PALADYN Journal of Behavioral Robotics

VERSITA

Research Article · DOI: 10.2478/s13230-012-0013-4 · JBR · 3(1) · 2012 · 23-34

Robot Path Planning using Dynamic Programming with Accelerating Nodes

Rahul Kala*, **Anupam Shukla**†, **Ritu Tiwari**‡

Soft Computing and Expert System Laboratory, Indian Institute of Information Technology and Management Gwalior, Gwalior, Madhya Pradesh-474010, India

Received 2011/09/11

Accepted 2012/01/26

Abstract

We solve the problem of robot path planning using Dynamic Programming (DP) designed to perform well in case of a sudden path blockage. A conventional DP algorithm works well for real time scenarios only when the update frequency is high i.e. changes can be readily propagated. In case updates are costly, for a sudden blockage the robot continues moving along the wrong path or stands stationary. We propose a modified DP that has nodes with additional processing (called accelerating nodes) to enable different segments of the map to become informed about the blockage rapidly. We further quickly compute an alternative path in case of a blockage. Experimental results verify that usage of accelerating nodes makes the robot follow optimal paths in dynamic environments.

Keywords

path planning · dynamic programming · artificial neural networks · A · robotics · heuristics*

1. Introduction

Robot Path Planning is a much studied problem in the field of robotics. The problem is much harder to solve for dynamic maps, which have larger issues than static path planning, due to a constantly changing map and the real time nature of the algorithm. As a result many approaches primarily work only in static environments. In dynamic maps, the path of the robot may frequently be completely obstructed by obstacles. In such a case a blockage is said to have occurred and the robot needs to re-plan the path. Blockage is a still harder problem in robotics, as there may be few similarities in the optimal path after blockage to the one prior to blockage. This fundamentally means re-planning the entire robotic path. For most algorithms this re-planning may be very time consuming.

Our motivation is to study the problem of path planning in case of blockages, and to propose a new algorithm that performs well in case of such blockages. In a dynamic map blockages happen suddenly. While the robot is moving, there is little time available for the algorithm to detect the blockage, and once detected to decide the plan of action. A good algorithm must hence be able to guide the robot towards an optimal path immediately after the occurrence of a blockage.

We choose Dynamic Programming as the base algorithm for work [1]. The basic concept behind DP is to break up a problem into multiple related sub-problems. A sub-problem may in turn be broken down into similar sub-problems and so on until the sub-problem is of a unit size. Solving the main problem naturally requires solving a large number of sub-problems. DP attempts to solve all unique sub-problems once only, and these results are stored in a memory structure from where they may be fetched next time they need to be solved.

In robot path planning, the problem solved by DP is to calculate the distance of the goal from all the points in the map. Distance to further points (problem) is computed based on the pre-computed distance to nearer points (sub-problem). In this way the entire problem is divided into sub-problems, with a unit sub-problem as the distance of the goal from itself, which is zero.

We use a recurrence relation to show the relation between a problem and its sub-problems. Iterative solutions of the recurrence relation, starting from the smallest sub-problem and proceeding towards the higher sub-problems, may result in the solution of the main problem. In the problem of path planning we calculate the shortest distance from any point to destination at iteration $i+1$ using the shortest distance of neighboring points at iteration i. Every time we update the shortest path of a point, we even update the source from which this smallest path was found. This helps in building up the path that corresponds to the smallest path length. It can be easily shown that for all points, smallest distances can be computed after a maximum of I iterations where I is the total number of points in the map.

The basic DP may be able to solve the problem of robot path planning to a good extent; however it is unable to solve the problem of blockages which is the main intent behind the work. In this work we hence propose a new set of nodes, called the accelerating nodes, which aid the DP in performing well in cases of blockages. These nodes are fewer in number and give a coarser representation of the map. These nodes are able to quickly detect blockages and further build an approximate path from the current position to goal in case of a blockage. The robot may use this path for its immediate motion, while the optimal path may take time to be built.

Another similar approach to solve the problem is with the use of Artificial Neural Networks [2]. Here there are numerous models that work on the basic principle of propagating the least distances from the destination in various directions. As the passage of information takes place between the nodes, the distances are computed and passed. The obstacles have a fixed maximum distance. These approaches also include the solutions using shunting equation for propagation of distances [3]. A point in our DP formulation plays a similar role to a neuron. However neurons have different dynamics governed by neural activation, unlike our approach.

*E-mail: rahulkalaiiitm@yahoo.co.in
†E-mail: dranupamshukla@gmail.com
‡E-mail: rt_twr@yahoo.co.in

Lebedev et al. [4] used a type of neural network called dynamic wave expansion neural network for the efficient and optimal planning of the robot in a real time environment. Here every grid is a neuron that propagates some activity to the other nearby neurons or grids. The obstacles and goals are clearly identified by the activity they propagate. The robot motion is computed by measuring this activity, which is driven by the intrinsic neuron equations, also referred to as the shunting equations. The model proposed avoided the robot oscillating in certain likely situations. In turn, the robot was made to remain stationary and wait for a new activity being propagated by the network. All these approaches cannot perform well in case of blockages unless the update frequency is very high, which demands very high computational requirements. The only way of sensing and acting in response to blockages is through the activity of neurons. This may lead the robot to come very near to the blockage and it may then have to back track its path, which is clearly sub-optimal.

DP principles are also used in numerous graph search algorithms abundantly used for robotic planning. Literature points to extensive use of A* algorithm [5] which is additionally guided by heuristics to direct the search process towards goal. The various points on the map correspond to the nodes of the A* algorithm, with weighted connections between two neighboring nodes if the robot can directly move between these nodes. The weights correspond to the Euclidean distances between the nodes. The A* algorithm considers both historic as well as heuristic cost to find the path between the source and the destination. It keeps expanding the nodes until the destination node is reached. Heuristics make the algorithm computationally less expensive; however in case of a blockage the only thing that can be done is to re-calculate the path. This means the robot has to wait for some time while the path is re-computed. In DP however the extra computations are a source of blockage detection and alternative path buildup, which justifies its high initial computational cost.

Both DP and graph search based techniques have a common problem of having a large number of nodes or high complexity which restricts their use in real time systems. An alternative is to use a multi-resolution approach where the map is represented at multiple resolutions and a planning algorithm may start the search process on a higher resolution and query lower resolutions if needed. Commonly used representations include Quad Tree [6], Mesh [7] and Pyramid [8]. Another representation is the framed quad-tree representation [9]. In this approach every grid is additionally broken into a number of unit sized grids that surround the main grids. This representation gives the capability to generate more flexible paths, at the same time limiting the number of grids for easy execution.

Kala et al. [10] used a similar approach of multi-resolution maps. Planning was initiated at the lower resolution and upon receiving a valid path, the resolution along areas of returned path was increased. Graph search was done using Multi-Neuron Heuristic search [11, 12], which is an advancement over the conventional A* algorithm. In another recent approach, multi-resolution based planning is presented [13] where the planning algorithm used is Lifelong A* algorithm (LPA*) [14]. The basic idea behind these approaches is quick computation of a path in case of blockage. Since the map is assumed to be of high resolution, the computation of a path using these hierarchical approaches may also take time, resulting in the robot waiting. When a blockage is detected, the algorithm needs to initiate the process of constructing the path right from the start. In DP with accelerating nodes, many of the computations of the previous iterations can be used to give an approximate path for immediate traversal of the robot. Further, solely using these approaches means a robot traveling on a sub-optimal path provides a computational speedup at the cost of minimal loss of optimality.

The D* [15] algorithm is the dynamic variant of the A* algorithm. Here the planning starts with an offline phase where heuristic guided graph search is performed. Change in the robotic map leads to re-planning and cost modification of nodes. Hierarchical planning may also be carried out using D* as presented in [16]. These algorithms perform well in dynamic environments with small changes in map, for which they are best suited. For cases of a path blockage there is a completely new path which needs to be determined. In such cases computational requirements are very large, which may require the robot to wait while a new path is sought. Blockage completely changes all costs, and hence the computations prior to blockage have less use after the blockage scenario.

Another popular mechanism to handle the problem of a large number of nodes is to sample out some nodes for the planning purpose. Rapidly-exploring random trees (RRT) [17, 18] are widely used for this purpose. They start the search process by using the source as the root of a tree-like data structure. Search proceeds by expansions of the tree structure until a goal is found. Recent works includes use of multiple RRTs and combining them to produce optimal results [19]. It is difficult to pass through narrow passages using these sampling based approaches for which a better sampling technique was proposed in [20].

Another popular sampling based method is probabilistic roadmaps (PRM) [21, 22] where random samples are drawn out from the entire map and connectivity is determined by a local search algorithm. Once formulated, the map may be made adaptive as per the changing environments [23]. Similarly, the local search may be performed on a coarser level before checking on a finer level, as most connections are collision prone which may be easily found at a coarser level [24]. Clark [25] presented a PRM solution for robot planning and control where multiple robots could communicate and build a complete roadmap.

Sampling based approaches are able to come up with a travel plan early, which may be used post detection of blockage. However this leaves the question unanswered of how blockages would be detected, which is a big problem in itself. Further sampling based approaches would start afresh the task of building up an alternative path. This means a greater time as compared to our notion of accelerating nodes. Also these approaches return sub-optimal paths. Hence the complete motion of the robot cannot be done by the results generated by these algorithms.

O' Hara [26] presents another unique algorithm that works at the hardware level with embedded systems. Here multiple sensors are embedded in the map which communicate with each other and propagate the distances to the goal. The robot simply queries the sensors and moves. A similar approach was used by Yao and Gupta [27] where each sensor was a local planning agent whose role was to guide the robot to the nearest sensor such that the robot advances towards the goal along a shortest path. These approaches are naturally better as they can rapidly detect blockages and plan an alternative path. However for these approaches to be applied it is important that the map already has sensors embedded. This restricts planning to some specific application areas and further may involve higher costs.

The basic purpose behind the use of DP with accelerating nodes is that some extra computation (while the robot is moving) can be used for rapid development of an alternative path in case of blockages. This enables the algorithm to handle the cases of blockages better. While the robot is traveling, most planning approaches would have no computation except a small validation phase to check the validity of the path being followed. While the robot travels at its pace, some amount of computations may be performed. It is here that we monitor and store map state as computed by all the nodes and accelerating nodes. These computations help in case of blockages.

The problem considered is path planning of a mobile robot. The planning especially needs to be near-optimal and computationally less expensive in scenarios of blockages. Since the map is dynamic and may

constantly change, the optimality of any travel plan cannot be guaranteed, although for static maps the planned path is optimal. It is assumed that the map is of high resolution. Further it is assumed that computation is limited and hence the maximum number of calculations per step of the robot needs to be limited.
The contributions of the work are: (i) We propose new nodes called accelerating nodes in the conventional structure of the DP, which work at a coarser map, while the normal nodes work at a finer level. (ii) We state a mechanism to deal with the problem of blockages consisting of blockage detection, alternative path buildup, and optimal path rebuildup. (iii) Through simulations we show the ability of DP to handle cases of moving obstacles, moving targets and blockages, generating near-optimal trajectories.
This paper is organized as follows. The problem of this work is formally introduced in Section 2. In Section 3 we briefly summarize the Dynamic Programming paradigm, discuss the problems with DP and present the concept and working of the accelerating nodes. The monitoring of changes and working of the algorithm in a dynamic scenario is explained in Section 4. Section 5 presents the simulation results. Section 6 gives the concluding remarks.

2. Problem Formulation

Map (donated by ζ) of size $m \times n$ is assumed to be already known. Each cell of this map $(x, y) \in \zeta$, $1 \leq x \leq m$, $1 \leq y \leq n$, corresponds to a navigable area ($(x,y) \in \zeta^{free}$), or an obstacle ($(x,y) \in \zeta^{obs}$). Here ζ^{free} denotes the region of the map free of obstacles and ζ^{obs} denotes the region of the map with obstacles ($\zeta^{free} = \zeta - \zeta^{obs}$). The obstacles may be static, where the position of the obstacles does not change over time, leading to a constant map ζ^{free}; or they may be dynamic where the position of obstacles changes over time, leading to a dynamic map $\zeta^{free}(t)$. The problem is to come up with a path $\tau\left(\bigcup_c (x(c), y(c))\right)$ which the robot may use for navigation without colliding with any of the obstacles or $(x(c),y(c)) \in \zeta^{free}(c)$. Here $(x(c), y(c))$ denotes the planned position of the robot at time c ($\geq t$) knowing the map ($\zeta^{free}(t)$) at the time the plan was generated (say t). While t denotes the global time, c denotes the time an action is scheduled to take place in the future. We assume that there is only one source ($S = \tau(0) = (x(0), y(0))$) and one destination ($G = \tau(T) = (x(T), y(T))$) both of which are given, and the aim of the robot is to reach the destination starting from the source. T is the time in which the robot is projected to reach the given destination. The entire path generated by the algorithm needs to be optimal in terms of length, so the objective is to minimize $||\tau||$.

Changes in the environment may be sudden or gradual. Let O be the set of obstacles and $area(t, o)$ be the region of the map occupied by obstacle o at time t, such that $\bigcup_{o \in O} area(t, o) = \zeta^{obs}(t)$. Sudden environment changes may be caused in real life by various factors such as an obstacle rapidly entering the map or emerging somewhere in the map ($O(t+1) = O(t) \bigcup o$), or an obstacle suddenly leaving the map or disappearing from somewhere in the map ($O(t+1) = O(t) - o$), or two obstacles blocking the space between them ($Blockage(o_1, o_2, t)$), or two obstacles clearing the way between them ($Clear(o_1, o_2, t)$), etc. Here the two functions are given by equation 1 and equation 2

$$Blockage(o_1, o_2, t) = \left(Boundary(t+1, o_1) \bigcap Boundary(t+1, o_2) \neq \phi \wedge Boundary(t, o_1) \bigcap Boundary(t, o_2) = \phi\right) \quad (1)$$

$$Clear(o_1, o_2, t) = \left(Boundary(t+1, o_1) \bigcap Boundary(t+1, o_2) = \phi \wedge Boundary(t, o_1) \bigcap Boundary(t, o_2) \neq \phi\right) \quad (2)$$

$Boundary(o)$ represents the boundary of an obstacle o. The sudden changes are very difficult to incorporate in many implementations of path planning algorithms since we may not be able to reuse many of the properties of solutions from the previous time step in the next time step. Sudden changes at many times may be equivalent to re-calculating the entire robotic path. This is difficult and in many cases time consuming. The effect of gradual changes to a map may not be that large which may be caused by gradual movements of the obstacles (equation 3) or similar reasons.

$$||Boundary(t+1, o) - Boundary(t, o)|| < \varepsilon_1 \neg Blockage(o, o_1) \forall o_1 \in O \quad (3)$$

Here ε_1 is a small number. In such scenarios only a few changes may be required in the robotic path returned by the algorithm to make it valid and optimal as per the new map. Consider you are traveling on a road with some cars. The internal movement of other cars does not appreciably affect your movements or path followed.
However many times changes may be considerable when they result in completely segregating two parts of the map, or in other words blocking the path, or alternatively clearing a path which the robot can now use to navigate. This results in invalidating the present path and engineering a new path by the algorithm. Let $\tau^*(t)$ $\left(\bigcup_c (x^*(t, c), y^*(t, c))\right)$ denote the optimal path at time t using a static map which appears as $\zeta^{free}(t)$. In such scenarios $||\tau^*(t+1) - \tau^*(t)|| > \varepsilon_2$ for small ε_2. In the same example consider a car covering the entire road. Now you may need to choose another road in order to reach the destination. A path planning algorithm in a dynamic environment is supposed to cater to the needs of all these situations.

The purpose behind the algorithm is efficient planning in cases of blockages. This emphasizes that the algorithm is quickly able to detect blockages and quickly direct the robot to move, by the optimal path considering the map after the blockage.

3. Accelerating Nodes

The contribution of this paper is to propose a new set of nodes called accelerating nodes within the conventional working of the DP. In order to understand the approach proposed, it is important to study the conventional DP and understand its limitations in dynamic scenarios,

	I (x-1,y+2)		J (x+1,y+2)	
P (x-2,y+1)	B (x-1,y+1)	C (x,y+1)	D (x+1,y+1)	K (x+2,y+1)
	A (x-1,y)	V (x, y)	E (x+1,y)	
O (x-2,y-1)	H (x-1,y-1)	G (x,y-1)	F (x+1,y-1)	L (x+2,y-1)
	N (x-1,y-2)		M (x+1,y-2)	

Figure 1. The neighborhood of any node in the map.

which we do in the following sub-sections. We then present the basic concept behind accelerating nodes.

3.1. Dynamic Programming

We first present the general DP technique of solving the problem of robotic path planning. The model has been motivated by the work of Willms and Yang [1] who used DP for real time robot path planning. At every iteration we try to propagate the distance from the destination to all the cells of the map. In this manner this approach is similar to the Neural Network approach used by Yang and Weng [2] and Zelingsky [3]. The surrounding few cells are used for the distance update or distance propagation at every iteration. These cells constitute the neighborhood of a cell.
Suppose that we are updating the cell $V(x,y)$ at an iteration i. In our model, the cells that constitute the neighborhood ($\delta(V)$) and thus aid in computation of the distance from the destination are given by equation 4 and shown in Figure 1.

$$\begin{aligned}\delta(V) = \{&A(x-1,y), B(x-1,y+1), C(x,y+1), D(x+1,y+1),\\ &E(x+1,y), F(x+1,y-1), G(x,y-1), H(x-1,y-1),\\ &I(x-1,y+2), J(x+1,y+2), K(x+2,y+1), L(x+2,y-1),\\ &M(x+1,y-2), N(x-1,y-2), O(x-2,y-1),\\ &P(x-2,y+1)\}\bigcap\zeta\end{aligned} \tag{4}$$

The weights of the edges are 1, $\sqrt{2}$ or $\sqrt{5}$. It may be noted that other cells in the figure are not considered directly as they may easily be visualized as a combination of 2 or more moves totaling to same weight. The recurrence relation of the DP to calculate the distance $D(V, i+1)$ of the target (G) from any node V at an iteration $i+1$ using the values of iteration i may hence be given by equation 5.

$$D(V,i+1) = \left\{\begin{array}{ll} 0 & V = G \\ D_{\max} & V \in \zeta^{\text{obs}} \\ \min\left(D_{\max}, D(Z,i) + \left\|V - Z\right\|\right) & \text{otherwise}\end{array}\right\}$$

$$D(V,0) = \left\{\begin{array}{ll} 0 & V = G \\ D_{\max} & \text{otherwise}\end{array}\right\} \tag{5}$$

Here Z is given by equation 6.
$D_{\max}$ is the maximum possible value of $D(V)$. All values greater than $D_{\max}$ are trimmed down to $D_{\max}$.

$$Z = z{:}D(z,i)+\left\|V-z\right\| \leq D(k,i)+\left\|V-k\right\| \wedge z,k \in \delta(V) \forall k \neq z \tag{6}$$

We further need to keep a track of the manner in which the update is taking place to facilitate the construction of the path with the shortest distance to the goal. This is done by using equation 7 that stores the successor $S(V, i+1)$ of any node V from the information up until iteration $i+1$.

$$\begin{aligned} S(V,i+1) &= Z \\ S(V,0) &= \text{null}\ \forall V \in \zeta \end{aligned} \tag{7}$$

At any instance of time the robot moves to the successor $S(V)$ while placed at node V. In this manner the robot traverses the entire path from the source to the destination. The successor of destination is kept *null*. We do not assume the number of cells to be small for real time operations. As a result the algorithm has an initial setup phase where the initial values of $D(V, i)$ and $S(V, i)$ are computed for all V. The robot has to be stationary until this phase has completed. Once the source is given a non-null value of $S(V,i)$, the robot can start moving. It is natural that in a static environment the robot would move as per the path calculated by the DP. Also, the path would be optimal in nature due to the properties of DP.

3.2. Dynamic Programming for Dynamic Scenarios

In a static environment the robot continues its motion by following the successor $S(V,i)$ at every node V resulting in the path τ. However, in a dynamic case, this may not be possible as some obstacle might move into the path of the robot. The algorithm then needs to adjust accordingly. DP is able to perform sufficiently well in the case of a dynamic environment as well.
In most dynamic environments it is common to have obstacles that move constantly with time. The motion of the obstacle changes the map ($\zeta^{\text{free}}(t)$). The motion of the obstacles causes a change in the optimal path of the robot ($\tau^*(t+1) \neq \tau^*(t)$). The DP keeps updating the values of $D(V, i)$ and $S(V,i)$ using equations 4, 5, 6 and 7. Hence as soon as the obstacle moves, the distances are re-calculated as per the modified map. These changes would need time to propagate to all sections of the map from the location of the change. Let the robot be moving with an optimal plan at time t which is $\tau^*(t)$. The number of iterations (I) of DP required to generate an optimal path at time $t+1$ which is $\tau^*(t+1)$ is given by equation 8.

$$I = nodes(\tau^*(t+1,c)), t \leq c \leq c_2 \tag{8}$$

$nodes()$ returns the number of nodes on the input path.
c_2 is the projected time corresponding to the first common cell between $\tau^*(t+1)$ and $\tau^*(t)$ beyond which the two paths are equivalent. This is given by equation 9.

$$\begin{aligned} c_2 = d{:}\ &\min_d(\tau^*(t+1,e) \in \tau^*(t) \wedge \tau^*(t+1,d-1)\\ &\notin \tau^*(t) \forall e, t < d < e \leq T) \end{aligned} \tag{9}$$

The statistics are therefore wrong if fewer than I iterations have taken place after the obstacle moves. After these iterations the optimal path $\tau^*(t+1)$ may be generated.

In most cases, an obstacle motion produces only a small a change in distance values (equation 3). Let us consider motion of a single obstacle o from time t to time t+1. Modified distances are propagated from point c_2 and take I iterations for generation of optimal plan $\tau^*(t+1)$. If the updates of DP per unit time are fewer than I, it is evident that the robot would continue its motion on a non-optimal path. Since changes in the map $\zeta^{\text{free}}(t+1)$ are small and there is no blockage, we can compute a path $\tau'(t+1)$ $\left(\bigcup_c (x'(t+1,c), y'(t+1,c))\right)$ by the smallest deviation of the path $\tau^*(t)$ such that $(x'(t+1,c), y'(t+1,c)) \in \zeta^{\text{free}}(t+1)$. It is evident that cost of $\tau'(t)$ is almost equal to that of $\tau^*(t)$ (or $abs(||\tau'(t+1)||-||\tau^*(t)||) < \varepsilon_3$ for small ε_3). After a unit update of DP, all non-obstacle nodes V in the path of the robot receive a value $D(V, i) < D_{\max}$, which means the robot cannot collide with an obstacle (if not already at an obstacle node). This means the robot starts following a path similar to $\tau'(t+1)$ until I DP updates are made, after which it follows an optimal path $\tau^*(t+l_c)$ which may itself be close to $\tau'(t+1)$. l_c denotes the time taken for I DP updates. We know $abs(||\tau'(t+1)||-||\tau^*(t)||) < \varepsilon_3$ and $abs(||\tau^*(t+1)||-||\tau^*(t)||) < \varepsilon_3$ (changes in map are small). Hence following path $\tau'(t+1)$ is sub-optimal, however, the loss of optimality is not significant. If the obstacle is continuously moving, we may similarly argue that there is a non-significant loss of optimality as long as no blockage occurs, and the number of DP updates per unit motion of robot is more than unity. The loss of optimality may however be proportional to total traversal of the obstacle, summing a small loss at every move.

The other case may be that the target is moving. In this situation as well, the optimal path of the robot changes as the target moves. The update is carried in the map which is later reflected or propagated by the DP using equations 4, 5, 6 and 7. Consider motion of the target from G_1 at time t to G_2 at time t+1. We consider that the target signifies a physical entity of interest and hence (assuming no dynamic obstacles) the path of the target from G_1 to G_2 is collision free. This means that a path $\tau'(t+1)$ is possibly formed by appending the traversed path of the target to the original robot path, or $\tau'(t+1) = \tau'(t) \bigcup G_2$. If the target motion is small, by same reasoning we may have $abs(||\tau'(t+1)||-||\tau^*(t)||) < \varepsilon_3$ and $abs(||\tau^*(t+1)||-||\tau^*(t)||) < \varepsilon_3$ which means the loss in optimality on initially tracing the path close to $\tau'(t+1)$ is small. If the target moves continuously, similar reasoning may be applied. Again, loss of optimality would be proportional to the total motion of the target which emphasizes that the gap between target and robot (following a sub-optimal path) should decrease with time. If the target moves faster than the robot, it is hence possible that the robot may never catch the target. Similarly, the number of DP updates per unit move of the robot should be more than the number of nodes traversed by the target in order to make construction of the path $\tau'(t+1)$ possible.

The most interesting condition is when a blockage takes place affecting the optimal path (equation 1) or a blockage becomes cleared (equation 2). This may result in a significant change of optimal path or $abs(||\tau^*(t+1)||-||\tau^*(t)||) >> \varepsilon_3$. In case of blockage it means producing a path $\tau'(t+1)$ by small deviations may not be possible. Moving by $\tau^*(t)$ may be regarded as random with no certainty of moving towards goal. Further it may require a long time for the optimal plan to be built (equation 8 and 9). If the map is of high resolution or the DP iterations are expensive, it would take a long time for the change to propagate. Hence the discussed DP approach may not work unless the updates are very fast or the map is of small size.

3.3. Accelerating Nodes

Looking at the limitations of DP in case of blockages, we need to model the problem in such a manner that the changes in the map which may largely affect the path are well identified. We achieve this through the use of accelerating nodes. They monitor the changes in the distances and hence sense the blockage in the path presently being used by the robot. In case of a blockage an approximate path is swiftly built, until the main path is being computed. These nodes are put into the map in addition to the normal nodes and work in the map at a coarser level.

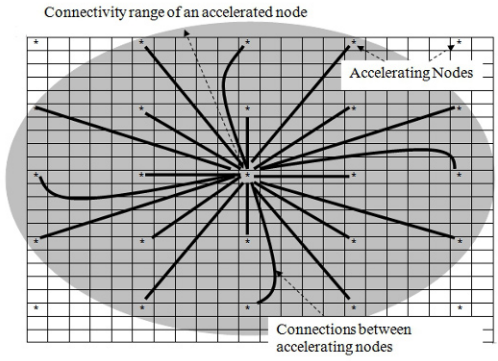

Figure 2. Map showing accelerating nodes.

The concept of accelerating nodes may be easily understood with the help of Figure 2. Here the various nodes marked with '*' are the accelerating nodes. From a general theoretical perspective these nodes are blockage detectors and path re-constructors which may be placed anywhere in the map. Large number of accelerating nodes may require a large computational cost which means their total number has to be limited. However, a large number of nodes may aid in early blockage detection and an optimal alternative path construction. If an a priori analysis of the robotic map can be done, it would be viable to place these nodes near the optimal travel plan of the robot at areas where blockage may be likely, as well as at the alternative areas to aid construction of a new alternative path. The same reasoning holds for the alternative path. In this paper we however do not rely on any such a priori analysis and present a uniform placement strategy. All these nodes always lie in points without obstacles. In case the uniform placement results in an accelerating node being placed at a point with an obstacle, that node is not placed at all, resulting in lowering the total number of accelerating nodes in the graph. Let N be the number of accelerating nodes in the map.

All the nodes of the map, V, store distances from these accelerating nodes in addition to the target. Let Q be the set of accelerating nodes. Every accelerating node Q_j denotes a physical position on the map. Hence we re-formulate the notations of Section 3.1 to denote $D(V,Q_j,i)$ as the distance of node V in the map from the accelerating node Q_j after i DP iterations, and $D(V, G, i)$ to denote the distance of node V from the target after i DP updates. The corresponding successors are given by $S(V,Q_j,i)$ and $S(V, G, i)$. The update procedure for the target remains the same as shown in equation 4, 5, 6 and 7. The update procedure of the accelerating nodes is given by equation 10.

$$D(V, Q_j, i+1) = \left\{ \begin{array}{ll} 0 & V = Q_j \\ D^Q_{\max} & V \in \zeta^{\text{obs}} \\ \min\left(D^Q_{\max}, D(Z, Q_j, i) + \|V - Z\|\right) & \text{otherwise} \end{array} \right\}$$

$$D(V, Q_j, 0) = \left\{ \begin{array}{ll} 0 & V = Q_j \\ D^Q_{\max} & \text{otherwise} \end{array} \right\} \quad (10)$$

Here Z is given by equation 11.
$D^Q_{\max}$ is the maximum possible value of $D(V,Q_j)$. All values greater than $D^Q_{\max}$ are trimmed down to $D^Q_{\max}$.

$$Z = z{:}D(z, Q_j, i) + \|V - z\| \leq D(k, Q_j, i) + \|V - k\| \wedge z, k \in \delta(V) \forall k \neq z \quad (11)$$

The successor of any node S(V, Q_j, i) is given by equation 12.

$$S(V, Q_j, i+1) = Z \quad S(V, Q_j, 0) = \text{null } \forall V \in \zeta \quad (12)$$

In such a way $|Q|$ accelerating nodes may be visualized as $|Q|$+1 agents ($|Q|$ accelerating nodes and 1 target) propagating their distances. Each node V has to store $|Q|$+1 distances. In this manner the computational cost for each update of DP increases with an increase in the number of accelerating nodes.
The major difference between the target and accelerating nodes is that the maximum distance $D^Q_{\max}$ for these nodes is much less than that of $D_{\max}$ of the target ($D^Q_{\max} < D_{\max}$). This is done to restrict the view of all nodes to the nearby accelerating nodes only, rather than having all accelerating nodes within their view and computation. Hence, as we move a considerable distance from the accelerating nodes, their effect reduces and all other nodes outside a range of $D^Q_{\max}$ take the maximal possible value.
Two accelerating nodes Q_j and Q_k are said to be connected to each other if $||Q_j - Q_k|| < Th$, where Th is a threshold governing connectivity. The connections are preferred to be local only, or in other words every accelerating node is connected to a few accelerating nodes around it. Let $D^*(Q_j, Q_k)$ denote the stable value of $D(Q_j, Q_k, i)$ (beyond which any number of updates do not produce any change) which is the optimal distance between the nodes Q_j and Q_k. In case of any change in the environment the maximum number of DP iterations needed to ensure that value of $D(Q_j, Q_k, i)$ is stable or optimal in value is equal to $D^*(Q_j, Q_k)$, as per the modified map. This value has a threshold of $D^Q_{\max}$. Hence the maximum number of iterations needed to determine a loss of connectivity between 2 accelerating nodes or to detect a blockage is $D^Q_{\max}$.
The value $D^Q_{\max}$ is chosen such that it is small enough to detect any change in the environment reasonably early and large enough to imply a high connectivity between nodes (which results in the production of better alternative paths in case of a blockage). In this paper with uniformly distributed accelerating nodes this value is chosen to facilitate connectivity to the 2nd accelerating node. Alternative methods may be formulated as well. Accelerating node connectivity is shown by the circle for one accelerating node in Figure 2.

4. Dynamic Environment Changes

In a static environment the algorithm is a simple DP solution to the problem. The novelty in the use of accelerating nodes lies in the scenario when the map changes in a manner resulting in a change in the optimal path due to blockage. Consider Figure 3. The light region is the obstacle that suddenly appears in the map. As a result the optimal path when the robot is at position A suddenly changes. A similar scenario would occur if a moving obstacle causes a blockage. Now we need to use the information stored in the accelerating nodes for computation of a strategy to overcome this problem.

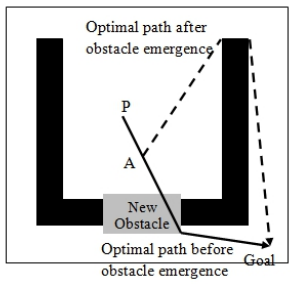

Figure 3. Map showing change in optimal path due to sudden environment change.

Figure 4 depicts the overall procedure of working in the case of dynamic environments. The first step is to compute the initial values of the DP ($D(V,Q_j,i)$ and $D(V,G,i)$) and the successor ($S(V,Q_j,i)$ and $S(V,G,i)$) of all nodes V. Then the robot starts its journey and keeps moving at every instant of time based on the successor of the node. Let $R(t)$ be the position of the robot at time t. The robot moves towards the node $S(R(t),G,i)$ where the motion continues. The robot terminates if $S(R(t),G,i)$ is $null$ denoting reaching the target (G). As the robot moves, the DP iterations continue (equation 4, 5, 6, 7, 10, 11, and 12) wherein a change in values is observed in the case of a change of the map $\zeta^{\text{free}}(t)$. Each accelerating node (Q_j) monitors for any loss of connectivity with other neighboring accelerating neurons. If a blockage is detected ($Blockage(Q_j, Q_k)$), an alternative path ($Path$) is built using graph search which uses the accelerating node graph as the input. After necessary iterations of DP have been reached, the robot terminates its motion along the alternative path and continues along the DP path. The algorithm ends when the target is reached. Each part of this concept is discussed in the next subsections.

4.1. Monitoring Change

The first requirement is to monitor the changes in the map that may cause a blockage between any segments of the map. Monitoring of the map is performed by the accelerating nodes. Upon detection of a blockage alternative path construction takes place. Only the segments in the way of the current path (τ) are of interest. Each node V of the map knows the distances from itself to the target as well as the accelerating nodes. Consider any two accelerating nodes Q_j and Q_k. The distance between these accelerating nodes $D(Q_j, Q_k, i)$ is computed by the DP updates. In the case these nodes are far apart from each other the distance would be $D^Q_{\max}$. Suppose we make a series of J iterations of DP. A blockage between two segments covered by accelerating nodes Q_j and Q_k is identified by a constant increase in $D(Q_j, Q_k, i)$ along with iterations until it reaches a maximum value of $D^Q_{\max}$. Every time equations 10 and 11 are executed in a single DP iteration in case of a blockage, the distance $D(Q_j, Q_k, i)$ is increased. Blockages are checked by using equation 13.

$$Blockage(Q_j, Q_k) = (D(Q_j, Q_k, i+J) - D(Q_j, Q_k, i) > U) \quad (13)$$

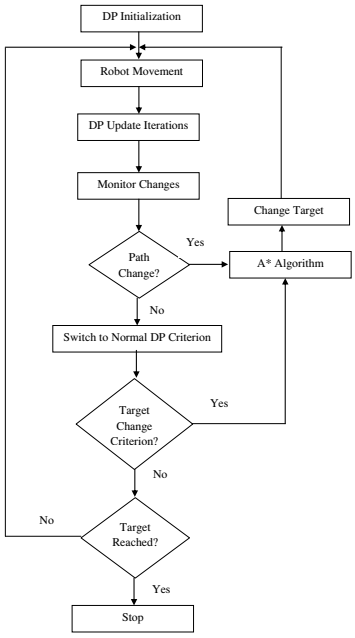

Figure 4. *Algorithm for handling dynamic environment.*

U is a constant that is kept just smaller than J. The factor J needs to be small enough to detect blockages early as a blockage can only be detected after J iterations of DP. However the factor has to be large enough so as not to identify as a blockage a small change in the map which does not actually cause a blockage.
We are only interested in blockages that affect the current path of the robot. The current path (τ) is looked up using the successor values ($S(V,G,i)$). The distance of the current path (τ) from the blocked segment (Q_j, Q_k) is used to decide whether the blockage potentially affects the current path of the robot. In case the path is affected, an alternative path is built and acted upon. In all other cases, no change is performed.

4.2. Graph Search Algorithm

An alternative path is computed with the help of graph search on the map formed by the accelerating nodes. The input graph contains all the accelerating nodes (Q), the present location of the robot ($R(t)$) and the target (G) as vertices ($Vertices = Q \bigcup R(t) \bigcup G$). An edge exists between any two vertices if the distance between them is recorded to be less than $D^Q_{\max}$. The weights of the edges correspond to the distances between them as computed by the DP (equation 14).

$$Edges = (ver_1, ver_2)/D(ver_1, ver_2, i) : D(ver_1, ver_2, i) < D^Q_{\max} \wedge ver_1, ver_2 \in Vertices \quad (14)$$

Here (ver_1,ver_2)/w denotes an edge between vertices ver_1 and ver_2 with weight w.
Edges from one accelerating node are shown in Figure 2 by dark black lines. Uniform Cost Search algorithm is used to compute the shortest path between the source ($R(t)$) and the goal (G). The execution time of the algorithm depends on the number of accelerating nodes. More accelerating nodes ($|Q|$) implies higher execution time. However a higher number of nodes also implies a better path resulting from the graph search. The path returned by uniform cost search is a guiding path that has been built over an abstract map consisting only of accelerating nodes. This path, in a very small time, gives an approximate idea of the optimal path ($\tau^*(t)$) that the robot may follow to reach the goal. By working over an abstract map we are able to generate an approximate path when the optimal path cannot be found instantaneously. This compensates for the time wasted earlier in the computation of the distances propagated by the accelerating nodes.

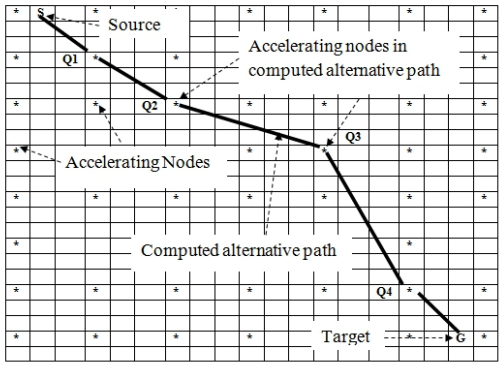

Figure 5. *Generation of an alternative path by graph search algorithm.*

The other major issue is the use of this path generated for the motion of the robot. Consider the path as a set of nodes ($Path = <R(t), Q_1, Q_2, Q_3, \ldots Q_n, G>$). It might however not be possible to travel in a straight line between any two consequent nodes mentioned in the path without collision. Hence the coarser path formulated by a coarser graph needs to be worked out at a finer level. Consider the case that the robot is located at position S given in Figure 5. The path generated by the graph search algorithm is marked with a solid line. We instantaneously change the target of the robot from the actual target to the second point in the path ($Path$) of the robot which is Q_2. In case the path does not have two points, then the last point serves as the target. This may be easily implemented by using the successors $S(R(t),Q_2,i)$ for motion of the robot when it is placed at $R(t)$. The robotic movement continues until the robot has almost reached the current target Q_2. In the present implementation motion continues until a distance of $D^Q_{\max}/2$. As soon as the distance from the robot to Q_2 is less than this amount ($D(R(t),Q_2,i) < D^Q_{\max}/2$), the uniform cost search algorithm is again invoked. The re-invocation accounts for the updated distance measures while the robot moved using the alternative path. In this manner the process continues.
It would be interesting to note why the views of the accelerating nodes are restricted to be local only by keeping a low value of $D^Q_{\max}$ rather than giving them a full global view of the entire map. For this we compare the accelerating nodes to the target node. The target node has a global view as its distances are accurately recorded by the various nodes without trimming down the values after some amount. Now when a node reports some value as its distance from the target ($D(V, G, i)$), we can never be assured of correctness of the value. This is because it may be possible that this value was achieved by some path that passed through a segment that is now blocked by an obstacle. Let I be the number of DP iterations after a potential change of map. Any node V with $D^*(V, G) \leq I$ is guaranteed to have correct value after I iterations, while the other nodes may be storing incorrect values not accounting for a changed map. I is a reasonably low number in case the blockage has happened recently. Now consider the accelerating nodes. They

only store local information. Since the number $D^Q_{\max}$ is small, the connectivity information about nodes may be relied on after $D^Q_{\max}$ iterations at the maximum. This ensures that on detecting a blockage, the input graph to the graph search algorithm is correctly incorporating all changes which have taken place earlier than $D^Q_{\max}$ DP updates.

4.3. Switch to Normal DP

In Sections 4.1 and 4.2 we discussed detecting blockages and thereafter taking an alternative path to reach the destination. The alternative path is however sub-optimal. After some DP iterations all nodes have optimal values of distances ($D(V, G, i)$) and successors ($S(V, G, i)$). It is hence advisable to discontinue a sub-optimal alternative path ($Path$) and continue with the path as per the updated values of the successors ($S(V, G, i)$). The important issue is to decide the time when the revocation of the DP may be done considering all the distances updated as per the changed map.

Let $nodes(Path)$ denote the number of nodes in the built alternative path and $||Path||$denote the path length. Let the revocation take place at a time t_2 after I iterations of DP. Let t be the time the blockage was detected. Since $Path$ is a sub-optimal path to reach destination as per the modified map, $||Path|| \geq ||\tau^*(t)||$. Since the robot moves towards the goal $||\tau^*(t)|| \geq ||\tau^*(t_2)||$, which means $||Path|| \geq ||\tau^*(t_2)||$. Considering all weights greater than or equal to unity we have $||\tau^*(t_2)|| \geq nodes(\tau^*(t_2))$, or $||Path|| \geq nodes(\tau^*(t_2))$. The number of DP updates needed when the robot is at $R(t_2)$ is $nodes(\tau^*(t_2))$, which is always less than the alternative path length ($||Path||$). Hence after a total of $||Path||$DP iterations (a computed quantity) the revocation may safely be initiated.

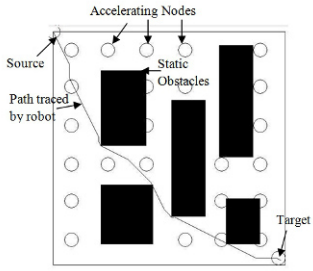

Figure 6. Map and path traced by robot in case of static environment.

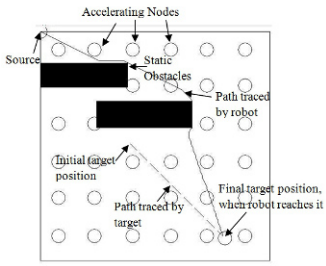

Figure 7. Map and path traced by robot in case of moving target.

5. Results

The algorithm was implemented using a simulation engine implemented in JAVA. The simulation engine used JAVA Applets for the display of the map, obstacles and the robot movement. The map was given as a JPEG image. Sudden obstacle changes were simulated by giving multiple images to the simulation agent and loading the needed image at a given time. The algorithm was implemented as a separate module in the simulation. This implemented the DP with all accelerating nodes. All simulations used maps of size 100×100 pixels. The source is the top left corner of the map and the destination is the bottom right corner. The simulations were carried on a 3.0 3.0 Core 2 Duo processor with 4 GB RAM.

The entire algorithm has many parameters that we analyze one by one. These include the number of accelerating nodes, number of iterations of the update cycle, and maximum distance of accelerating nodes. We even discuss the trade-off in time and path length in the next subsections.

5.1. Path Optimality

We first study the optimality of the path for various maps. We provided various maps and used the stated algorithm to solve the problem. In all the test cases that we supplied, the algorithm was able to generate optimal paths. In the first case we took a static map. There was no movement of the target or obstacles. This was done to see the path finding capability of the algorithm in static conditions. The map and the path generated by the algorithm are shown in Figure 6. The accelerating nodes are shown by circles in the map. Here it can easily be seen that the algorithm gave an optimal path as the final result. An initial setup phase occurred in the algorithm before the robot could start moving. During this time the target propagated the distances to the various nodes in the map. Thereafter the robot started moving. After the start of the robotic movement, update cycles of the DP continued. However no change was made in these updates as the environment was completely static.

In the second case we moved the target as the robot moved in the map. We know that the robot can always catch the target if the speed of motion of the robot is greater than that of the motion of the target. Hence the simulation was run keeping the speed of the target half of that of the robot. At every instance of time the robot tried to chase the target. While doing so the robot also avoided the obstacles that appeared in its path. The target moved in a straight line that was predefined. The simulation result for this problem has been given in Figure 7. The circles denote the accelerating nodes. The motion of the target has been shown by the dashed line. It can clearly be seen that the robot was not only able to catch the target, but the entire path length was also optimal. During the course of the run, the value of the DP changed multiple times. However no blockage was detected by the algorithm.

The next experiment analyzed the behavior in case of moving obstacles. Obstacles were moved while the robot was traveling from the source to the goal. The robot was now supposed to reach the target and at the same time avoid the moving obstacles. The avoidance of obstacles was difficult because of the fact that they came between the source and the target in the path of the robot. We gave two kinds of motions and maps for the simulation purposes. In the first simulation we observed that the robot was able to catch the target using an optimal path, and at the same time avoid the obstacles. However no blockage was detected. The obstacles moved at half the speed of the robot. This is shown in Figure 8(a), (b), (c) and (d) at timestamps of 20, 120, 158 and 174 respectively. The initial position of the moving

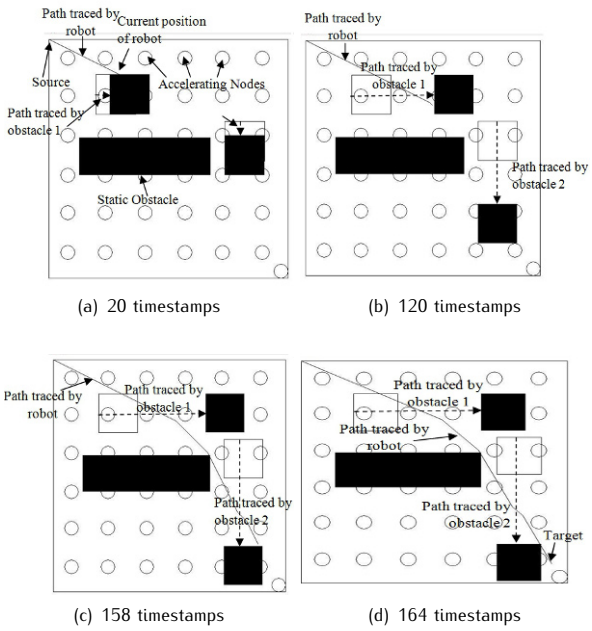

Figure 8. Map and path traced by robot in case of moving obstacle after (a) 20 timestamps, (b) 120 timestamps, (c) 158 timestamps, (d) 164 timestamps.

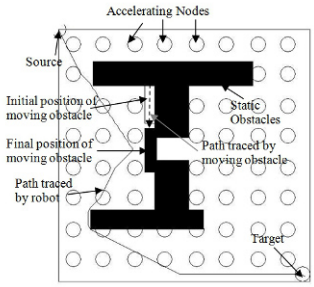

Figure 9. Map and path traced by robot in case of moving obstacle causing path blockage.

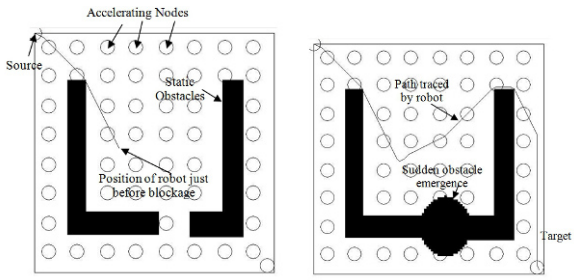

Figure 10. Map and path traced by robot in case of sudden obstacle emergence causing path blockage (a) before obstacle emergence (b) at the end of journey.

obstacle is shown by empty boxes and instantaneous position by filled boxes. It may be easily noted that the robot showed behaviors of normal human walk. The robot arrived at the top of the first obstacle when the destination was on the other side. These facts caused it to wander around for some time as the obstacle moved as shown in Figure 8(a). The obstacle moved rightward and the robot struggled for a diagonal movement at the 120th timestamp given in Figure 8(b). The obstacle was completely passed at timestamp 158 as shown in Figure 8(c). Then the robot could easily rush through to find the second obstacle which it passed in a similar manner as given in Figure 8(c). The final positions of the robot, obstacles and path are given in Figure 8(d).

This second path witnessed a path blockage, and the accelerating nodes were put to use. The robot now completely changed direction in the middle of its journey and used some other path when the blockage happened. This case is shown in Figure 9. It may be easily noticed that the accelerating nodes helped in early detection of the blockage and hence the robot did not go very close to the obstacle that caused the blockage. The obstacle traveled downwards as the robot moved. When the robot was very near to the turn it made (as shown in Figure 9), blockage occurred which was quickly detected. The robot hence took the alternate path.

The most innovative part of the algorithm is its capability of handling the sudden occurrence of an obstacle that causes a blockage of the path from the source to the destination. As a result of this change, the complete path of the robot needs to be re-calculated in real time. The accelerating nodes are again useful in detecting the blockage and optimally moving the robot using good moves until the actual path is re-calculated by the DP algorithm. Here we took a map and made the robot move. As soon as the robot was somewhere in the middle of the journey, an obstacle was placed that caused its path to become completely blocked. The original map and part of the path traced is given in Figure 10(a). At this instance sudden obstacle occurrence took place which blocked the path. The path thereafter is given in Figure 10(b). It may be seen that accelerating nodes helped as the robot did not go very close to the obstacle.

5.2. Algorithmic Analysis

One of the major parameters of the algorithm that largely contributes towards the algorithmic novelty is the total number of accelerating nodes ($|Q|$) in the algorithmic execution. In this section we try to analyze and verify the effect of increasing or decreasing the number of accelerating nodes. A high number of nodes in the map has a multiplicative increase in the execution time of the algorithm. This is because of the multiple processing of the distances propagated by each and every accelerating node. Further a high number of nodes mean a greater time of execution in building of an alternative path by graph search. Since all this happens in real time, the effects may be adverse. The robot would have to traverse slowly as well as wait for the alternative path to build in case of a blockage. The advantage of having a larger number of accelerating nodes is in optimality of the alternative path generation. The larger the number of accelerating nodes, the better the path returned by the graph search algorithm due to more information being available from the various nodes. This would have two benefits. Firstly the algorithm would be able to account for small obstacles and blockages in the map that affect the path. Secondly the path of the graph search algorithm would be close to the optimal path. Hence until the algorithm

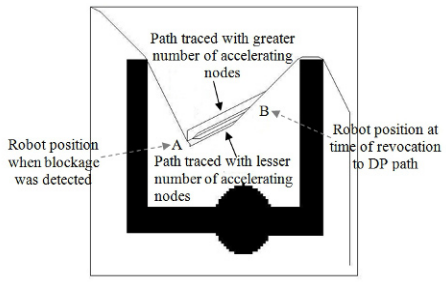

Figure 11. *Path traced by robot for increasing number of accelerating nodes.*

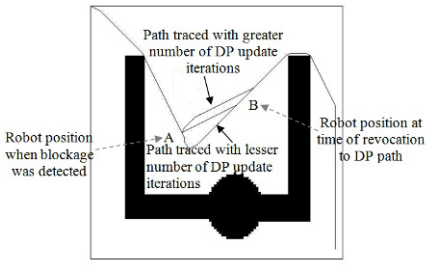

Figure 13. *Path traced by robot for increasing number of dynamic programming update iterations.*

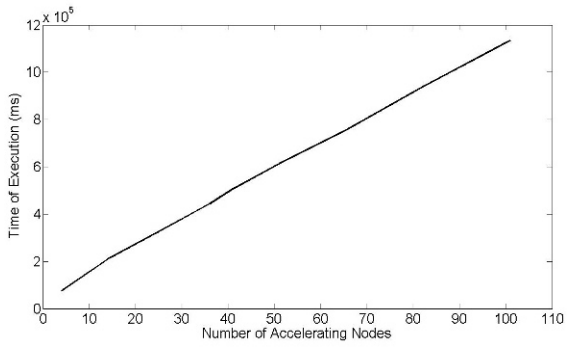

Figure 12. *Effect of number of accelerating nodes on time of execution.*

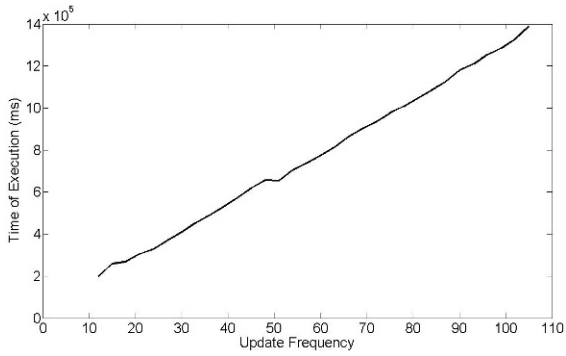

Figure 14. *Effect of dynamic programming number of update iterations on time of execution.*

builds the distance values by DP iterations, the robot would be traveling by a path which very closely resembles the optimal path of the modified map.

Figure 11 shows the paths for various numbers of accelerating nodes for the case of the sudden appearance of an obstacle. It may be easily seen that in all cases the robot moved from the initial point to the point A using the target DP values. As soon as the point A was reached, the information regarding the sudden blockage was received. This is the time that the alternative path was computed and followed. The closeness of the computed alternative path to the optimal path decreased as we decreased the number of accelerating nodes. This can be easily seen from the various paths in Figure 11. Point B in the same figure denotes when the algorithm reverts back to the normal DP algorithm. The corresponding execution times used by the robot are shown in Figure 12. The time almost linearly increases with the increase in the number of accelerating nodes. By examining Figures 11 and 12 we can state that the trade-off for this parameter is that a higher number of accelerating nodes implies better paths (desirable) with higher computational cost (non-desirable), and vice versa.

Another important factor is the number of updates of DP carried out by the system between every two consecutive moves of the robot. In practice this parameter would depend on the natural speed of the robot. The robot would continue moving at its natural speed along the guided path and the time to travel in between the nodes would be the time given to the system to make all computations. This determines the number of iterations of DP update that the system can perform. A high speed robot would mean a low number of iterations and vice versa.

We further analyze the contribution of this parameter to the overall path of the robot and the corresponding execution times. It is natural that the increased number of iterations in between robot moves would mean an increase in computation time. This is because of the extra number of iterations or updates. The advantage would however be the decrease in time in which information regarding the blockage takes to reach the nodes. A larger update frequency would result in more flow of information per move and as a result the information regarding blockages would be readily available before the robot makes many moves. Similarly, the larger update frequency would allow the robot to catch up with the conventional DP algorithm reasonably early and continue its journey for the rest of the path.

Figure 13 shows the path traced by the robot for various update frequencies. An early deviation is found in the cases with a larger update frequency compared to ones with a smaller update frequency. This is due to the early information availability regarding blockages. The corresponding time of execution is given in Figure 14. By examining Figures 13 and 14 we can state that the trade-off for this parameter is that a higher update frequency implies better paths (desirable) with higher computational cost (non-desirable), and vice versa.

The factor $D^{Q}_{\max}$ storing the vision of every accelerating node is a rather passive parameter in terms of computational time. The value of this parameter does not appreciably affect the computational cost of the algorithm since all the computations have to be made. This factor mainly affects the graph search algorithm and its result. A very low value of this parameter would result in no connectivity of the accelerating node graph. This would eliminate any blockage detection or alternative

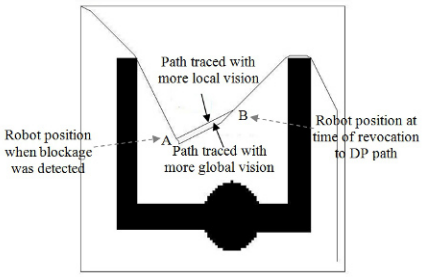

Figure 15. Path traced by robot for increasing vision of accelerating nodes (or increasing factor $D^{Q}_{\max}$).

path computation. For execution of the algorithm, some connectivity is needed between the accelerating nodes. A medium value of this parameter which is sufficient to establish the connectivity of every node to its neighbors would mark a local vision for the graph. The robot in case of blockage can select the alternative path. Very large values of this parameter would imply a global vision. Here the major problem would be that every node does not know whether the connectivity reported by the accelerating nodes account for the blockage or not. Since they are now allowed to report even large values, it may be possible that the blockage has not yet been updated and the graph search algorithm works over wrong inputs. This would result in the algorithm working like conventional the DP algorithm since connectivity is guaranteed by some pair of nodes. These two conditions are shown in Figure 15.

6. Conclusions

In this paper we presented a simple and powerful algorithm to compute the optimal path from the source to the destination in the presence of static and dynamic obstacles. The novelty of the algorithm was its real time nature in detecting and acting on the path blockages which is a case not appreciably discussed in the literature so far. The algorithm gives good performance in the presence of complex maps with high resolutions and the occurrence of a sudden or gradual blockage in the path being followed by the robot. Our algorithm can detect the blockage very early which saves it from continuing its motion in the wrong path. Further, the robot is not asked to stand stationary or explore the wilderness for the rest of the path. It is rather guided by a path that closely resembles the optimal path in the modified map. As soon as the optimal path is computed by DP, the robot may continue its movement using the optimal strategy.

The functionality of sensing a blockage and building up an alternative path was implemented with the help of special nodes called accelerating nodes. These nodes performed the additional task of computing their distances to the other nodes. This was done using the same mechanism of DP that is referred to as the distance propagation mechanism. This additional computation helped in sensing changes as well as taking the needed actions. The accelerating nodes were modeled as a connected graph with weighted edges. A detected change in connectivity of this graph corresponded to a blockage of the robotic path. When any sort of blockage was reported by the accelerating nodes, the graph search algorithm was used to compute an abstract path comprising only of the accelerating nodes. This path was followed until the actual path was re-computed by the DP.

We analyzed the working of the algorithm in several cases and found our algorithm to be optimal and real-time in all experimented cases. An analysis of the various algorithmic parameters was also conducted. We studied how each one of these contributed to effective reporting of the blockage, optimality of the alternative path, and other elements of the real-time nature of the algorithm. We further studied the execution time of the algorithm in each of these cases. We could easily see that best performances were obtained using parameter settings which also implied high computational cost. This emphasizes the trade-off between path optimality and execution time for an optimal parameter setting.

The presented algorithm effectively solves the problem for experimented scenarios. It may however be augmented by the study of obstacle dynamics in case of moving obstacles. Obstacle motion may be extrapolated for early blockage predictions. Furthermore, a closer study of the algorithmic parameters may be conducted. These parameters may be made adaptive in future to automatically adjust for each map and past behavior of the robot in the map. The results so far have been simulation only. This implementation may also be used by real robots in dynamic environments.

References

[1] A. R. Willms, S. X. Yang, An Efficient Dynamic System for Real-Time Robot-Path Planning, IEEE Transactions on Systems Man and Cybernetics – Part B Cybernetics, 36(4) (2006), 755-766

[2] S. X. Yang, M. Meng, An efficient neural network approach to dynamic robot motion planning, Neural Networks, 13(2)(2000), 143-148.

[3] A. Zelinsky, Using path transforms to guide the search for findpath in 2d, International Journal of Robotic Research, 13(4)(1994), 315–325

[4] D. V. Lebedev, J. J. Steil, H. J. Ritter, The dynamic wave expansion neural network model for robot motion planning in time-varying environments, Neural Networks, 18(3), 267-285.

[5] A. Shukla, R. Tiwari, R. Kala, Mobile Robot Navigation Control in Moving Obstacle Environment using A* Algorithm, In Proceedings of the International Conference on Artificial Neural Networks in Engineering Vol. 18, ASME Publications, 2008, 113-120.

[6] S. Kambhampati, L. Davis, Multiresolution path planning for mobile robots, IEEE Journal of Robotics and Automation, 2(3)(1986), 135-145.

[7] J. Y. Hwang, J. S. Kim, S. S. Lim, K. H. Park, A Fast Path Planning by Path Graph Optimization, IEEE Transaction on Systems, Man, and Cybernetics—Part A: Systems and Humans, 33(1)(2003), 121-128.

[8] A. Yahja, A. Stentz, S. Singh, B. L. Brumitt, Framed-quadtree path planning for mobile robots operating in sparse environments, In Proceedings of the 1998 IEEE International Conference on Robotics and Automation, 1998, 650-655.

[9] C. Urdiales, A. Bantlera, F. Arrebola, F. Sandoval, Multi-level path planning algorithm for autonomous robots, IEEE Electronic Letters, 34(2) (1998), 223-224.

[10] R. Kala, A. Shukla, R. Tiwari, Robotic path planning in static environment using hierarchical multi-neuron heuristic search and probability based fitness, Neurocomputing, 74(14-15) (2011), 2314-2335.

[11] A. Shukla, R. Kala, Multi-Neuron Heuristic Search, International Journal of Computer Science and Network Security, 8(6) (2008), 344-350.

[12] R. Kala, A. Shukla, R. Tiwari, Robotic Path Planning using Multi-

Neuron Heuristic Search, In Proceedings of the 4th International Conference on Computer Sciences and Convergence Information Technology, 2009, 1318-1323.
[13] Y. Lu, X. Huo, O. Arslan, P. Tsiotras, Incremental Multi-Scale Search Algorithm for Dynamic Path Planning With Low Worst-Case Complexity, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 41(6)(2011), 1556-1570.
[14] S. Koenig, M. Likhachev, and D. Furcy, Lifelong planning A∗, Artificial Intelligence, 155(1-2) (2004), 93–146.
[15] A. Stentz, Optimal and efficient path planning for partially-known environments, In Proceedings of the 1994 IEEE International Conference on Robotics and Automation, 1994, 3310-3317.
[16] D. Cagigas, J. Abascal, A Hierarchical Extension of the D* Algorithm, Journal of Intelligent and Robotic Systems, 42(4)(2005), 393–413.
[17] J. J. Kuffner, S. M. LaValle, RRT-connect: An efficient approach to single-query path planning, In Proceedings of the IEEE International Conference on Robotics and Automation vol. 2, 2000, 995-1001.
[18] S. M. LaValle, J. J. Kuffner, Randomized kinodynamic planning, In Proceedings of the IEEE International Conference on Robotics and Automation, 1999, 473–479.
[19] B. Raveh, A. Enosh, D. Halperin, A Little More, a Lot Better: Improving Path Quality by a Path-Merging Algorithm, IEEE Transactions on Robotics, 27(2) (2011), 365-371.
[20] L. Zhang, D. Manocha, An efficient retraction-based RRT planner, In Proceedings of the IEEE International Conference on Robotics and Automation, 2008, 3743-3750.
[21] L. E. Kavraki, M. N. Kolountzakis, J. C. Latombe, Analysis of probabilistic roadmaps for path planning, IEEE Transactions on Robotics and Automation, 14(1) (1998), 166-171.
[22] L. E. Kavraki, P. Svestka, J. C. Latombe, M. H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Transactions on Robotics and Automation, 12(4)(1996), 566-580.
[23] R. Gayle, A. Sud, E. Andersen, S. J. Guy, M. C. Lin, D. Manocha, Interactive Navigation of Heterogeneous Agents Using Adaptive Roadmaps, IEEE Transactions on Visualization and Computer Graphics, 15(1)(2009), 34-48.
[24] R. Bohlin, L. E. Kavraki, Path Planning Using Lazy PRM, In Proceedings of the 2000 IEEE International Conference on Robotics and Automation, 2000, 521-528.
[25] C. M. Clark, Probabilistic Road Map sampling strategies for multi-robot motion planning, Robotics and Autonomous Systems, 53(2005), 244–264.
[26] K. J. O' Hara, D. B. Walker, T. R. Balch, Physical Path Planning Using a Pervasive Embedded Network, IEEE Trans. Robotics, 24(3)(2008), 741-746.
[27] Z. Yao, K. Gupta. Distributed Roadmaps for Robot Navigation in Sensor Networks, IEEE Transactions on Robotics, 27(5)(2011), 997-1004.