We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

8

Graph search techniques for mobile robot path planning

Radu Robotin, Gheorghe Lazea and Cosmin Marcu
Technical University of Cluj-Napoca
Romania

1. Introduction

In a number of applications, the problem of determining the optimum path occurs. This applications range from finding the fastest path in a network, to determining the safest path for mobile vehicle, wandering on the surface of Mars. In this context, we shall limit our scope to the case of finding paths in Euclidean two-dimensional space. Moreover we shall limit the case study to movements along a surface that can be projected onto a directed graph.

To be specific, we shall look at 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.

Researchers have tried to come with new and better navigation technologies in the last years. With the development of path finding, several new classical routing algorithms have been introduced to generate better routing solution. For example the Dijkstra algorithm is the most famous one, 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 (Eklund et al., 1996). Around the same period of time, Best-First-Search algorithm is also introduced in the researchers' community.

A little different from the Dijkstra algorithm, Best-First-Search algorithm has a different approach because it estimates the distance from current position to goal position, and it chooses the step that is closer to the goal position (LaValle, 2006). The difficulty was growing with the new path finding situations so the old path finding algorithm had to be improved to meet the new introduced requirements.

A new path finding algorithm was introduced and it was named the A* algorithm. The A* algorithm tries to combine the advantages offered by Dijkstra algorithm and Best-First-Search algorithm.

This paper presents tests performed with various implementations of graph search algorithms (A*, D*, focused D*) as path planners for a mobile robot, focused on strong points and drawbacks of each implementation.

2. Graph search as path planners for mobile robots

2.1 General graph search

The search process in a graph can be seen as applying a set of operators to the graph's nodes until the goal node is found. Therefore, there is a general approach, applicable for any graph search, following the constraints:

- Define the start node
- Apply a set of operators to find out the successor nodes. The process is called node *expanding*.
- Each node has a *backpointer* to its parent. These pointers are used to represent paths from the start node to the goal node when the search is complete.
- The successor nodes are checked to see if one of them is the goal node. If the goal node is not found, the expanding continues until the goal is found. Then, the backpointers in every node is used to provide the solution.

A graph search algorithm should contain the following steps:

1. Place the start node in a list called OPEN. This list will contain all the graph's nodes that are candidates for future explorations.
2. Extract a node *n* form the OPEN list and place it in the CLOSED list. The CLOSED list will contain all the already explored nodes.
3. Expand node *n* by generating all its successors. If node *n* has no successors, repeat step 2. If a successor of node *n* is not in the CLOSED list (i.e. the node has not been visited yet), it will be placed at the end of the OPEN list.
4. Check each successor for finding the goal node. If the goal node has been found, the search terminates and the algorithm returns a sequence of backpointers to the goal node. Otherwise, go to step 2.

The above mentioned algorithm does not have any mention about the order in which the successors of a node should be selected for further explorations. The way node *n* is selected in step 2 determines the overall behavior of the search algorithm and 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. None of these types of search is influenced neither by the selection criteria of the successors of a node nor by the position of the goal node in the searched graph, as they perform a blind search.

2.2 Using Heuristic

Blind search in a graph may provide solutions, but, most of the times, because the number of expanded cells grows as the search advances, large running times and high memory consumption occurs.

This deficiency can be corrected if available information about the graph structure and the location of the goal node is used. This information is used to cannel the search effort in the direction of the goal node. Therefore, heuristic rules may be used to focus the search.

A possible use of this information is in the form of an *evaluation function*. The evaluation function, applied to a node, represent a measure of the node capacity to generate a path to

the goal node. The evaluation function allows nodes in the OPEN list to be sorted and allows the selection of the most promising node in generating a path to the goal node.
Several aspects have to be considered in evaluating a path cost: the following techniques are based on minimizing a function that includes the path cost. Path cost may be evaluated by either using path length (measured with an appropriate metric, i.e. Euclidean metric, Manhattan metric etc.) or the number of necessary steps to traverse this path.
Considering two vectors $\mathbf{X}=(x_1, x_2,\ldots, x_n)$ and $\mathbf{Y}=(y_1, y,\ldots, y_n)$, the Minkowski difference or *p-norm* in $\mathbf{R}^n$ may be defined as follows:

$$L_p = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{1}$$

Most of the time, equation (1) is computed for $p=1$, $p=2$ or for $p\to\infty$, which yields to the following metrics:

- $p=1$, resulting the Manhattan distance or L_1 norm:

$$L_1 = \sum_{i=1}^{n} |x_i - y_i| \tag{2}$$

- $p=2$, resulting the Euclidian distance or L_2 norm:

$$L_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{3}$$

- $p\to\infty$, resulting the Chebyshev distance or L_∞ norm:

$$L_\infty = \lim_{k\to\infty} \left(\sum_{i=1}^{k} |x_i - y_i|^k \right)^{\frac{1}{k}} \tag{4}$$

2.3 The A* algorithm

The A* algorithm was proposed by Nilson (Buckland, 2002), (Goldberg & Harrelson, 2005) and it uses a specific evaluation function that, it can be proven, 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.
Let $k(n_i,n_j)$ be the minimum cost of the path between any two arbitrary nodes, n_i and n_j. If the goal node is t then $h(n)$ is the minimum cost of the path from the node n to the goal node:

$$h(n) = k(n,t) \tag{5}$$

If the start node is s, then g is the minimum cost of the path form s to n:

$$g(n) = k(s,n) \tag{6}$$

At this point, the function, *f*, can be defined as the minimum cost path that passes through node *n*:

$$f(n) = g(n) + h(n) \tag{7}$$

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:

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \tag{8}$$

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 called *admissible*.

At this point the A* algorithm can be defined:

1. Place the start node, *s*, in the OPEN list and compute $\hat{f}(s)$;
2. Extract form the OPEN list the node with the minimum value of $\hat{f}$ and place this node in the CLOSED list. Let this node be node *n* (ties are solved arbitrarily, but always favor the goal node);
3. If *n* is the goal node, the algorithm ends and returns a sequence of backpointers to parent nodes to specify the path; otherwise go to step 4;
4. Expand node *n* and compute the value of $\hat{f}$ for each successor of node *n* that is not in the CLOSED list;
5. The computed value for $\hat{f}$ is associated with all the successors of node *n* that is neither in the OPEN list, nor in the CLOSED list. These successors are placed in turn in the OPEN list, each receiving a backpointer to the parent node, *n*;
6. For each successor of node *n* that is in the OPEN list, the evaluation function is updated, as the minimum between the computed value of $\hat{f}$ and the value of $\hat{f}$ computed at the time the node was placed in the OPEN list;

A search algorithm is called admissible if it always finds the minimum cost path from the start node to the goal node. If $\hat{h}$ is an optimistic estimate of *h*, the A* algorithm is admissible (Goldberg & Harrelson, 2005):

Proposition 1:

If $\hat{h}(n) \le h(n)$, there is always a node, $\tilde{n}$, on the minimum cost path, before the algorithm terminates, for which $\hat{f}(\tilde{n}) \le f(s)$.

Let the minimum cost path, *P*, be a sequence $(n_0, n_1, \ldots, n_k)$, with n_0 the start node and n_k the goal node. Let $\tilde{n}$ be the first node in this sequence that is in the OPEN list. The evaluation function for this node is:

$$\hat{f}(\tilde{n}) = \hat{g}(\tilde{n}) + \hat{h}(\tilde{n}) \tag{9}$$

Since $\tilde{n}$ is the first node in the sequence that is in the OPEN list, all the nodes in the sequence *P* prior to this node are in the CLOSED list, meaning that a minimum cost path from *s* to $\tilde{n}$ has been found and $\hat{g}(\tilde{n}) = g(\tilde{n})$, therefore:

$$\hat{f}(\tilde{n}) = g(\tilde{n}) + \hat{h}(\tilde{n}) \tag{10}$$

Since the initial hypothesis was, $\hat{h}(n) \le h(n)$, one can write:

$$\hat{f}(\tilde{n}) \le g(\tilde{n}) + h(\tilde{n}) = f(\tilde{n}) \tag{11}$$

But the value of *f* for any node in the minimum cost sequence is equal to *f(s)*, therefore:

$$\hat{f}(\tilde{n}) \le f(s) \tag{12}$$

Proposition 2:
If for any node, *n*, $\hat{h}(n) \le h(n)$, and if the cost of any arc is larger than a positive value, δ, then A* is admissible. (Bonet & Geffner, 2004).
Let us now consider the three possible cases when A* terminates:

1. The algorithm can only terminate at step 3, case in which the goal node has been found.
2. The algorithm does not terminate. Since all the arc costs are larger or equal to δ, for any node, *n*, located at a distance of *M* steps from the start node, *s*, $M=f(s)/\delta$ and:

$$\hat{f}(n) \ge \hat{g}(n) \ge g(n) > M\delta = f(s) \tag{13}$$

   Moreover, no node located at a distance larger than *M* steps from the start node will be selected for expanding since, according to Proposition 1, and on this minimum cost path there will always be a node, $\tilde{n}$, for which $\hat{f}(\tilde{n}) \le f(s)$, to be selected at step 2 for further expanding.
   An infinite loop may result by inserting repeatedly nodes that are located at a distance smaller than *M* steps in the OPEN list. Since there are a finite number of these nodes, the algorithm must end, because there are a finite number of paths from *s* to *n*.
3. The algorithm ends at the goal node, but the minimum cost path was not found. Assuming the algorithm ends in the goal node, *t*, but not on the minimum cost path, there will be $\hat{f}(t) = \hat{g}(t) > f(s)$. According to Proposition 1, on the minimum cost path there will be a node, $\tilde{n}$, in the OPEN list, for which $\hat{f}(\tilde{n}) \le f(s) < \hat{f}(t)$. This node will be selected for expanding, and thus, the hypothesis that the algorithm terminates is invalid.

The precision of the evaluation function depends on the heuristic involved. If $\hat{h} = 0$ is used for all the nodes, n, this means no heuristic information is used, whatsoever. Though, the algorithm will still find the minimum cost path, since $\hat{h}(n) < h(n)$.

If a restriction on $\hat{h}$ is used, the A* algorithm is optimal in the way that it will never expands more nodes than any other admissible algorithm, less informed[1] than A* (Bonet & Geffner, 2004). The restriction placed on $\hat{h}$ is that the difference between the estimated cost of any two nodes to the goal node should be smaller or equal to the real cost of the path between these two nodes. For any two nodes, n_i and n_j, for which $k(n_i, n_j)$ exists, it is necessary that:

$$\hat{h}(n_i) - \hat{h}(n_j) \leq k(n_i, n_j) \tag{14}$$

Equation (14) is called the consistency hypothesis.

Proposition 3:
In the consistency hypothesis, any node n in the CLOSED list has $\hat{g}(n) = g(n)$.

Let us assume that before placing the node n in the CLOSED list, $\hat{g}(n) > g(n)$, therefore the algorithm did not find the minimum cost path , but, according to Proposition 1, there is a node $\tilde{n}$, on the minimum cost path, in the OPEN list, that has $\hat{g}(\tilde{n}) = g(\tilde{n})$. If $\tilde{n} = n$, Proposition 3 is verified, otherwise one can write:

$$g(n) = g(\tilde{n}) + k(\tilde{n}, n) = \hat{g}(\tilde{n}) + k(\tilde{n}, n) \tag{15}$$

Considering the initial hypothesis in which $\hat{g}(n) > g(n)$:

$$\hat{g}(n) > \hat{g}(\tilde{n}) + k(\tilde{n}, n) \tag{16}$$

And by adding $\hat{h}(n)$ in both members of the equation (16):

$$\hat{g}(n) + \hat{h}(n) > \hat{g}(\tilde{n}) + k(\tilde{n}, n) + \hat{h}(n) \tag{17}$$

and with the consistency hypothesis, one obtains:

$$\hat{g}(n) + \hat{h}(n) > \hat{g}(\tilde{n}) + \hat{h}(\tilde{n}) \tag{18}$$

Thus resulting:

$$\hat{f}(n) > \hat{f}(\tilde{n}) \tag{19}$$

that contradicts the initial assumption that the algorithm has selected node n when $\tilde{n}$ would have been available and confirms the Proposition.

[1] An algorithm X is called more informed than an algorithm Y if the heuristic used by X to compute the estimate of *h(n)* is always larger than the one used by algorithm Y

Proposition 4:
For any node n in the CLOSED list, if $\hat{h}$ is an optimistic[2] estimate of h, then $\hat{f}(n) \le f(s)$.

If node n is the goal node, then $\hat{f}(n) = f(s)$ and the Proposition 4 is verified. If n is not the goal node, then right before the insertion of this node in the CLOSED list, according to Proposition 1, there is another node $\tilde{n}$ on the minimum cost path, for which $\hat{f}(\tilde{n}) < f(s)$. If $\tilde{n} = n$ the Proposition is verified, otherwise if the algorithm has chosen the node n over the node $\tilde{n}$, this means that:

$$\hat{f}(n) \le \hat{f}(\tilde{n}) \le f(s) \tag{20}$$

The next Proposition proves the optimality of A*.

Proposition 5:
If A* is more informed than any other admissible algorithm B, and if the consistency hypothesis is verified, then any node n that will be expanded by A* will be expanded by B also.
Let us assume by Reductio ad Absurdum that the node n is expanded by the A* algorithm, but not by the B algorithm. This implies that algorithm B has been using an information that a path cost through n would have been larger or equal than the cost of the minimum cost path, that is: $f(n) > f(s)$.

```
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
```

Algorithm 1. The A* Algorithm

[2] Optimistic estimate means the cost value of h is underestimated, that is $\hat{h}(n) < h(n)$

The effective minimum cost path through n is:

$$f(n) = g(n) + h(n) \tag{21}$$

Or:

$$h(n) = f(n) - g(n) \tag{22}$$

And in the hypothesis that $f(n)>f(s)$:

$$h(n) \geq f(s) - g(n) \tag{23}$$

Algorithm B could have use a lower bound on the estimate, that is $\hat{h}(n) = f(s) - g(n)$, while A* uses $f(n)=g(n)+h(n)$. Proposition 4 states that $\hat{f}(n) \leq f(s)$, so that:

$$\hat{g}(n) + \hat{h}(n) \leq f(s) \tag{24}$$

Equation (24) means that the estimate used by A* satisfy the constraint:

$$\hat{h}(n) \leq f(s) - \hat{g}(n) \tag{25}$$

Based on Proposition 3, $\hat{g}(n) = g(n)$, therefore:

$$\hat{h}(n) \leq f(s) - g(n) \tag{26}$$

Equation (26) proves that for a node n, algorithm B has used information allowing to lower the bound of h to a value at least equal to the one that A* is using, contradicting thus the initial assumption and demonstrating the proposition.

2.4 The D* algorithm

The A* algorithm can produce optimal paths from a start node to a goal node. A planner for a mobile robot is facing several discrepancies between the information stored in the map and the information from the environment, available by means of the sensors onboard the mobile robot.

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. The approach can be rather inefficient, especially when the information in the map does not reflect the reality or in the case when states change during the mobile robot navigation or when dealing with incomplete information. Such an 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 is 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 (Stentz, 1995) has proposed a different approach: D* algorithm. D* starts from the fundamental ideas 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*. Just like in the case of 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. Just like in the case of A* algorithm, the propagation takes place through the 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 neighbors. These neighbors 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:

$$\forall x \mid t(x) = OPEN,\ k_{\min} = \min\left(k(x)\right) \tag{27}$$

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.

A sequence of nodes, $\{x_1, x_2, \ldots, x_n\}$, is called monotone if and only if:

$$\begin{cases} t(x_i) = CLOSED \,\&\, h(x_i, goal) < h(x_{i+1}, goal) \\ \qquad\qquad or \\ t(x_i) = OPEN \,\&\, k(x_i, goal) < h(x_{i+1}, goal) \end{cases} \tag{28}$$

The aim is to construct, for each node, a sequence of optimal paths to goal. The algorithm consists of two functions (Stentz, 1995): *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.
Initially, all the nodes receive the tag, *t(x)=NEW*, the goal node is placed in the OPEN list and *h(goal)=0.* Then, the function *Process-State* is repeatedly called until either the robot configuration, *x*, is removed from the OPEN list (*t(x)=CLOSED*) or the function returns -1 representing no valid path can be found.
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 this last case, the function Modify-*Cost* is automatically called to correct the arc costs and place the affected nodes in the OPEN list.
Let node *y* be the robot configuration at the time when a discrepancy was found in arc costs to the neighboring nodes. Through repeated calls of the function *Process-State*, until *k_min*≥*h(y)*, these cost changes are propagated from the goal node to the node *y*. At this moment, a new sequence of backpointers from the goal to node *y* is created and the robot resumes its traverse.

2.5 Focusing the D* algorithm

The drawback of the D* algorithm is in 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 the case of 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:

$$f(x,r) = h(x) + g(x,r) \tag{29}$$

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. 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. If the robot sensorial system discovers a discrepancy between the map and the structure of the environment, the search should be focused on the new robot location. But, 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 (before the affected nodes are discovered and placed in the OPEN list) and thus the nodes in the OPEN list have incorrect values for the functions *f()* and *g()*.
One possible solution is to calculate these functions each time the robot moves or a node is inserted in the OPEN list. Empirical results (Berg, 2006), (Hansen & 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()*.
It may be considered 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. Assuming 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 $f(x,r_1)$:

$$f_L(x,r_1) = f(x,r_0) - g(r_1,r_0) - \varepsilon \tag{30}$$

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. The node x is repositioned in the OPEN list function of its $f_L(x,r_1)$ value, which is a lower bound on $f(x,r_1)$, thus the node x will be selected for expansion prior to other nodes in the list. Once the node is expanded, its real $f(x,r_1)$ is computed and thus the node will be repositioned in the OPEN list function of the value in $f(x,r_1)$.
At a first glance this approach may seem inefficient, because implies resorting nodes in the OPEN list based on the $f_L()$ value, then a partial adjustment and a resorting based on the real *f()* value. But since the value $g(r_0,r_1)+\varepsilon$ is subtracted from all the nodes in the OPEN list, the list is maintained in proper order and a re-sorting is not required. Moreover, this step may be completely avoided if $g(r_0,r_1)+\varepsilon$ is added to the *f()* value of all the nodes to be inserted in the OPEN list, rather than subtracting the same value form the nodes already in the list.
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:

$$f_B(x,r_i) = f(x,r_i) + d(r_i,r_0) \tag{31}$$

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

$$f(x,r_i) = h(x) + g(r_i,r_{i-1}) \tag{32}$$

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

$$\begin{cases} d(r_i, r_0) = g(r_1, r_0) + g(r_2, r_1) + \ldots + g(r_i, r_{i-1}) + \varepsilon, i > 0 \\ d(r_0, r_0) = 0, i = 0 \end{cases} \quad (33)$$

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.
Whenever a node is removed from the OPEN list, its *f()* value is examined to see if it was computed using the most recent focal point. If not, its *f()* and $f_B()$ values are recalculated using the new focal point and accrued bias, respectively, and the node is placed back on the list. Processing the $f_B()$ values in ascending order ensures that the first encountered *f()* value using the current focal point is the minimum such value, denoted by f_{min} and let k_{val} be its corresponding *k()* value. These parameters comprise an important threshold for D*. By processing properly-focused *f()* values in ascending order, the algorithm ensures that for all nodes, *x*, if $f(x)<f_{min}$ or ($f(x)=f_{min}$ and $h(x)=k_{val}$), then *h(x)* is optimal.

3. Experimental results

Several experiments were made, in both simulation and real life, using the Pioneer2 mobile robot, to determine the advantages and disadvantages of using A* and D*.

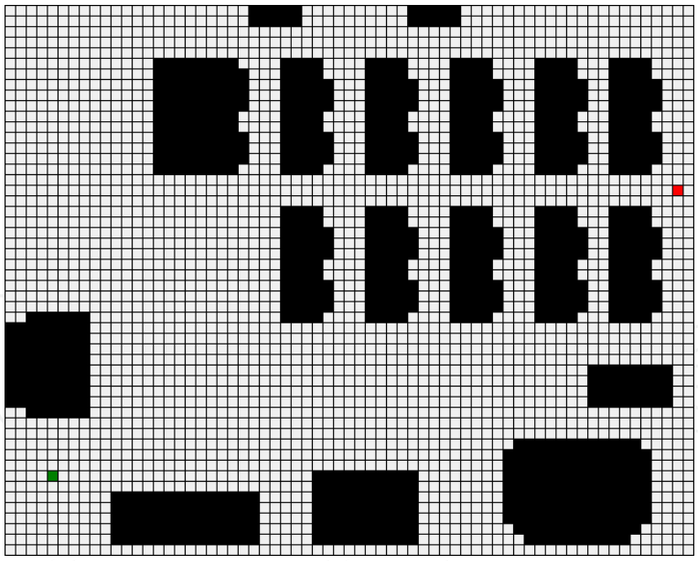

Fig. 1. The structure of the environment used for simulation

Fig. 1 presents a simulated environment having the 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).

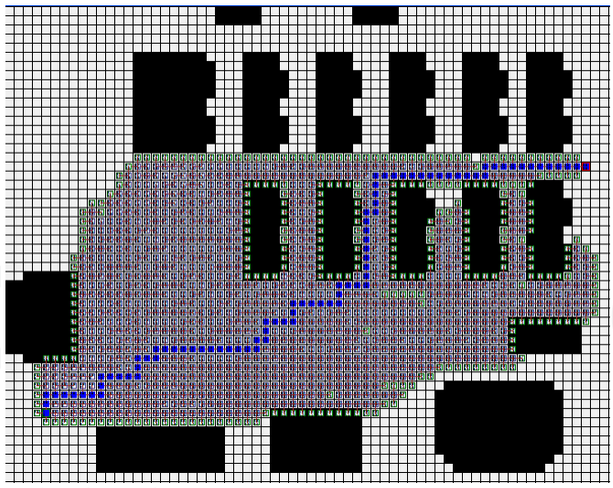

Fig. 2. Path returned by A* algorithm

Fig. 2 presents the path generated by the A* algorithm on the environment presented in Fig. 1. Distances between nodes were measured using the Manhattan metric (see also equation (2)) so that any neighboring 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.

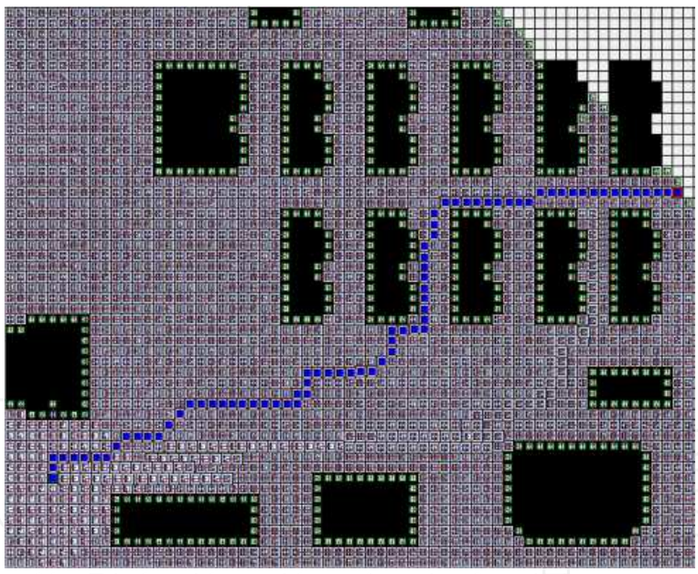

Fig. 3. Path generated by D* algorithm without focusing heuristic

Fig. 3 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. 2 and Fig. 3; 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. 4 presents the path generated by the D* algorithm with the focusing heuristic, for the same environment configuration as in Fig. 1.

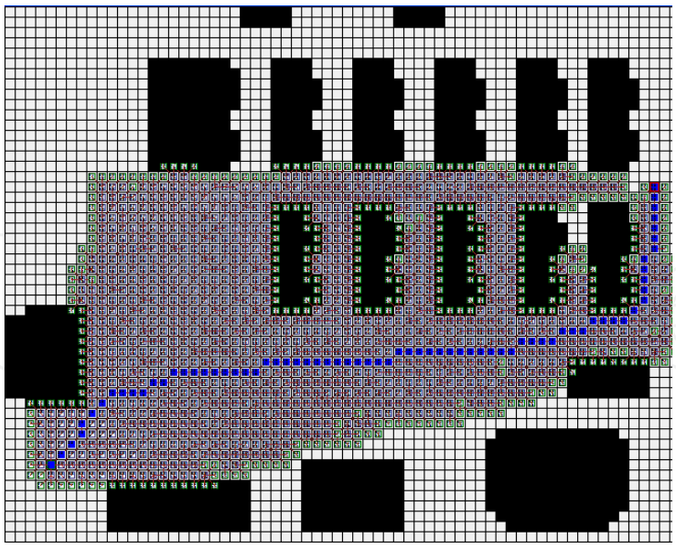

Fig. 4. Path generated by focused D* algorithm

By analyzing Fig. 2 to Fig. 4, several questions have to be answered: first, why the resulting path in the three cases is not identical and second, what are the benefits of using D* like algorithms, since, at a first glance, the A* seems to provide best results.

The fact that both A* and focused D* are using a focusing heuristic, while D* is not using it must be kept in mind when analyzing the path length in the three situations (Fig. 2 to Fig. 4). 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 minimum 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.

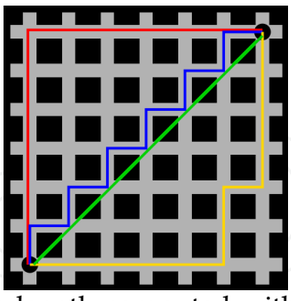

Fig. 5. Comparison between paths length computed with the Manhattan metric (blue, red and yellow) and Euclidian Metric (green)

The differences in Fig. 2, Fig. 3 to Fig. 4 can be explained using Fig. 5. The figure presents a comparison between a path produced by the Euclidian metric (in green) and three paths produced by the Manhattan metric (in blue, red and yellow). The paths produced by the Manhattan metric have all the same length, which is 12 units, while the path in green has a length of $6 \times \sqrt{2} \approx 8.48$ units, and is the unique shortest path between the two points in lower left corner and upper right corner respectively. The same green path has a length of 12 units if this path length is computed using the Manhattan metric.

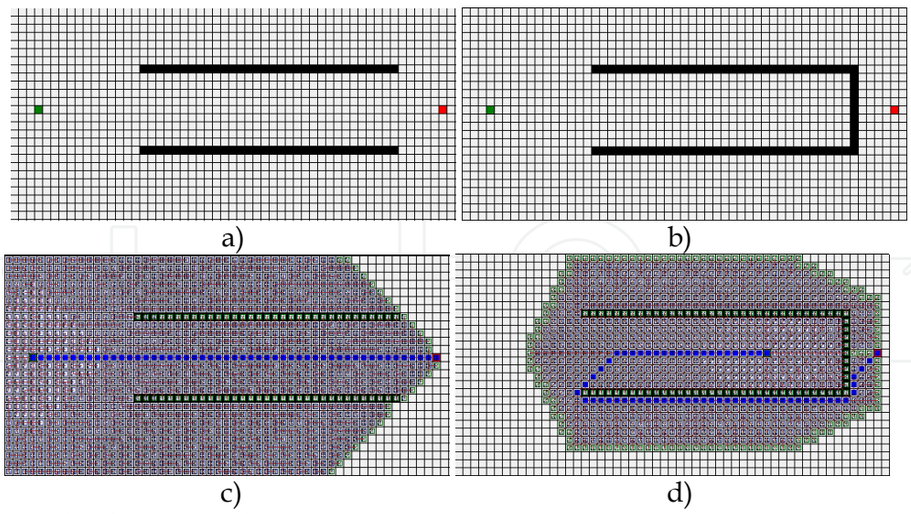

Fig. 6. Example of A* path planning.

The second problem is that both A* and focused D* algorithms expand the same number of nodes (Fig. 2 and Fig. 4), while D* expands a larger number of nodes. We will analyze the situation when the information in the map is incomplete or the structure of the environment changes. 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 corridor that has a door at the end. The robot has no information about the door presence.
Fig. 6. 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.

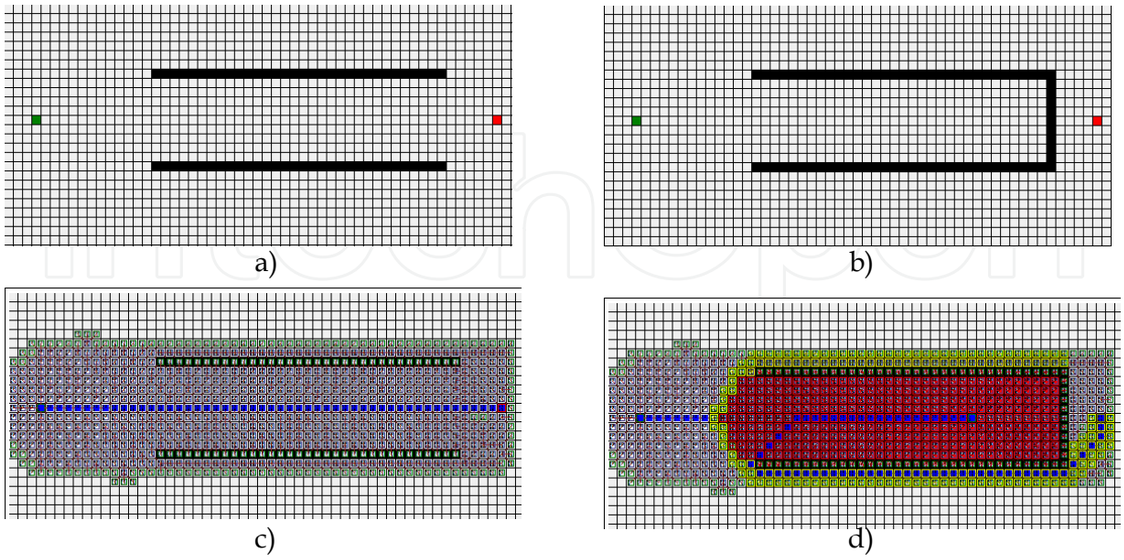

Fig. 7. Example of D* path planning

Fig. 7. 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.
Experiments presented in Fig. 6 and in Fig. 7 shows 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. 6. and Fig. 7.) because, based on the initial information, this is an unobstructed path. Then the robot starts following backpointers to the goal configuration until the closed door enters the range of the mobile robot sensorial system. At this moment 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 tries 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 Online tests

Fig. 8 presents a navigation of the Pioneer 2 mobile robot 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, errors in the data provided by the sensorial system of the robot and last but not least, error of the robot's odometric system.
Localization errors come in two types: systematic errors and non-systematic errors. The first type is represented by the error in position estimation, which, without a proper correction mechanism tend to grow in size due to their additive character and due to the integrating behavior of the localization system.
The second type of errors is non-systematic and is due to imperfect contact between mobile robot's wheels and the floor and due to slippage. The magnitude of these errors can be lowered if the mobile robot is forced to use a relative low velocity. Furthermore, there is an increase in the magnitude of these errors in case of varying roughness of the floor surface, irregular levels etc.

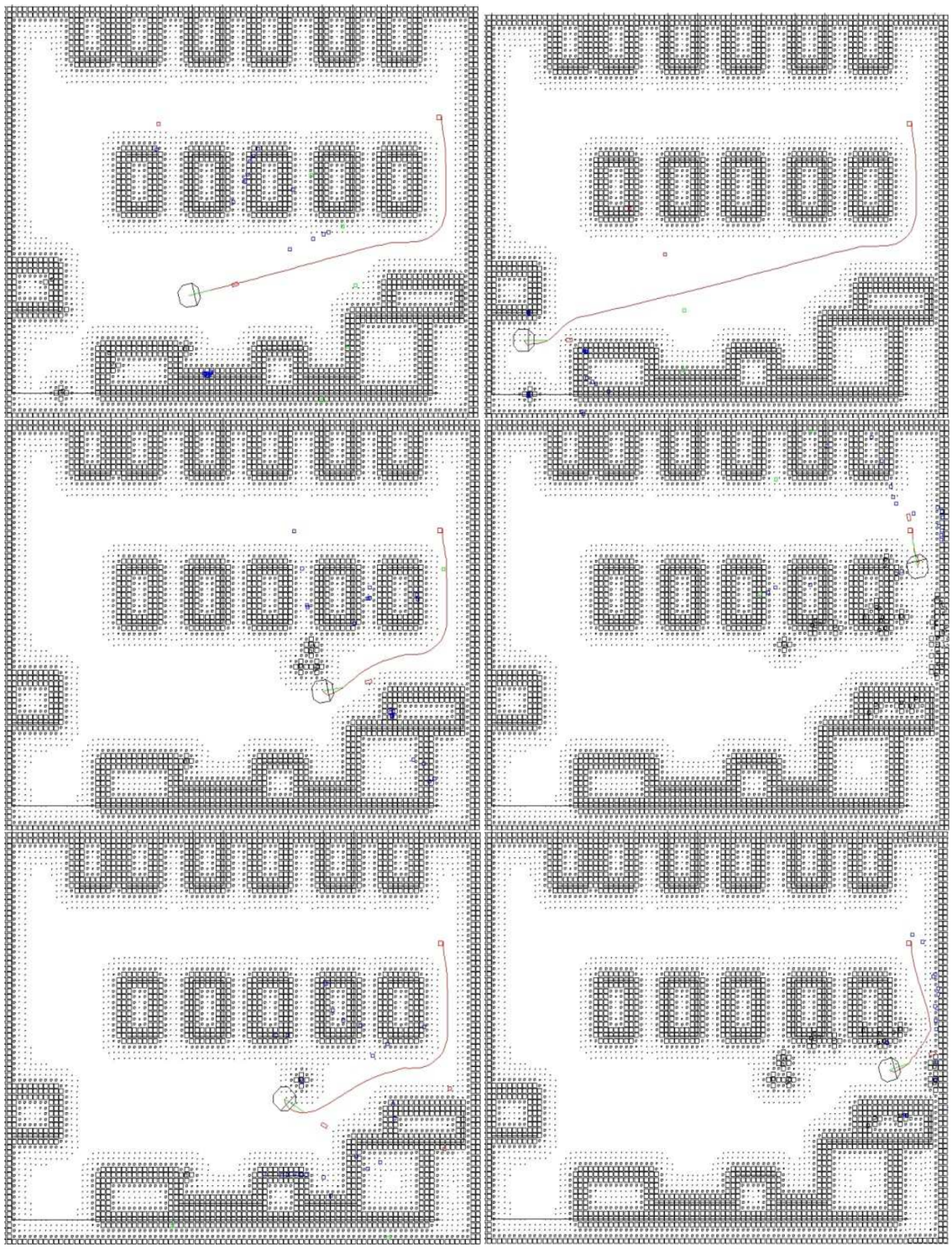

Fig. 8. Intermediate steps in navigation of Pioneer 2 mobile robot with Focused D*

3.2 Comparative tests

Several tests have been performed in order to determine the average running time between Fast A* implementation, D* without focusing heuristic and focused D*. The tests were performed off-line on random generated maze-like maps, having adjustable dimensions of 100×100 cells, 1000×1000 cells and 10000×10000 cells. Table 1 presents the run-time results (in seconds).

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.

Dimension	Fast A*	D*	Focused D*
Planning 10^4 cells	5.7 s	8.0 s	6.2 s
Re-planning 10^4 cells	3.0 s	2.1 s	1.3 s
Planning 10^6 cells	37.3 s	55.8 s	50.7 s
Re-planning 10^6 cells	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 1. Comparison between running time of the A*, D* and focused D* in planning and re-planning paths.

4. Conclusions

Although the A* and D* are algorithms specific to artificial intelligence, they can find applicability in any application 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.

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). On the other hand, 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.

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.

5. References

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. & 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, Society for Industrial and Applied Mathematics

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

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, P*roceedings IEEE International Conference on Robotics & Automation*, pp. 3310--3317

IntechOpen

Engineering the Future

Edited by Laszlo Dudas

ISBN 978-953-307-210-4
Hard cover, 414 pages
Publisher Sciyo
Published online 02, November, 2010
Published in print edition November, 2010

This book pilots the reader into the future. The first three chapters introduce new materials and material processing methods. Then five chapters present innovative new design directions and solutions. The main section of the book contains ten chapters organized around problems and methods of manufacturing and technology, from cutting process optimisation through maintenance and control to the Digital Factory. The last two chapters deal with information and energy, as the foundations of a prospering economy.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Radu Robotin, Gheorghe Lazea and Cosmin Marcu (2010). Graph Search Techniques for Mobile Robot Path Planning, Engineering the Future, Laszlo Dudas (Ed.), ISBN: 978-953-307-210-4, InTech, Available from: http://www.intechopen.com/books/engineering-the-future/graph-search-techniques-for-mobile-robot-path-planning

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen