Mobile Robots Path Planning With Heuristic Search

Radu Robotin*. Gheorghe Lazea. Petru Dobra. ****

**Technical University of Cluj-Napoca, Deprtment of Automation*
Cluj-Napoca, Romania (Tel: 40 264 401267; e-mail: radu.robotin@ aut.utcluj.ro).
*** Technical University of Cluj-Napoca, Deprtment of Automation*
Cluj-Napoca, Romania (Tel: 40 264 401239; e-mail: {gheorghe.lazea, petru.dobra}@aut.utcluj.ro)

Abstract: Mobile robots often operate in domains that are incompletely known. This article adresses the goal-directed navigation problem in unknown terrain where a mobile robot has to move from its current configuration to given goal configuration. We will discuss a series of tests performed with various implementations of graph search algorithms (A*, D*, focused D*) as path planners for a mobile robot, focusing on the inherent strengths and pitfalls specific to each implementation.

Keywords: mobile robot, graph search, path planning, heuristic search, comparative tests

1. INTRODUCTION

The problem of determining the optimum path occurs in a number of applications; ranging from finding the fastest path in a network, to determining the safest path for a mobile vehicle wandering on an outer-space surface. The present research we will limit its areas of investigation to finding paths in Euclidean two-dimensional space. This article presents the goal-directed navigation problem in unknown terrains where a mobile robot has to move from its current configuration to a given goal configuration. The issue benefits from a special attention in the field, robotics researchers have proposed various navigation strategies, including the well-known bug algorithms.

The idea behind the approach is that the robot always plans a shortest path from its current coordinates to the goal coordinates, operating under the assumption that the unknown terrain is traversable. Moreover, the robot may factor in the initial environmental information when available. When following its path, the robot incorporates the information on present obstacles inserting them into its map; the procedure is repeated until it eventually reaches the goal configuration or cannot find any traversable path.

This navigation strategy is an example of sensor-based motion planning. According to Berg (2006), if the navigation problem is modeled as a graph-traversal problem on an eight-connected grid with edges that are either traversable or un-traversable, it must terminate because the robot either follows the planned path to the goal vertex or increases its knowledge about the true edge costs, which can happen only once for each edge. More specifically, the present research investigates the case of finding the optimum path for a mobile robot moving along a flat surface; the robot's configurations in the configuration space being the graph's nodes while the graph's arcs represent the cost of moving from one configuration to another.

In the recent years, the specialists in the field have tried to come up with innovative navigation technologies. With the development of path finding, several new classical routing algorithms have been introduced to generate improved routing solutions. For example, Eklund et al. (1996) use the Dijkstra algorithm as a path planner, one of the most famous routing algorithms, which evaluates the moving cost from one node to any other node and sets the shortest moving cost as the connecting cost of two nodes. Concurrently, Best-First-search algorithm gains popularity in the field.

LaValle (2006) presents a slightly different path-planner solution compared to Dijkstra's algorithm; the Best-First-search algorithm proposes a different approach estimating the distance from current position to goal position, selecting the node closer to the goal position. The emerging developments associated with the new path finding situations imposed rapid improvements of the old path finding algorithm due to the newly introduced requirements.

In the late '70's the artificial intelligence community introduces the A* algorithm - a new path finding algorithm. The A* algorithm tries to combine the advantages offered by Dijkstra algorithm and Best-First-Search algorithm.

In the light of the above mentioned conceptual developments, the present paper discusses a series of tests performed with various implementations of graph search algorithms (A*, D*, focused D*) as path planners for a mobile robot, focused on the strengths and pitfalls inherent to every implementation.

2. GRAPH SEARCH AS PATH PLANNERS FOR MOBILE ROBOTS

2.1 General graph search

The search process in a graph can be described as applying a set of operators to the graph's nodes until the goal node is found. The process usually initiates in the goal node and then moves to the successors of the node.
The procedure previously described doesn't specify the order in which the successors of a node should be selected for further explorations. The way a node, *n*, is selected for exploration, determines the overall behavior of the search

algorithm and the resulting path to the goal. For example, if the nodes are selected for expanding in the order in which they are generated, the search is performed in a "breadth-first" fashion. On the other hand, if the most recent generated successor is selected for expanding, then the algorithm performs a "depth-first" search. These search types are not influenced by the selection criteria of the successors of a node or by the position of the goal node in the searched graph, as they perform a blind search. In order to implement the navigation strategy, the robot needs to replan the shortest path from its current vertex to the goal vertex whenever detects that its current path is untraversable. Brummitt and Stentz (1998) suggested that the robot could use conventional graph-search methods; in most of the cases this method proves to be inefficient since most edge costs do not change between replanning episodes.

An equally important issue from the planner's perspective, beside the search strategy, is the the representation of the robot's free space. Bonet and Geffner (2004) discuss a number of methods for discretizing continuous terrain, all of which attempt to balance the inherent trade-offs between two conflicting criteria's, namely the path planning runtime and the length of the resulting path. Visibility graphs contain the start vertex, the goal vertex and the corners of all blocked cells. The shortest paths on visibility graphs are also the shortest paths in the continuous terrain. The path planning is slow on large visibility graphs since the number of edges can be quadratic in the number of cells and the runtime complexity of the search algorithm remains linear with the number of cells. On the other hand, Koening and Likhachev (2002) showed that path planning is faster on grids than visibility graphs, since the number of edges is linear in the number of cells. However, paths formed by grid edges can be sub-optimal and unrealistic looking since the possible headings are artificially constrained.

2.2 The A Algorithm*

The A* algorithm introduced by Nilson, consecutively refined by Buckland, (2002) and Goldberg et. al. (2005) uses a specific evaluation function that minimizes the number of visited nodes during search. The algorithm returns the minimum cost path between the start node and the goal node. The evaluation function, $\hat{f}$, is defined in such a fashion so that its value, $\hat{f}(n)$, for any node, *n*, is an estimate of the minimum cost path passing through *n*. This estimate is computed as a sum between an estimate of the minimum cost path from the start node to node *n*, and the estimate of the minimum cost path from node *n* to goal:

$$f(n) = g(n) + h(n) \tag{1}$$

It is necessary the evaluation function, $\hat{f}$, to be an estimate of the function *f*, so that, let $\hat{g}$ be an estimate of *g* and $\hat{h}$ be an estimate of *h*. The evaluation function is:

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \tag{2}$$

The value of $\hat{g}(n)$ can be easily computed by adding the arc costs on the path from the start node, *s*, to node *n*. Finding an expression for $\hat{h}(n)$ is not an easy task. Information contained in the graph must be used along with the proper choice of metric for measuring distances. If $\hat{h}(n)$ is an optimistic estimate of *h(n)*, $\left(\hat{h}(n) \le h(n)\right)$, then A* will find the minimum cost path and the algorithm is ***admissible*** (it always finds the minimum cost path from the start node to the goal node).

Algorithm 1. A*

```
use an OPEN list to store all the partial expanded paths
place the start node in OPEN list
repeat
  look at the first path in list
  if reaches_goal then
    SUCCESS
  else
    remove the first path in the list
    expand the last node in the path
    compute the cost of newly generated paths an place these paths
in OPEN list
    sort the OPEN list using estimated cost to target plus the path
cost
    if more than a path reaches a node then
      keep only the minimum cost path to that node
    end if
  end if
until goal_found or OPEN list is empty
if goal_found then
  return optimal path
else
  return FAILURE
end if
```

2.3 The D Algorithm*

If a planner is based on A*, the affected nodes and corresponding arcs must be updated in the graph that is used for storing the map, before the search and navigation process continues. There are some inherent limitations of the approach determined by the discrepancy between the information in the map and the reality, the changing states during the mobile robot navigation or when dealing with incomplete information. Such approach is based on the following scenario: every time a discrepancy between the data in the map and the data provided by the sensors onboard the mobile robot is found, the planner updates the map followed by a new planning process. The inefficiency of the approach becomes visible especially in the case when the robot is close to the goal state or when large portions of the map have to be recomputed.

Stentz (1995) has proposed an alternative approach by using the D* algorithm. D* starts from the fundamental principles of A* and it can be used to find an optimal path in a graph. Graph nodes represent possible robot locations in the configuration space (states) while the arcs represent the cost of moving from one state to another. Considering a *start* node and a *goal* node in the graph, let the robot current state be denoted by *r*. Every node, *y*, in the graph has a backpointer to its parent node, *x*, denoted by *b(x)=y*. Similar to A* algorithm, when the search process is completed, the path is returned using sequences of backpointers from *goal* to *start*.

The cost of moving the robot form one node, *x*, to another node, *y*, is *c(x,y)*, a positive number.

The D* algorithm uses an OPEN list to propagate arc cost changes and to store sub-optimal paths in the graph. Each node has also attached a tag: *t(x)=NEW* if the node has never been in the OPEN list before, *t(x)=CLOSED* if the node was removed from the OPEN list and *t(x)=OPEN* if the node is currently in the OPEN list.

For each visited node *x*, the algorithm maintains an estimate of the sum of the arc costs from *x* to *goal* given by the path cost function *h(x)*. Given the proper conditions, this estimate is equivalent to the minimal cost from node *x* to *goal* node.
For each node *x* on the OPEN list (i.e., *t(x)=OPEN*), the key function, *k(x)*, is defined to be equal to the minimum of *h(x)* before modification and all values assumed by *h(x)* since node *x* was placed on the OPEN list. The key function classifies a node *x* on the list into one of two types: a RAISE node if *k(x)<h(x)*, and a LOWER node if *k(x)=h(x)*. The algorithm uses RAISE nodes in the OPEN list to propagate information about path cost increases and LOWER nodes in the OPEN list to propagate information about path cost reductions. Similar to A* algorithm, the propagation takes place through repeated removal of nodes from the list. Each time a node is removed from the OPEN list, it is *expanded* to pass cost changes to its neighbours. These neighbours are in turn placed on the OPEN list to continue the process.

Nodes in the OPEN list are sorted by the key function. An important threshold in the functioning of the algorithm is the k_{min} parameter. It is defined as:

$$\forall \boldsymbol{x} \mid \boldsymbol{t}(\boldsymbol{x}) = \boldsymbol{OPEN}, \boldsymbol{k}_{\min} = \min\left(\boldsymbol{k}(\boldsymbol{x})\right) \tag{3}$$

Paths with cost less or equal with k_{min} are optimal, paths with costs greater that k_{min}, may not be optimal. The parameter k_{old} is the value of k_{min} before the last node was extracted from the OPEN list.

The aim is to construct, for each node, a sequence of optimal paths to goal. The algorithm consists of two functions: *Modify-Cost* and *Process-State*. The *Modify-Cost* function has the role of changing the arc costs as the robot sensorial system discovers new information during the environment exploration and places the affected nodes in the OPEN list. Function *Process-State* computes optimal paths to the goal.
The robot starts following the sequence of backpointers to goal until either reaches the goal configuration or its sensors discover a discrepancy between the information in the map (arc cost changes in the graph do not match sensor measurements) and the environment. In the latter case, the function *Modify-Cost* is automatically called to correct the arc costs and place the affected nodes in the OPEN list.

2.4 Focussing the D Algorithm*

One of the major pitfalls associated with the D* algorithm the way it propagates cost changes. These changes are propagated to the affected states regardless of their importance to the robot navigation. The aim is to focus the search and the propagation of cost changes to those states that are likely to generate optimal paths to the goal. Similar to A* algorithm, D* can also use a heuristic function for decreasing the number of expanded nodes and search focus.

Let *g(x,r)* be the estimated path cost from robot position, *r*, to the node *x*. This function will be the *focusing heuristic*. Furthermore, a new function, *f*, the *estimated robot path cost* is defined as follows:

$$\boldsymbol{f}(\boldsymbol{x},\boldsymbol{r}) = \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x},\boldsymbol{r}) \tag{4}$$

All the LOWER nodes in the OPEN list will be sorted using function *f()* as a sort key. Function *f()* is the estimated path cost from node *r* to node *goal*, passing through node *x*. Function *f()* will provide the optimum cost path from *r* to *goal*, passing through *x*, if *g()* is satisfying the monotonic restriction, due to the fact that *h(x)* is optimal when a LOWER node is extracted from the OPEN list.

For RAISE nodes, the previous value of function *h()* defines a lower bound on the *h()* value of all the LOWER nodes that can be discovered. Thus, if the same focusing heuristic is used, the previous value of *f()* for the RAISE nodes defines a lower bound for the value of *f()* for all the LOWER nodes that can be discovered. Thus, if the value of *f()* for the LOWER nodes in the OPEN list is larger than the previous value of *f()* for the RAISE nodes, it is useful to expand the RAISE nodes in order to discover more advantageous LOWER nodes.

Using this work hypothesis, the RAISE nodes in the OPEN list should be sorted using the value of the function *f(x,r)* as a sort key, and to avoid infinite loops in the backpointers, ties in this key are to be sorted using the value of *k()*.

The process terminates when the lowest value of *f()* function for all the nodes in the OPEN list is grater or equal to the path cost, since further expanding will not be able to produce a LOWER node with a sufficiently small value of the cost function and located close enough to the current node to influence the search. According to Stentz, (1996), this termination is more drastic and abrupt than in the previous case (D* without the focusing heuristic).

The major problem in using a focusing heuristic is that once an optimal path to the goal has been found, the robot starts following backpointers to the goal state and moves to another node and the problem is that the nodes in the OPEN list are sorted based on the value of the path cost computed for the old robot position and thus the nodes in the OPEN list have incorrect values for the functions *f()* and *g()*. A possible solution is to calculate these functions each time the robot moves or a node is inserted in the OPEN list. Empirical results of Berg (2006), Hansen and Zhou (2007) have shown that this is a major slow-down in the algorithm and the speed-up gained through focusing search is outrun by the slow-down introduced by the recalculation of *f()* and *g()*.

Stentz (1996) demonstrated that is an advantage that usually the robot moves only a few nodes before a re-planning operation is necessary. Thus, the values of *f()* and *g()* functions are only slightly deviated. that a node *x* is placed in the OPEN list at the time the robot is in the configuration indicated by the node r_0 and the value of *f()* is $f(x,r_0)$. If the robot moves to another node, r_1, $f(x,r_1)$ may be computed and the position of node *x* in the OPEN list may be adjusted. On the other hand, to avoid the computational cost, one may compute a lower bound on the value of $f(x,r_1)$:

$$f_L(x,r_1) = f(x,r_0) - g(r_1,r_0) - e \quad (5)$$

Function f_L represents a lower bound on $f(x,r_1)$, since it asumes that the mobile robot has moved in the direction of the node x, thus the cost of $g(r_1,r_0)$ is subtracted. The parameter ε is a positive constant.

States are sorted on the OPEN list by a *biased f()* value, given by $f_B(x,r_i)$, where x is the node in the OPEN list and r_i is the robot's state at the time x was inserted or adjusted on the OPEN list (Stentz, 1996). Let $\{r_0, r_1,\ldots,r_n\}$ be the sequence of nodes occupied by the robot when the nodes were inserted in the OPEN list. The value of $f_B()$ is given by:

$$f_B(x,r_i) = f(x,r_i) + d(r_i,r_0) \quad (6)$$

where $f()$ is the estimated robot path cost given by:

$$f(x,r_i) = h(x) + g(r_i,r_{i-1}) \quad (7)$$

and $d()$ is the *accrued bias* function given by:

$$\begin{cases} d(r_i,r_0) = g(r_1,r_0) + g(r_2,r_1) + \mathbf{K} + g(r_i,r_{i-1}) + e, i > 0 \\ d(r_0,r_0) = 0, i = 0 \end{cases} \quad (8)$$

The function $g(x,y)$ is the focusing heuristic, representing the estimated path cost from a node y to a node x. The nodes in the OPEN list are sorted by increasing $f_B()$ value, with ties in $f_B()$ ordered by increasing $f()$, and ties in $f()$ ordered by increasing $k()$. Ties in $k()$ are ordered arbitrarily. Thus, a vector of values $\langle f_B, f, k\rangle$ is stored with each node in the list.

3. EXPERIMENTAL RESULTS

The present research is based on a number of experiments implemented in both simulation and real life, using the Pioneer2 mobile robot, to determine the advantages and disadvantages of using A* and D*.

Fig. 1 presents a simulated environment having a configuration space similar to the obstacles distribution in the Robotic Research Lab at Technical University of Cluj-Napoca. Each cell in Fig. 1 represents a square area of 15 cm^2. The initial robot configuration is in the lower left corner (green square) while the goal configuration is in the upper right corner (red square). The path generated by the A* algorithm is also presented in Fig. 1. The distances between nodes were measured using the Manhattan metric, so that any neighbouring node on a N, S, E or W direction is at a distance of 1 from the current node, while nodes on NW, NE, SW and SE direction are at a distance of 2 from the current node.

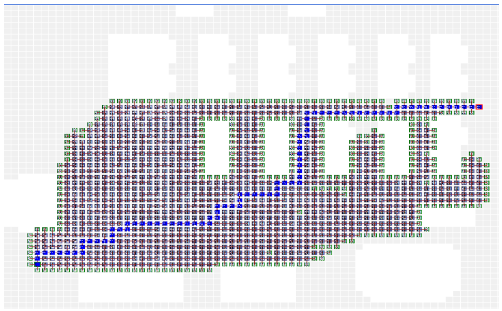

Fig. 1. Path returned by A* algorithm.

Fig. 2 presents the path generated by the D* algorithm without the focusing heuristic, for the same environment configuration as in Fig. 1. Expanded nodes are presented in both Fig. 1 and Fig. 2; nodes depicted with a green rectangle are the nodes in the OPEN list (on the frontier of the area representing the set of expanded nodes) while the nodes on the optimal path are presented in dark blue. Fig.3 presents the path generated by the D* algorithm with the focusing heuristic, for the same environment configuration as in Fig. 1.

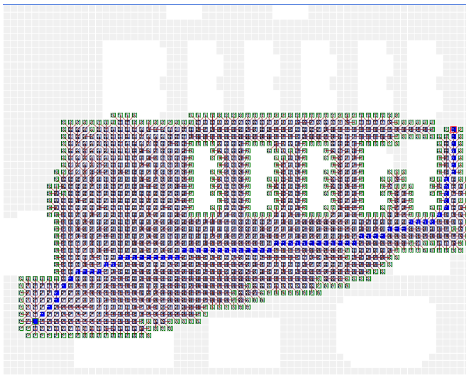

Fig. 2. Path generated by D* algorithm without focusing heuristic.

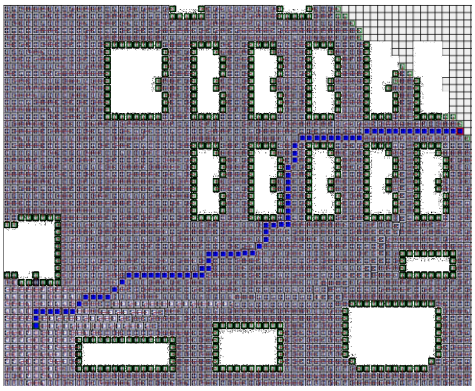

Fig. 3. Path generated by focused D* algorithm.

When analyzing the information covered in Fig. 1 to Fig. 3, there were a number of issues that needed to be addressed, such as: why the resulting path in the three cases is not identical? and more importantly what are the benefits of using D* like algorithms, since, at a first glance, the A* seems to provide optimal results?

When analyzing the path length in the three situations (Fig. 1 to Fig. 3) we need to remember that both A* and focused D* are using a focusing heuristic, while D* is not using it. Moreover, the distances are determined using the Manhattan metric, thus the results may appear different due to aliasing. The path cost (the sum of all arcs) is minimal in the three cases. In addition, focused D* uses three keys to sort the nodes in the OPEN list, ties resulted by using the first sort criterion (value of f_B) are solved by using the value of f while ties in this case are solved using the third sort key, that is the value of k.

The second problem is that both A* and focused D* algorithms expand the same number of nodes (Fig. 1 and Fig. 3), while D* expands a larger number of nodes. In the following we will analyze the situation when the information in the map is incomplete or the structure of the environment

changes. We can assume the robot is equipped with sensors capable of detecting the environment on a radius of 10 nodes around the mobile robot. The robot is supposed to traverse a

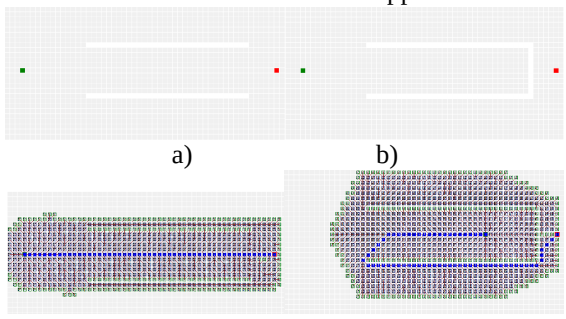

Fig. 4. Example of A* path planning.

corridor that has a door at the end. The robot has no information on the existing door.

Fig. 4. Presents an example of A* path planning: subfigure a) presents the environment as it is known by the planner, while subfigure b) presents the real structure of the environment. Subfigure c) presents the initial path plan (through the closed door) while subfigure d) present the re-planned path, after the robot discovers the closed door.

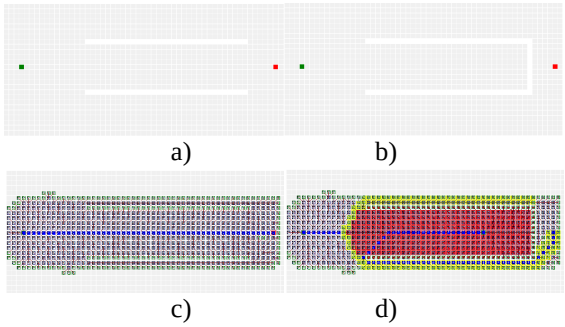

Fig. 5. Example of D* path planning.

Fig.5. Shows an example of D* path planning: subfigure a) presents the environment as it is known by the planner, while subfigure b) presents the real structure of the environment. Subfigure c) presents the initial path plan (through the closed door) while subfigure d) present the re-planned path, after the robot discovers the closed door; nodes in red represent RAISE nodes, while nodes in yellow are LOWER nodes.

The experiments presented in Fig. 4 and in Fig. 5 indicate the different way algorithm A* and focused D* operate when facing the same problem. Both algorithms plan an initial path through the closed door (subfigure c) in both Fig. 4. and Fig. 5.; based on the initial information, this is an unobstructed path. The robot follows backpointers to the goal configuration until the closed door enters the range of the mobile robot sensorial system. At this point, the planner based on A* updates the map and starts a new planning process having the start node the robot current position, while the goal node remains unchanged. On the other hand, the D* algorithm initiates an attempt to repair the map (the affected portion of the map containing the initial path through the closed door). The number of expanded cells is smaller than in the case of A* because the algorithm uses portions of the map that has the nodes unaffected by the cost changes.

3.1 On-line Tests

Fig. 6 represents the navigation of the mobile robot Pioneer 2 in a real-life environment, having the same characteristics as the environment used for simulations. Even if the information stored in map is completely accurate (the algorithm is completely informed), cost changes in arcs are due to a series of external factors such as: localization errors, error in specifying the initial robot position, data errors provided by the sensorial system of the robot and errors of the robot's odometric system.

Several tests have been performed in order to determine the average running time between breadth-first search, Fast A* implementation, D* without focusing heuristic and focused D*. The tests were performed off-line on random generated maze-like maps, represented as eight-connected grid. The maps contain 35% of blocked cells and have adjustable dimensions of 100×100 cells, 1000×1000 cells and 10000× 10000 cells (except for the breadth-first search which was inefficient and the memory requirement were too large for such a high number of cells). Table 1 presents the run-time results (in seconds) while Table 2 presents a comparison between the number of expanded cells for each complete planning-replanning process.

The Open List in A* and D* is implemented as a balanced binary tree sorted on corresponding key values, with tie-breaking mechanism. This tie-breaking mechanism results in the goal state being found on average earlier in the last *f()* value pass.

In addition to the standard Open/Closed Lists, marker arrays are used for answering (in constant time) whether a node is in the Open or Closed List. We use a "lazy-clearing" scheme to avoid having to clear the marker arrays at the beginning of each search. Each path finding search is assigned a unique (increasing) id that is then used to label array entries relevant for the current search. The above optimizations provide an order of magnitude performance improvement over a standard "textbook" A* implementation. All experiments were run on a 2.1 GHz PC under MS Windows XP.

Table 2. Comparison between running time of the breadth-first search, A*, D* and focused D* in planning and re-planning paths

Dimension	Breadth-First	Fast A*	D*	Focused D*
Planning 10^4 cells	35.2s	5.7 s	8.0 s	6.2 s
Re-planning 10^4 cells	12.7s	3.0 s	2.1 s	1.3 s
Planning 10^6 cells	178.9s	37.3 s	55.8 s	50.7 s
Re-planning 10^6 cells	113.2s	28.2 s	10.1 s	7.6 s
Planning 10^8 cells	-	136.4 s	335.0 s	298.7 s
Re-planning 10^8 cells	-	126.8 s	87.4 s	54.3 s

Table 2. Comparison between number of expanded cells.

Dimension	Breadth-First	Fast A*	D*	Focused D*
10^4 cells	625982	9658	15352	1672
10^6 cells	5569854	21566	36254	7625
10^8 cells	-	153694	279125	16369

4. CONCLUSIONS

Although specific to artificial intelligence, the A* and D* demonstrate their impact on any applications requiring graph search, including mobile robotics. This is due to the fact that both A* and D* are generic algorithms, applicable to any optimum path problems.

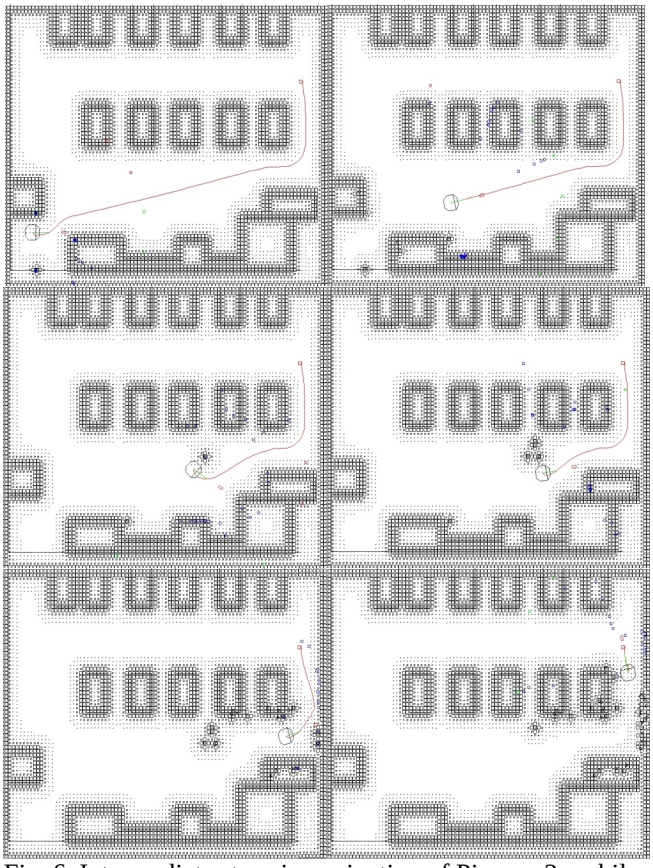

Fig. 6. Intermediate steps in navigation of Pioneer 2 mobile robot with Focused D*.

The A* algorithm is capable of producing optimum paths (lowest cost path) as long as the structure of the environment is completely known (arc costs do not change during robot traverse). In the case where discrepancies exist between the map and the structure of the environment, the efficiency of A* is limited, due to the necessary re-planning operation.
These operations are time consuming since the algorithm is not capable of using information retrieved between searches or the costs of partially expanded nodes, thus any re-planning operation means another planning from with zero information from the previous search. These deficiencies are eliminated by D*. As opposed to A*, D* can cope with arc cost changing during robot traverse. This because the algorithm is capable of using the partially expanded nodes and subsequent path costs leading to smaller wait time between re-planning operations.
The approach is more efficient if the arc cost changes are detected in the close vicinity of the current node (like in the case of a mobile robot equipped with on-board sensorial system). Consequentially, the efficiency of D* in terms of expanded cells during the first stages of the planning process resembles the efficiency of a brute-force planner. In accordance with the results presented in Table 1, D* has the largest wait time before the planning process is completed. This because D* expands the largest number of nodes amongst all three algorithms.
Like in the case of A*, D* can also use a heuristic to focus the search and propagate the cost changes in graph.

ACKNOWLEDGMENT: This paper was supported by the project Progress and development through post-doctoral research and innovation in engineering and applied sciences – PRiDE - Contract no. POSDRU/89/1.5/S/57083", project co-funded from European Social Fund through Sectorial Operational Program Human Resources 2007-2013.

REFERENCES

B. Brumitt and A. Stentz, "GRAMMPS: a generalized mission planner for multiple mobile robots," in *Proceedings of the International Conference on Robotics and Automation*, 1998.

Berg, J.V.D. (2006). Anytime path planning and replanning in dynamic environments, *Proceedings of the International Conference on Robotics and Automation*, pp. 2366-2371.

Bonet, B. and Geffner, H. (2004). Planning as heuristic search, *Artificial Intelligence*, Vol. 129, pp. 5-33.

Buckland, M. (2002). *AI techniques for game programming*, Premier Press, Portland, OR, USA.

Eklund, P.W.; Kirkby, S. & Pollitt, S. (1996). A dynamic multi-source Dijkstra' algorithm for vehicle routing. *In Proc. Of Conf. on Intelligent Information Systems*, Australia and New Zealand.

Goldberg, A.V. & Harrelson, C. (2005). Computing the shortest path: A* search meets graph theory, *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 156-165.

Hansen, E.A. and Zhou, R. (2007). Anytime heuristic search, *Journal of Artificial Intelligence Research (JAIR)*,Vol. 28, pp 267-297.

S. Koenig and M. Likhachev, Incremental A*, in *Advances in Neural Information Processing Systems 14*,. MIT Press, 2002.

LaValle, S.M. (2006). *Planning Algorithms*, Cambridge University Press, Cambridge, U.K.

Stentz, A. (1995). The focussed D* algorithm for real-time replanning, *International Journal of Robotics and Automation*.

Stentz, A. (1996). Optimal and efficient path planning for partially-known environments, in P*roceedings IEEE International Conference on Robotics & Automation*, pp. 3310-3317.