Finding the best path in a partially known graph

Antonio Moreno

Computer Science and Mathematics Department
University Rovira i Virgili - URV
Av. dels Països Catalans, 26. 43007-Tarragona, Spain
amoreno@etse.urv.es

Abstract. The problem of finding the shortest path between two nodes in a directed, positively labelled graph has been extensively studied. It can be solved using global methods (such as an *A** heuristic search) or local methods (such as the *LRTA** algorithm). However, these techniques require the graph to be completely known in advance. In this paper we propose a modification to the *LRTA** algorithm which permits the use of this technique to find the shortest paths in graphs which are partially unknown or change dynamically over time.

1. Introduction

Finding the shortest path between two nodes in a directed, positively labelled graph is one of the most basic problems in Artificial Intelligence, and it has been extensively studied in the literature. If the graph is static and completely known, it is possible to find the complete optimal path between two given nodes by making an *A** search of the target node from the origin node ([6]). This method is outlined in section 2. The main problem of this technique resides in its computational cost and the need of knowing the whole graph. One way of overcoming the computational cost is to have *local* algorithms, in which the decision of which edge to follow is taken in each node of the path (rather than calculating the whole path in advance). An example of this kind of algorithms is the *LRTA** (*Learning Real Time A**) procedure ([1]), shown in section 3. This method is complete, but it does not guarantee the optimal solution; however, it has the nice property of converging towards the optimal path after repeated executions. The main disadvantage of this method is that it also requires to know the complete graph. We have developed a project ([4]), described in section 4, in which an autonomous car is requested to find the optimal routes between pairs of nodes in a graph that it does not know. The main idea is that the car discovers the edges and nodes of the graph when it travels through it. Moreover, the graph is also dynamic, so edges can appear or disappear over time. In section 5 we show how the *LRTA** may be modified to help the car to find the optimal routes in this dynamic and partially unknown environment. The paper finishes with a brief discussion of the positive and negative features of this approach.

2. A global method: *A**

There are several techniques that may be used to find the shortest path between two nodes in a directed, positively labelled graph, when it is known and static (e.g. the classical Dijkstra algorithm may be applied, [2]). In this section we describe one of

them: the *A** heuristic search ([6]). All the methods described in the paper will be illustrated with the help of the graph shown in fig. 1. In all the examples G=(V,E) denotes a graph, where V is a set of *v* nodes and E is a set of *e* labelled edges.

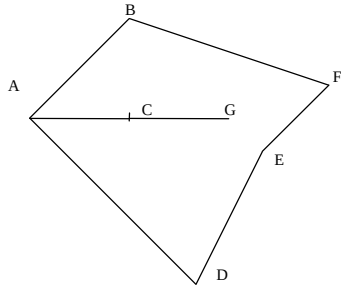

Fig.1: Graph to be used in the examples
(A-0,5 B-3,8 C-3,5 D-5,0 E-7,4 F-9,6 G-6,5)

An A* search builds a tree following this procedure:

```
VisitedNodes := empty set;
T := tree with root R (the initial node);
PathCost[R] := 0;
f(R) := h(R);
found := false;
while there are unvisited leaves in T and not(found) do
   p := leaf of T that is not in VisitedNodes with minimum f;
   if p is the target node
    then found := true;
    else
      VisitedNodes := VisitedNodes + p;
      for each edge (p,q,label) in E leaving from p do
        if q is not in VisitedNodes then
          add q as a son of p in the tree;
          PathCost[q] := PathCost(p) + label;
          f(q) := PathCost[q] + h(q);
        end if;
      end for;
   end if;
end while;
```

PathCost is an array, indexed by the nodes of the graph, that stores the optimal cost of going from the initial node *R* to each node in the graph. *h* is a heuristic function that estimates the cost of arriving from a node to the target node. This function is used to compute the function *f*, which takes into account the cost of going from *R* to a node plus the estimated cost of going from the node to the target. In order for this algorithm to find the optimal solution, the heuristic function *h* must be *admissible* (i.e. it must never over-estimate the real cost of reaching the target node). The shortest path to the target node is recovered by following the branch from the root to that node. In the example shown in fig. 1, the tree that would be built to find the optimal path between *A* and *E* would be the one show in fig.2. In that figure you can see the *f* value of each

node. The numbers outside the nodes indicate the order in which the nodes are selected and expanded.

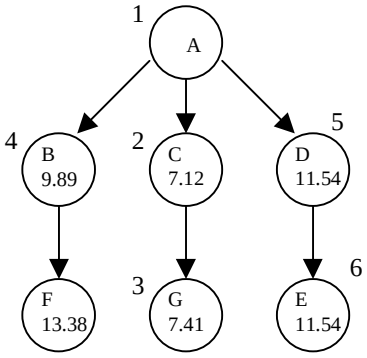

Fig. 2. Tree developed by the *A** algorithm to find the best path between *A* and *E*

In this tree it may be seen that the best path from *A* to *E* is *ADE*, and its cost is 11.54. The maximum number of nodes in the tree is *O(v*v)*, because in each step of the loop the set *VisitedNodes* is increased in one node and we only add edges that go to unvisited nodes.

The main drawbacks of the *A** search are the following:

- It calculates the whole path from the start node to the target node, and its computational cost is high (at least quadratic with respect to the number of nodes of the graph, since that is the number of nodes we may have in the tree).
- The whole graph has to be completely known.
- The graph must be static. If the graph is modified, the optimal paths have to be re-calculated.

3. A local method: *LRTA**

There are many applications in which the path betwen two nodes has to be calculated in real time, and the use of expensive global methods such as *A** is unfeasible. In these situations it is necessary to discover the optimal path at the same time that it is being traversed, rather than first calculating the whole path and then following it. Therefore, in this case we need *local* methods, which compute in each node which is the next branch that must be followed in the graph. One of these methods is the *LRTA** algorithm ([1]).

3.1 *LRTA**

The *Learning Real Time A** method ([1]) is a procedure that searches the optimal path between two nodes in a local way. It starts by deciding which edge to follow from the initial node. When the edge is taken and the next node is reached, then it decides which is the next branch to follow. This procedure is repeated until the target node is

reached. It must be noticed that, by taking purely local decisions, it is not possible to guarantee that the optimal path is being followed. The computation needed to take the decision in a node *p* is the following:

```
for all the edges e=(p,q,label) in E leaving from p do
  f(e) = label  + h(q,target);
end for;
e := edge with minimum f; (ties are broken randomly)
h(p,target) := f(e);
```

In this code, *h(n1,n2)* is a measure that estimates the cost of going from node *n1* to *n2*. This measure, as in the case of the *A** algorithm, must always be *admissible*. A simple way of initializing this measure is to set it to 0 or to the length of the straight line between *n1* and *n2*. The function *f* applied to an edge *(p,q,label)* estimates the cost of going from *p* to the target node by following that edge; it simply adds the length of the edge towards *q* and the estimation of the cost of going from *q* to the target node. It is interesting to note that the last line updates the estimate of the cost of reaching the target node from *p*, after having evaluated the costs of all the edges leaving from that node. The computational cost of the decision to take in each node is just linear with respect to the number of edges leaving that node.

3.2 Discussion of *LRTA**

The following properties of this method are known ([1], proofs in [3]):

Property 1. Completeness of *LRTA**
In a graph with a finite number of nodes with positive link costs, in which there exists a path from every node to a goal node, and starting with non-negative admissible initial estimates, *LRTA** is *complete*, i.e. it always reaches a goal node.

Property 2. Convergence to the optimal solution
If the initial estimates are admissible, the updates of the estimates tend to converge towards the optimal solution through repeated applications of the algorithm, i.e. the value of *h(n1,n2)* keeps increasing and it converges towards the cost of the optimal path between *n1* and *n2*.

Thus, the main advantages of this method is that it *runs in real-time* (the decision of which edge to follow is taken directly with local information in each node), it always *finds a path to the target node* (if it exists) and it *learns the optimal paths* through repeated traversals of the graph (due to the update of the *h* estimates).
It is interesting to see the application of this algorithm to the problem of finding the best route from *A* to *E* in the graph shown in fig. 1. Initially, *h(n1,n2)* will have the length of the straight line from *n1* to *n2*. The following decisions would be taken:

- Step 1 (node A)
  f(B)=4.24+5.65=9.89 f(C)=3+4.12=7.12 f(D)=7.07+4.47=11.54
  The edge (A,C) is chosen, and h(A,E) is updated to 7.12.
- Step 2 (node C)
  f(A)=3+7.12=10.12 f(G)=3+1.41=4.41
  The edge (C,G) is chosen, and h(C,E) is updated to 4.41.

- Step 3 (node G)
  f(C)=3+4.41=7.41
  The edge (G,C) is chosen, and h(G,E) is updated to 7.41.
- Step 4 (node C)
  f(A)=3+7.12=10.12 f(G)=3+7.41=10.41
  The edge (C,A) is chosen, and h(C,E) is updated to 10.12.
- Step 5 (node A)
  f(B)=4.24+5.65=9.89 f(C)=3+10.12=13.12 f(D)=7.07+4.47=11.54
  The edge (A,B) is chosen, and h(A,E) is updated to 9.89.
- Step 6 (node B)
  f(A)=4.24+9.89=14.13 f(F)=6.32+2.82=9.14
  The edge (B,F) is chosen, and h(B,E) is updated to 9.14.
- Step 7 (node F)
  f(B)=6.32+9.14=15.46 f(E)=2.82+0=2.82
  The edge (F,E) is chosen, and the final node is reached.

The path found by the algorithm is *ACGCABFE*, which is far from optimal. It has first tried to go towards *E* by the most promising edge, which was *AC*, but then it had to come back after reaching a dead end in *G*. Furthermore, the second route that it has tried (*ABFE*), has led to the goal node but is not optimal, since *ADE* is shorter. However, let us see what happens if we use the estimates learned in this execution to find again a path between *A* and *E*:

- Step 1 (node A)
  f(B)=4.24+9.14=13.38 f(C)=3+10.12=13.12 f(D)=7.07+4.47=11.54
  The edge (A,D) is chosen, and h(A,E) is updated to 11.54.
- Step 2 (node D)
  f(A)=7.07+11.54=18.61 f(E)=4.47+0=4.47
  The edge (D,E) is chosen, and the final node is reached.

Thus, the second time that we search the path from *A* to *E* the algorithm finds the optimal path, and *h(A,E)* has converged towards the cost of this minimal path. In any other request of the best path from *A* to *E*, the path *ADE* would be followed and *h* would not be modified.
Despite the positive properties of the *LRTA** algorithm, it can only be applied if we have a graph that is static and we know, in each node, which are the edges that leave from that node. Therefore, it will not be possible to apply this algorithm directly in cases in which the graph is only partially known or it changes in time. This is the case of the application described in the next section.

4. A dynamic and unknown environment

We developed a project in which the objective was to construct a small car that could follow the edges of a graph painted on the floor ([4]). The car has a camera that allows it to discover the edges and nodes of the graph. Once the mechanical and vision details were finished and the car was able to follow the white lines and go from node to node, we had to construct the software component. The user can specify paths that the car has to find: from the current node to a target node. When the car reaches the target node, the user defines a new target, and so on. One of the challenges of the

project was to build a path finding algorithm that could find the best paths to the target node, but taking into account three important facts:

- The car has initially *no knowledge of the graph*. It constructs the description of the graph when it is traversing it.
- The graph may be *dynamically changing* in time (at a moderate rate, otherwise the problem of trying to find optimal routes would be impossible to tackle). That means that we can add or remove edges of the graph from time to time.
- We wanted to define an algorithm that decides in each node which is the most promising branch to follow, i.e. the algorithm has to take *local decisions*, it does not have to construct the whole path in advance.

The car always maintains a *partial* description of the graph, with the edges and nodes that it has discovered in previous routes. Note that, on one side, there may be many edges and nodes which the agent does not know yet; on the other hand, there may be wrong information in the graph constructed by the car (it may contain links that no longer exist in the graph, due to its dynamic nature). These characteristics of the domain make it unfeasible to use the *LRTA** algorithm directly. In the next section we explain how we used the main idea of the algorithm to provide a good path finding algorithm for the car.

5. Modification of the *LRTA** algorithm

When the car is located in a particular node, it receives the request of travelling from that node to another one. The objective of our project was to find the optimal path from the present node to the target node. Of course, the dynamic changes in the graph and the fact that the car maintains only a partial description of the graph make it impossible to ensure that the optimal path will always be followed. However, it is still feasible to develop an intelligent algorithm that approximates (and learns through time) the optimal results. In the next subsections we detail the input data of the algorithm and the process that it follows to select the best branch in each node. After that, we show an example of the application of the algorithm.

5.1 Input data of the path finding algorithm

The data that the path finding algorithm has are the following:

- Coordinates *(x,y)* of the actual node and the target node.
- Partial description of the graph, with the edges and nodes that have already been visited (some edges might no longer exist).
- Information about the edges that leave from the present node.

This last point deserves some comments. The car has a bunch of circular sensors that give information about the white lines (the edges) that leave from the present node. Thus, the car knows how many edges leave from the actual node and the angle of each one; however, it can not know the length of the edges or the coordinates of the

succesor nodes. When the car reaches a node, it compares the local information about the edges with the partial graph; there are three possible situations:

- An edge present in the partial graph does not longer exist. In this case, the edge is removed from the partial graph.
- An edge of the partial graph coincides with one of the edges leading the node (we can only compare its angle). In this case, it is assumed that the edge has not changed since the last time it was traversed; therefore, the car knows its length and the coordinates of the next node.
- An edge leaving the node is not reflected in the partial graph. In this case, either the edge is new or it simply had not been traversed by the car before. The car only knows the direction of the edge, but it does not know its length.

With this information the car has to decide which of the edges has to be followed. For instance, in fig. 3 the car is located in *P* and has to decide between two known branches (that go to *Q* and *R*) and three unknown edges. After having traversed the chosen edge, the car will know its length and the coordinates of the next node, and it will add this information to the partial graph (if it is the first time that the car travels through that edge).

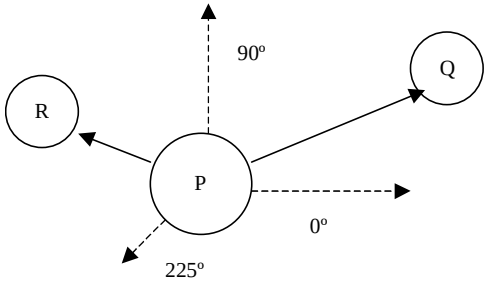

Fig. 3 Decision problem in node *P*

5.2. Decision algorithm

The algorithm that the car follows to decide which edge to take follows the same basic idea than *LRTA**. In the case of a known edge *e=(p,q,label)* leaving from the present node *p*, we make the same computation: *f(e) = label + h(q,target)*. In the case of an unknown edge *e*, from which only the direction is known, we propose to make the *most optimistic* evaluation of it: to assume that it has only length 1 (the minimal length that the car may move, which is the minimal length of an edge) and then we reach a node from which there is a straight link to the target node. Therefore, we will compute *f(e) = 1 + distance(s,target)*, where *s* is a point located at distance 1 from *p* in the direction of the unknown edge. This idea is illustrated in fig. 4, when the car is in node *P* and has to go towards the target node *T*. The evaluations of the unknown edges are *f(e1)=1+d1*, *f(e2)=1+d2* and *f(e3)=1+d3*. The evaluations of the known edges are *f(PQ)= distance(PQ) + h(Q,T)* and *f(PR)= distance(PR)+h(R,T)*.

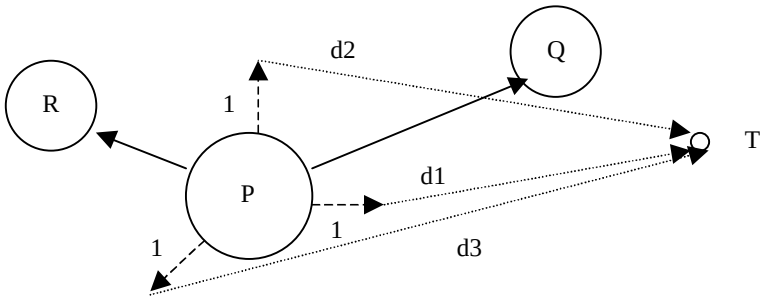

Figure 4. Optimistic evaluation of the unknown edges

5.3 Example

Let us consider again the problem of finding the optimal path between *A* and *E* in the graph shown in fig.1. We will now apply the algorithm outlined in the previous section, considering that initially the car does not have any knowledge of the graph (it only knows its actual coordinates and the coordinates of the target node *E*). This is the process followed by the algorithm:

- Step 1 (node A, 3 unknown edges)
  f(e1)=1+6.52=7.52 f(e2)=1+6.08=7.08 f(e3)=1+6.30=7.30
  The second edge (which is (A,C)) is chosen, and h(A,E) is updated to 7.08
- Step 2 (node C, 1 known edge -CA- and 1 unknown edge)
  f(A)=3+7.08=10.08 f(e4)=1+3.16=4.16
  The unknown edge (which is (C,G)) is chosen, and h(C,E) is updated to 4.16.
- Step 3 (node G, 1 known edge, GC)
  f(C)=3+4.16=7.16
  The edge (G,C) is chosen, and h(G,E) is updated to 7.16.
- Step 4 (node C, 2 known edges)
  f(A)=3+7.08=10.08 f(G)=3+7.16=10.16
  The edge (C,A) is chosen, and h(C,E) is updated to 10.08.
- Step 5 (node A, 1 known edge -AC- and 2 unknown edges)
  f(e1)=1+6.52=7.52 f(C)=3+10.08=13.08 f(e3)=1+6.30=7.30
  The last unknown edge (which is (A,D)) is chosen, and h(A,E) is updated to 7.30.
- Step 6 (node D, 1 known edge -DA- and 1 unknown edge)
  f(A)=7.07+7.30=14.37 f(e4)=1+3.47=4.47
  The unknown edge (which is (D,E)) is chosen, and h(D,E) is updated to 4.47. The final node is reached.

Note that, in this example, the performance of the proposed modification of the *LRTA** algorithm is better than the one of the standard algorithm, since it discovers the optimal route *(ADE)* after trying the most promising (but wrong) route through nodes *C* and *G*. At this point, the car has knowledge of the edges AC, CG, AD and DE. If we later ask again about the path from A to E, the decision process would be the following:

- Step 1 (node A, 2 known edges -AC, AD- and 1 unknown edge)
  f(e1)=1+6.52=7.52 f(C)=3+10.08=13.08 f(D)=7.07+4.47=11.54
  The unknown edge (which is (A,B)) is chosen, and h(A,E) is updated to 7.52.

- Step 2 (node B, 1 known edge -BA- and 1 unknown edge)
  f(F)=4.24+7.52=11.76 f(e5)=1+3.32=4.32
  The unknown edge (which is (B,F)) is chosen, and h(B,E) is updated to 4.32.
- Step 3 (node F, 1 known edge -FB- and 1 unknown edge)
  f(B)=6.32+4.32=10.64 f(e6)=1+1.82=2.82
  The unknown edge (which is (F,E)) is chosen, and h(F,E) is updated to 2.82. The final node is reached.

In this case, even though the optimal route *ADE* was already known, the car has decided that it was worth exploring the unknown path through *B* and *F*, since it could have been shorter (although, in fact, it is longer). After this travel, the car has already visited all the edges of the graph. Let us see what happens if we ask for the path between *A* and *E* for the third time:

- Step 1 (node A, 3 known edges)
  f(B)=4.24+4.32=8.56 f(C)=3+10.08=13.08 f(D)=7.07+4.47=11.54
  The edge (A,B) is chosen, and h(A,E) is updated to 8.56.
- Step 2 (node B, 2 known edges)
  f(F)=6.32+2.82=9.14 f(A)=4.24+8.56=12.80
  The edge (B,F) is chosen, and h(B,E) is updated to 9.14.
- Step 3 (node F, 2 known edges)
  f(B)=6.32+9.14=15.46 f(E)=2.82+0=2.82
  The edge (F,E) is chosen, and h(F,E) is updated to 2.82. The final node is reached.

This time the car would travel again by the non-optimal route *ABFE*, but it has changed the estimation of the cost of arriving from *B* and *A* to *E*. These changes cause a different behaviour in a fourth request of the same route:

- Step 1 (node A, 3 known edges)
  f(B)=4.24+9.14=13.38 f(C)=3+10.08=13.08 f(D)=7.07+4.47=11.54
  The edge (A,D) is chosen, and h(A,E) is updated to 11.54.
- Step 2 (node D, 2 known edges)
  f(A)=7.07+11.54=18.61 f(E)=4.47+0=4.47
  The edge (D,E) is chosen, and h(D,E) is updated to 4.47. The final node is reached.

This time the car follows the optimal route again, and updates the *h* estimates with the actual values of this path. In any other request of the route from *A* to *E*, the car would now always follow this path, and the *h*-estimations of *A* and *D* would no longer change.

6. Discussion

In this paper we have described a modification of the *LRTA** algorithm that allows its application in domains in which the graph to be explored is only partially known or may change dynamically over time. This algorithms has the following positive properties:

- It is a *local* algorithm, with a very *reduced computational cost* because it only evaluates the branchs available in each node.

- The algorithm may be used in graphs that are only *partially known*, using an optimistic view of the unknown edges. If an unknown edge can not posibly lead to a shorter path that a known edge, it will not be explored. However, if there is the possibility of finding a better path by exploring an unknown edge, the car will follow it.
- The estimations used in the heuristic evaluation of each node are always *admissible* (both for known and unknown edges). Therefore, the properties of *completeness* and *convergence* to the optimal solution after repeated executions are maintained in our version of the algorithm.
- If edges are *removed* dynamically from the graph, the estimates will continue to be optimistic, since optimal paths can only get longer. When the car arrives to one of the nodes that was connected to the removed edge, it will notice that it no longer exists and it will delete it from its partial representation. From that point on, the car will be able, in succesive travels, to change the *h* estimates in order to find again the optimal paths.

The two main drawbacks of the algorithm are the following:

- As we are dealing with a partial representation of the graph and, furthermore, the graph may change in time, we can never be sure that the car will follow the optimal path between two nodes. However, if the graph reaches a stable state and the car has made enough travels to update the *h* estimates to the actual values of the optimal routes, then it will always follow these routes.
- If edges can be dynamically *added* to the graph, then the *h* estimates are no longer admissible, and the properties of completeness and (eventual) optimality are lost (for instance, adding a link between *G* and *E* in the example graph would not modify the routes from *A* to *E*). This aspect is the subject of our future work. We can advance some comments on this issue:
  - If the new edge connects two nodes that have still not been visited by the car, then the *h* estimates do not have to be modified and the algorithm will continue to work correctly.
  - We may re-set the *h* estimates to the Euclidean distance each time that an edge is added, and the car will have to learn again the optimal routes.
  - The car may be lucky enough to recover and learn again optimal paths using the new edge, depending on the routes it is obliged to make.

References

[1] Korf, R.E., ``Real-time heuristic search". *Artificial intelligence*, 42 (2-3), pp. 189-211, 1990.

[2] Dijkstra, E.W., "A note on two problems in connexion with graphs", *Numerical Mathematica*, 1, pp. 269-271, 1959.

[3] Yokoo, M., Ishida, T., ``Seach algorithms for agents", in G.Weiss (Ed.), *Multiagent systems: a modern approach to Distributed Artificial Intelligence*, pp. 165-199. MIT Press, 1999.

[4] Vargas, M., Medina, D., León, X, "*Vehículo rastreador. Una aplicación de la IA*". MSc projects, University Rovira i Virgili. Tarragona, September 1999.

[5] Pearl, J. "*Heuristics: intelligent search strategies for computer problem solving*". Addison-Wesley, 1984.

[6] Hart, P.E., Nilsson, N.J., Raphael, B., "A formal basis for the heuristic determination of minimum cost paths", *IEEE Transactions on Systems, Science and Cybernetics*, SSC-4 (2), pp. 100-107, 1968.