy. However by announcing these arbitrary hierarchies the path optimization is often lost [3].

The hierarchical algorithm practices a discrete no of hierarchy levels. A Radius search is a hierarchical search with a non-stop range of hierarchy levels. A Radius search takes benefit of the fact that fastest path among two junctions is more probable to use a highway than the local road, particularly if the two junctions are far apart. In this technique each node i has an associated radius $r(i)$.Before we reflect how $r(i)$is calculated, we first inspect how radii can be used to limit a search.

When viewing for a shortest path from $s$to t,a node i is measured as a possible node to contain in the search only if t or $s$lies inside a circle of radius $r(i)$placed at node i. If both distances are larger than the node radius, the node is simply unnoticed [5]. For the any given origin\destination node, we

can instantly simplify the network by eliminating all the nodes (and related arcs) whose radii don't encircle the origin \destination nodes. The radius search is not the search algorithm by himself, but an independent mechanism of decreasing search complexity. Hence the radius idea can be used in the conjunction with any search algorithm.

The efficiency of the Radius search depending on the way we calculate the radii. The optimal radius for a node i is the least radius $r(i)$ for which the radius placed at node i encircles either the origin\destination node for all optimum paths that include node i. If the radii are designed as an extreme over all such shortest paths, then it is certain that the radius search algorithm is exact (i.e. guaranteed optimality). The radii are also minimum since with any smaller radius at minimum one optimal shortest path will not be create.

The optimal radius for the node$i$ can be defined as

Let $R = (R_{[1],}R_{[2],}\ldots\ldots\ldots\ldots\ldots R_{[P],}$be an optimal path (sequence of the nodes) from an origin node $R_{[1],}$ to the destination node $R_{[P],}$

Let $\Re = \{R_{1,}R_2\ ,\ldots\ldots\ldots\ldots\ldots,R_{|\Re|}\}$ be set of all the optimal path on G.

Let $\Re(i)$be set of all optimal paths that use node i

$$\Re(i) = \{R\ \mathcal{E}\Re\colon \exists h\mathcal{E}\{1,2,\ldots\ldots\ldots|R|-1\}\colon R_{[h]} = i\}$$

$$r(i) = max_{R\mathcal{E}\Re(i)}\{\min\{h_E(i,R_{[1]}), h_E(i,R_{[|R|]}$$

One probable difficulty is that calculation of the radii by investigative all paths over a specific node takes much else long since every probable shortest path in the network and has to be calculated at minimum once. Instead we are implemented a heuristic method to calculate these radii [5]. In the very first phase of this heuristic method we divide the network into the overlapping grids of around 2000 nodes and initialise all the node radii to be 0. We then choose a random starting node s from all the possible nodes N and a random final destination node t within the similar grid as s. we use the Symmetric Dijkstra's algorithm we solve for shortest path R from s to t. We then modernize the radii of nodes in path R using the below mentioned algorithm:

Begin:

updateNodeRadii(R)

{

For all $n\ \mathcal{E}R$ do

$$r(n) = \max\left(r(n), \min\left(h_{\mathcal{E}}(n,R_{[1]}), h_E(n,R_{[|R|]})\right)\right)$$

Next n

}

End

We carry on this process for selecting random starting and destination nodes and modernizing or updating the radii of nodes in the shortest path as many times as possible.

If we don't generate sufficient random paths in first phase then the radii of few nodes will never get updated and hence will be still be 0. However if the node is a 'closed node' (i.e. the node is the only used in the shortest path if it is either origin or the destination of that shortest path) then it will be never a part of a shortest path except we start (finish) at that node. Hence the radii of the closed nodes will at all times be 0. In second phase of this improved algorithm we go through all the nodes in network and survey their radii. If the node is not closed and has a 0 radius, then we conduct the shortest path searches in vicinity of the node that generates a realistic lower bound on its radius. We do this in second phase by generating a sub graph of the 200 of the neighbouring nodes and associated arcs G_{SUB2} to node with 0 radius and resolve all-to-all the shortest paths on G_{SUB2}this should force some the shortest paths R through this node and provide it an improved radius lower bound than 0.

So far in the above first two phases we have calculated the shortest paths within grids. Hence radii are no higher than the grids they are created in. As a result, after first two phases we have a fairly worthy coverage of the local radii only (i.e. those radii only limit a search for the shortest paths within the grids). If we were to usage these radii to the restrict a search over large distance (i.e. over numerous grids) then we will not be able to find the path because no nodes exist which have a radii larger than the size of a single grid. To travel over the large distances we need to calculate the radii of the roads such as motorways and highways. To do this we nominated 50 nodes along extremities (i.e. the circumference) of network, and solved all-to-all the shortest paths from these extreme nodes to additional update the radii. By computing the shortest paths over those extreme points, we are hence able to determine the nodes used frequently over the large distances and as a result gives them large radii. The below mentioned is our heuristic algorithm to estimate radii of nodes:

• Phase 1.

Divide Network into the grids of approximately 2000 nodes $N_{SUB1} \in$ N.

Initialise the Radii of all nodes n ∈ N to 0

While the time permits

choose a random node s ∈ N

choose random node t within same grid as s (i.e. t $\in N_{SUB1}$).

Solve shortest path R from s to t

UpdateNodeRadii(R)

Loop

• Phase 2

For all nodes n ∈ N **do**

if r(n) = 0 and **ClosedNode**(n) = **false then**

Define a sub-graph G_{SUB2}containing:

f 200 of the closest nodes $N_{SUB2}$to n

All the shortest paths $R\ \mathcal{E} R_{SUB2}$defined on G_{SUB2}

for every the shortest path R $\in R_{SUB2}$**do**

UpdateNodeRadii(R)

next R

end if

next n

• Phase 3

Describe extreme nodes $N_{EXTREME}$

for all the shortest paths R from s $\in N_{EXTREME}$ to t $\in N_{EXTREME}$: s ≠ t do **UpdateNodeRadii(R)**

next R

The key to the presentation of this algorithm is loop condition "while time certificates" in 'Phase 1'.more iterations executed in this loop, the extra shortest paths are going to be explored, hence giving the better estimate of lower bound of the radii. We ran the first phase for about 70 hours, and hence explored approximately a quarter of the million shortest paths. The basic indication behind this is heuristic algorithm is to iteratively develop the lower bound on radii. In first and second phases we develop lower bound on the radii of the nodes to give analysis of local paths (i.e. paths within the grids). In third phase we fundamentally have just one grid cover the entire network. By solving all-to-all

The shortest paths along the extreme nodes we are capable to update the radii so that we can succeed coverage of the global paths. The end consequence of this pre-processing is the radius arrangement that guides searches over large distances through the fast roads. We can then custom these pre-processed radii in the conjunction with a search algorithm to the restrict the search space. These radii will have property that further in the distance we get it from the either origin or destination the more careful the search becomes on the type of the roads it uses. This benefit alleviates the combinatorial effects suffered by original search algorithm.

5. Performance Analysis Search Algorithms

In this paper we implemented 4 search algorithms:

1. Dijkstra's algorithm Labelling (named as 'Dijkstra's in our results)

2. Dijkstra's Bi-directional algorithm (named 'Symmetric' in our results)

3. A* algorithm (named 'A*' in our results)

4. Dijkstra's Bi-directional algorithm with in radius limit (named 'Radius' in our results)

From the experimental outcomes plotted in Figure 1, we can see that the Symmetric Dijkstra's algorithm is nearly twice as fast as Dijkstra's algorithm and A* algorithm is nearly three times faster than Dijkstra's Algorithm. However the most inspiring reduction in time is through Symmetric Dijkstra's algorithm used in conjunction with radius limit which is almost 50 times faster on an average.

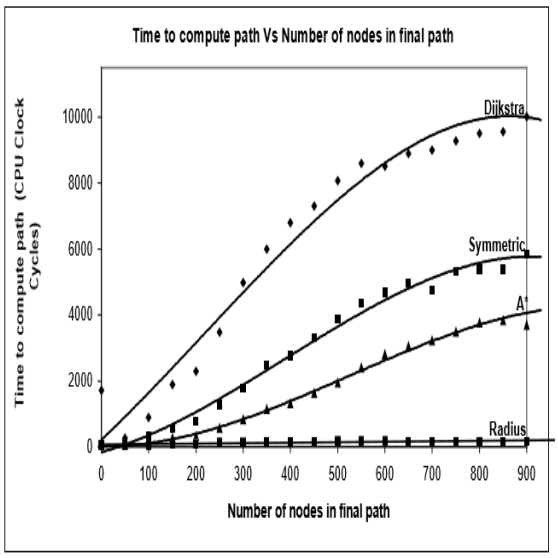

Figure 1: Time to compute paths using different algorithms

While this shows that Radius algorithm can pointedly reduce the search time when associated with the Dijkstra's algorithm, we have to put this into the perspective by associating these gains in the reduced search space to the inaccuracies formed by using the heuristic approach. Figure2 shows that Path Inaccuracy Rate (PIR), distinct as the relative increase in the cost over the optimum value, for the executed algorithms.

Both the Dijkstra's and Symmetrical Dijkstra's algorithm assurance optimality, hence we would guess them to have 0 PIR. Though our implementations of A* and the Radius algorithms do not assurance optimal results. From the Figure 2 we can see that t average PIR of A* algorithm is virtually level at approximately 0.5%. However the inaccuracies of Radius algorithms produce approximately exponentially with number of nodes in the path. But even with this the exponential growth in errors, the biggest PIR for our network is within the 5%

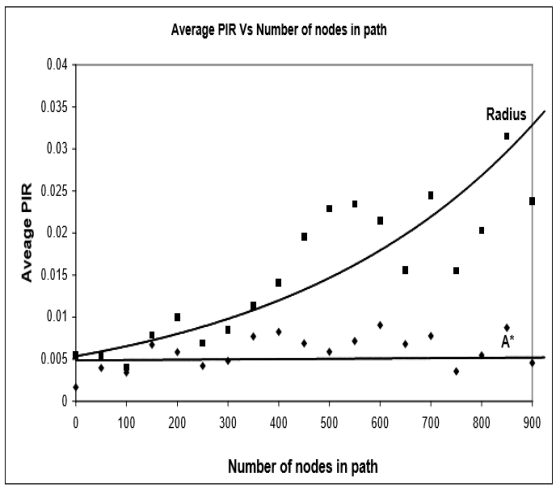

Figure 2: the number of nodes in the path vs Average Path Inaccuracy Rate (PIR)

6. Conclusions

By manipulating the physical structure of the road networks, A* algorithm is able to bias its search towards the goal and decrease the search space. By using the idea of radii as a measure of the importance of nodes, we are also able to incorporate pre-processing within the shortest path algorithm to the further limit the search space. This dramatically decreases the search complexity in terms of run time performance though still maintaining a suitable level of inaccuracy.

Acknowledgment

I would like to thank my parents, whose love and support has boosted my confidence at every step of life. I would also wish to express my gratitude towards my wife Ms. Zubaida Aijaz for providing the unending support and motivation.

I am also grateful to Dr. Chitaranjan Sharma whose guidance made this work possible.

References

[1] Wang, Peng and Tseng, Energy balanced Dispatch of Mobile Sensors in a Hybrid Wireless Sensor Network, IEEE Trans. On parallel and distributed systems, vol. 21(Issue 12), December, 2010

[2] Hart and Nilson, A formal basis of the heuristic determination of minimum cost paths, IEEE Transactions of Systems Science and Cybernetics 4(2), pp. 100-107, 1968

[3] Pearsons, Heuristic Search in Route Finding, University of Auckland pp 123(1996).

[4] Sedgewick and Vitter, Shortest Paths in Euclidean Graphs. Algorithmica 1, pp. 31-48, 1986.

[5] 5.Robot. Res , Path planning using tangent graph for mobile robots among polygonal and curved obstacles, MlT Press, vol. 11(Issue 5), pp. 376-382, 1992.