The A-r-Star (A_r^*) Pathfinder

Daniel Opoku
Department of Electrical and Computer Engineering
North Carolina Agricultural and Technical State University
Greensboro, NC 27411

Abdollah Homaifar
Department of Electrical and Computer Engineering
North Carolina Agricultural and Technical State University
Greensboro, NC 27411

Edward Tunstel
Research and Exploratory Development Department
Johns Hopkins University Applied Physics Laboratory
Laurel, MD 20723

ABSTRACT

This paper presents a variant of the A-Star ($\boldsymbol{A}^*$) pathfinder for robot path planning called$\boldsymbol{A}_r^*$ (pronounced A-r-Star)and demonstrates that the $\boldsymbol{A}_r^*$algorithm outperforms $\boldsymbol{A}^*$ in a uniformly gridded sparse world and gives performance matching that of $\boldsymbol{A}^*$ in a uniformly gridded cluttered world. This algorithm is simple to implement and understand. It alsohighlights the performance advantages of the$\boldsymbol{A}_r^*$algorithm and proves its properties experimentally and analytically (where appropriate). Some challenges affecting the performance of $\boldsymbol{A}_r^*$have been presented and some solutions to these challenges have been developed and implemented. The performance of $\boldsymbol{A}_r^*$has been compared to$\boldsymbol{A}^*$running on both uniform and multi-resolution grids of different world scenarios. Results show that on a sparse high-resolution uniform grid world$\boldsymbol{A}_r^*$'s search speed scales well and it outperforms $\boldsymbol{A}^*$ by an exponential factor.

General Terms

Artificial Intelligence, Algorithms, Pathfinder, Graph Search

Keywords

Pathfinder, Path Planning, A-r-Star, A-infinity-Star, Multi-resolution, Path Smoothing

1 INTRODUCTION

Many researchers have studied path planning for robotics [1] using occupancy grid map representations of the world. In this context, the world is usually considered as a finite area within which a robot will operate. Here the map is represented as a mesh of nodes S spread over the continuous space of locations in the world. Each node, $s \in S$ of the mesh is parameterized by its occupancy p_o, or probability of being occupied by an object or obstacle. Thus, for the 2D binary occupancy grid such as that used in this paper, $p_o = 1$ expresses an occupied node and $p_o = 0$ expresses an unoccupied node. For robotic path planning, the 2D grid is often used to represent a slice of a 3D world. The path planning problem is stated as: Given a start node and goal node belonging to a given grid world, find the shortest unblocked path connecting them. This paper presents a novel algorithm, $\boldsymbol{A}_r^*$(pronounced A-r-Star), a modified version of the $\boldsymbol{A}^*$ pathfinder for path planning in a uniform grid world. This new algorithm outperforms $\boldsymbol{A}^*$ path planning in a sparse uniformly gridded world and matches $\boldsymbol{A}^*$ in a cluttered world. The $\boldsymbol{A}_r^*$algorithm essentially interweaves the building of a non-uniform grid out of a uniform gridded world with path-finding. When given a uniform grid world, $\boldsymbol{A}_r^*$decimates the nodes within a given radius/range (r) and forms bigger nodes out of them. It has been shown both experimentally and analytically (where appropriate) that $\boldsymbol{A}_r^*$ possesses most of the desirable features of$\boldsymbol{A}^*$. Besides, some challenges that affect the performance of$\boldsymbol{A}_r^*$ have been identifiedand some strategies to mitigate them have been developed thereby bringing$\boldsymbol{A}_r^*$'s performance close to optimal. The performance of $\boldsymbol{A}_r^*$has been compared to$\boldsymbol{A}^*$running on both uniform grids and multi-resolution grids of different world scenarios. Results show that, when running on sparse high dimensional, high resolutiongrid worlds, $\boldsymbol{A}_r^*$'s performance scales well (linearly) with increasing grid size and outperforms $\boldsymbol{A}^*$ by an exponential factor.

The rest of the paper is organized as follows. Section 2 presents a literature review of related pathfinders. Section 3presentsthe motivation for developing the $\boldsymbol{A}_r^*$ algorithm followed by the description of $\boldsymbol{A}^*$in Section 4. The new pathfinder $\boldsymbol{A}_r^*$ is presented in Section 5 followed by the description of its properties in Section 6, challenges in Section 7 and suggested solutions in Section 8. Section 9 presents results of simulation experiments highlighting algorithm properties followed by conclusion and future work in Section 10.

1.1 Nomenclature

S = finite set of all nodes to be searched, $\Rightarrow\ S_b \cup S_f = S$;
S_f =set of all unoccupied/unblocked nodes;
S_b = set of all blocked nodes;
S_{Open} = set of all the nodes that ever make it to the $OPEN$ list, thus $S_{Open} \subseteq S_f$;
S_{Closed} = set of all the nodes on the $CLOSED$ list, thus$S_b \subseteq S_{Closed}$;
$n(S)$ = cardinality of set S;
s =an individual node, $s \in S$;
$c(s', s)$ = cost of traversing from node $s$to its neighbor s';
s_{Start} = starting node;
s_{goal} = goal node
$g(s)\colon s \in S_f$ = cost of moving from s_{Start} to s;
$h(s)\colon s \in S_f$ = estimated heuristic cost of moving from node s to s_{Goal};
$succ(s)$= set of all the successors of s;
$pred(s)$ = set of all the predecessors of s;
$parent(s)$ = parent of $s, \Longrightarrow parent(s) \in pred(s)$.

2 LITERATURE REVIEW

The heuristic search method, $\boldsymbol{A}^*$[2] is by far the most commonly used in artificial intelligence for solving minimum distance path planning problems due to its simplicity and ability to find an optimal obstacle-free path if and only if one exists between a given start node and a goal node. Moreover, if the heuristic function used for the cost modeling never overestimates the actual minimal cost of reaching the goal (i.e., it is admissible) then $\boldsymbol{A}^*$is admissible and thus always optimal (i.e., it will explore/expand the minimum possible number of nodes necessary) [3]. Subsequently, the use of the $\boldsymbol{A}^*$ algorithm for path planning in both robotics and video games has increasingly gained popularity. To use $\boldsymbol{A}^*$ for path planning given a 2D map of the environment, one first reduces the robot size to a point robot and enlarges the obstacles in the environment by the radius of the robot's cross-section (using configuration space computation) [4][5]. This ensures that the robot can traverse any path of empty nodes returned by the search algorithm without colliding with an obstacle. Given start and goal nodes, the algorithm searches through continuous obstacle free 4-connected or 8-connected nodes (depending on application) for a path with minimum cost from the start to goal, where the cost of a path is the sum of the costs of its arcs or segments.

The $\boldsymbol{A}^*$ algorithm has limitations that make its application to some especially large grid worlds unattractive. One of the limitations is that the$\boldsymbol{A}^*$search time increases exponentially with increasing grid size. This is so because finding adjacent nodes with lowest cost,$f(s)$, can be time consuming due to the growing size of the OPEN list, a data structure storing the nodes to be evaluated. Also, insertion of a node onto the OPEN list involves computation of the node's associated costs which makes searching through many nodes very computationally expensive. Various solutions have been suggested in literature to mitigate this challenge such as the following [6]:

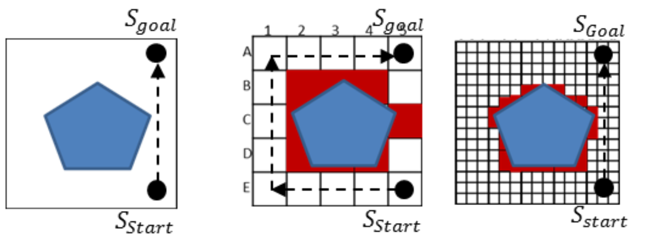
(i)The OPEN list can be implemented with a priority queue data structure sorted according to the cost function, $f(n)$. Thus, determination of the node with lowest cost reduces to simply popping the priority node of the queue. The priority queue creation takes $O(n)$ time[7]and thus the time scales linearly with fast growing OPEN lists;

(ii) Instead of using a list/queue, if memory is not a limiting resource, the OPEN list can be implemented with arrays using memory allocation and access routines to reduce the time it takes to find the node with lowest cost function to $O(1)$;

(iii) A low resolution/coarse grid can be used with fewer nodes to search, and hence permits faster run time. However, coarse discretization of the continuous world can result in the creation of unrealistic paths where an obstacle is close to the boundary (see Figure 1) and it can also place a node beyond reach especially in a cluttered environment (see Figure 2);

(iv) The use of a multi-resolution grid representation of the environment is another way to solve this problem in a sparse world. This involves preprocessing the world into a non-uniform grid representation such as quadtrees. Using this approach involves an initially high capital cost in building the world map. However, once that is completed, it yields good run time performance in terms of search speed. Hence, this approach is suitable for planning in a static world.

Regarding solution (iv) described above,small obstacles and protrusions from irregular obstacles in a region result in the creation of small obstacle nodes due to the recursive nature of the quadtree. These fragment the free space, giving rise to an undesirable increase in the depth of the quadtree and the number of leaf nodes and consequently increasing the cost of the search [8].

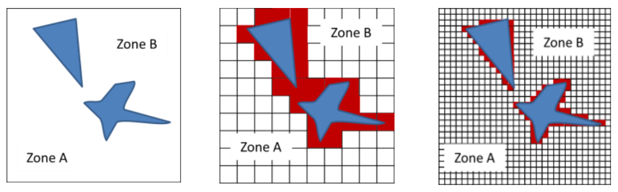


(a) Continuous world
(b) The course gridded world
(c)The fine gridded world

Figure 1: How increasing resolution can help graph search algorithms.

$\boldsymbol{A}^*$ usually returns a 'kinky' path (path with many unnecessary turns) making its direct usage by a robot impractical. Post smoothing and interleaved smoothing techniques are usually employed to remove these kinks [9].

$\boldsymbol{A}^*$, being an offline search technique works on the assumption that the world is fully known and unchanging. Whenever the robot observes a discrepancy between the given map and the navigation world that affects the planned path (e.g., a node on the path is unexpectedly blocked), it has to re-plan a new path starting anew from its current position. This becomes unbearable in even a moderately slow changing environment with high-dimension grid. Despite its optimality, the gross inefficiency accumulating from multiple re-plans may require a high-performance computer for real-time operation.



(a) Continuous world
(b) The course gridded world
(c)The fine gridded world

Figure 2: How increasing resolution can help graph search algorithms.

In effect, $\boldsymbol{A}^*$ is best suited for offline path planning in a static world. This has prompted the development of a vast number of incremental path planning techniques. Anthony Stentz's $\boldsymbol{D}^*$[10] is by far the most popular among the incremental search techniques. $\boldsymbol{D}^*$is capable of planning paths across the spectrum of fully known to unknown environments in an efficient, optimal, and complete fashion. As in$\boldsymbol{A}^*$, the environment is modeled as a graph, where each node represents a robot state (e.g., a location in a house), and each arc represents the cost of moving between two states (e.g., distance to travel). $\boldsymbol{D}^*$ initially exploits the available information contained in the map (if any) to plan a path from the goal to the robot's position in a way similar to $\boldsymbol{A}^*$. The robot then starts navigating using this path while simultaneously observing its environment through its sensors. If the robot discovers that the path is blocked (path changes are handled as arc cost changes), it calls on $\boldsymbol{D}^*$to plan a new path. $\boldsymbol{D}^*$propagates information about the arc cost changes minimally to all affected nodes in the graph to compute a new optimal path[11]. The robot repeats this process until it reaches the given goal or accrues enough information to establish that the goal is unreachable. This makes its application in a dynamic world attractive. However, the initial planning of $\boldsymbol{D}^*$a takes longer time since it operates as a Breadth-First Search. $\boldsymbol{Focused\ D}^*$[12] is an extension of $\boldsymbol{D}^*$ that uses heuristics to focus the search to significantly reduce the total time required for both the initial path calculation and subsequent re-planning operations thereby making the $\boldsymbol{D}^*$algorithm a full generalization of $\boldsymbol{A}^*$ for dynamic environments.$\boldsymbol{D}^*$ $\boldsymbol{Lite}$applies $\boldsymbol{Lifelong\ Planning\ A}^*$ to robot navigation in unknown terrain, including goal-directed navigation in unknown terrain and mapping of unknown terrain[13]. The $\boldsymbol{D}^*\boldsymbol{Lite}$ algorithm is easy to understand and analyze;it implements the same behavior as Stentz's$\boldsymbol{Focused\ D}^*$ but is algorithmically different. The $\boldsymbol{Field\ D}^*$[14,15] algorithm is a variant of $\boldsymbol{D}^*$that uses interpolation to improve the smoothness of the path returned both in the planning and the re-planning phase.

$\boldsymbol{A_r}^*$is a fast version of$\boldsymbol{A}^*$that achieves exponential speedup in a sparse grid map. Itachieves this by interleaving path finding with multi-resolution gridding. This reduces the number of nodes that need to be explored in a sparse grid map, the computational intensity and hence the speedup.

3 MOTIVATION

Map building using fixed node decomposition (i.e., the continuous world is tessellated into a discrete approximation of the continuous map) is inexact resulting in the loss of narrow passages in this transformation. The higher the resolution of the grid, the closer the approximation is to the continuous world. But increasing the resolution introduces more free nodes and increases the search space leading to a sparse grid. The map of most indoor environments can be considered sparse if decomposed into nodes of high resolution. Applying $\boldsymbol{A}^*$ to a sparse grid is not attractive since its run time is on the order $O(n^2)$. To solve this problem, multi-resolution gridding [8, 16] is usually adopted so that areas around obstacles receive high resolution gridding and areas far from obstacles are represented by much courser gridding. But building multi-resolution/adaptive grids requires more work in the decomposition than building uniform/fixed node grids. Besides, determining which node a given position belongs to and finding the neighbors of a given node in a multi-resolution grid is a non-trivial task [17]. The motivation is therefore to develop a complete and correct search algorithm that can plan paths in high-dimension, high-resolution grid maps faster. This will enable us to handle large grid sizes and thus encourage increasing the resolution of the grid without the need for multi-resolution.

4 A-STAR (A^*) ALGORITHM DESCRIPTION

$\boldsymbol{A}^*$is a best-first search algorithm that finds the least costly path from an initial configuration to a final configuration in a given finite and static grid world. It uses an estimate of the start distance $g(s)$ and heuristic estimation of the goal distance $h(s)$ to define a cost/sorting function,$f(s)$ (i.e., $f(s) = g(s) + h(s)$). Generally, the $\boldsymbol{A}^*$algorithm maintains two lists namely $OPEN$ list and$CLOSED$ list. The $OPEN$ list is a priority queue of all the states which have at least one of their predecessors already explored and as such are potential candidates for next exploration. The $CLOSED$ list holds the candidates that have been explored at least once (and often the blocked nodes). The algorithm starts with an empty $OPEN$ list and populates it with the starting node. At the beginning of every iteration, the node with the minimum cost (s) , is popped from the $OPEN$ list. If that is the goal node, the algorithm terminates and follows back pointers to extract the shortest path (i.e., the path with minimum cost). Else, that node is placed on the $CLOSED$ list and then expanded (i.e., the neighbors are generated and conditionally placed on the OPEN list).It proceeds with the iteration until the goal node is expanded or the $OPEN$ list becomes empty in which case it returns *'no path found'*. The pseudocode is shown in Algorithm 1.

```
{1}  $\boldsymbol{Main}()$
{2}      $g(s_{Start}) := 0;$
{3}      $parent(s_{Start}) := s_{Start};$
{4}      $OPEN := \emptyset$
{5}      $OPEN.\boldsymbol{Insert}(s_{Start}, g(s_{Start}) + h(s_{Start}));$
{6}      $CLOSED := S_b$
{7}      $\boldsymbol{while}\ OPEN \neq \emptyset\ \boldsymbol{do}$
{8}          $s := OPEN.\boldsymbol{Pop}();$
{9}          $\boldsymbol{if}\ s = s_{goal}\ then$
{10}             $\boldsymbol{return}$ "$Unblocked\ path\ found$";
{11}         $CLOSED := CLOSED \cup s;$
{12}         $neighbors := \boldsymbol{Expand}(s, CLOSED)$
{13}         $\boldsymbol{foreach}\ s' \in neighbors\ \boldsymbol{do}$
{14} $if\ s' \notin CLOSED$
{15}             $if\ s' \notin OPEN$
{16}                 $g(s') := \infty;$
{17}                 $parent(s') := NULL;$
{18}             $\boldsymbol{UpdateNode}(s, s');$
{19}     $\boldsymbol{return}$ "$no\ unblocked\ path\ found'$
{20} $\boldsymbol{end}$
{21} $\boldsymbol{Expand}(s, CLOSED)$
{22}     $r := 1;$
{23}     $neigbor(s) := \boldsymbol{NeighborsGen}(s)$
{24}     $\boldsymbol{foreach}\ s' \in neighbor(s)$
{25}         $\boldsymbol{if}\ s' \notin CLOSED;$
{26}         $neighbors := neighbors \cup s'$
{27}     $\boldsymbol{return}\ neighbors$
{28} $\boldsymbol{end}$
{29} $\boldsymbol{UpdateNode}(s, s')$
{30}     $\boldsymbol{if}\ g(s) + c(s, s') < g(s')\ \boldsymbol{then}$
{31}         $g(s') := g(s) + c(s, s');$
{32}         $parent(s') := s;$
{33}         $f(s') := g(s') + c(s', s_{goal})$
{34}         $\boldsymbol{if}\ s' \in OPEN\ \boldsymbol{then}$
{35}             $OPEN.\boldsymbol{Remove}(s');$
{36}         $OPEN.\boldsymbol{Insert}(s', g(s'), f(s'));$
{37} $\boldsymbol{end}$
```

Algorithm 1: The A* algorithm Pseudo code

5 A-R-STAR (A_r^*) ALGORITHM

5.1 Definitions

Node Distance of s'' from s is defined as the fewest number of nodes that will be touched by a straight line connecting s and s'' (see Figure 3). This is equivalent to the distance from s to s'' using the *'chessboard'* distance metric. Thus, two nodes with different Euclidian distances can have the same node distances.

Level-R-Neighbors of a node s comprise all nodes within node distances equal to *R* from s. This implies that the *Level-1-Neighbors* of s are its 8 contiguous neighbors (see Figure 3). Here *R*refers to the radius of the 'ball' (a box in the case of square grid) formed by connecting the centers of all *Level-R-Neighbors.*

5.2 Algorithm Description and Implementation

The $\boldsymbol{A_r^*}$algorithm is a modified version of the $\boldsymbol{A^*}$algorithm that interweaves node decimation with path-finding in a uniform grid/mesh. For a given node, $\boldsymbol{A^*}$ counts only immediate nodes (4- or 8- connected nodes) as its neighbors. This implies that even when all the nodes in a particular area are similar, it will still do some computation for all of them. The effect is that, if an obstacle blocks the direct line of sight close to the goal, the number of nodes needed to be explored more than doubles (see Figure 4) which translates into increasing the search time.

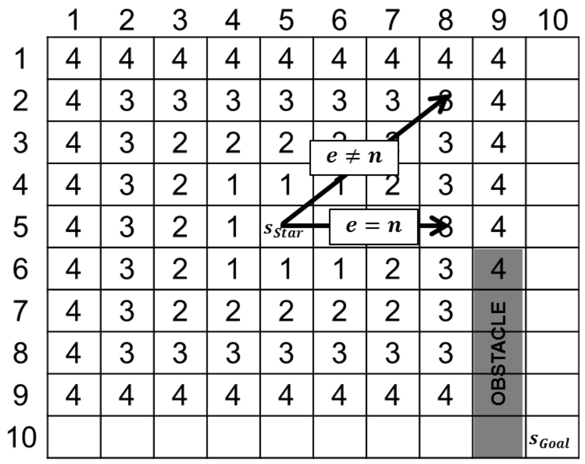


Figure 3: Showing the *Level-R-Neighbors* for a given node s_{start}. All the nodes at a given level bear their node distance and e is the Euclidean distance and n is the node distance (R). Here$r_0(s_{Start}) = 4$.

Using the idea from non-uniform mesh building[18], a cluster of nodes with similar characteristics can be represented with a bigger node with minimal loss of information (see Figure 5).

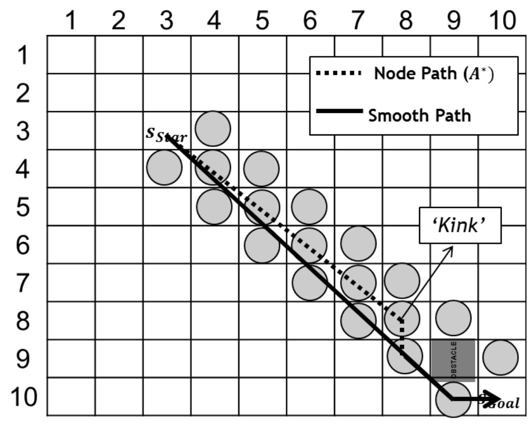


Figure 4: Showing (a) how A^* responds to an obstacle; (b) 'kinky' path returned by A^* and smooth path after post process. The circles indicate the nodes that were explored before the path was found.

The $\boldsymbol{A_r^*}$algorithm therefore allows collapsing of such nodes into a single node with properties that are commensurate with the union of those nodes (see Figure 5). For multi-level terrain, one will use a distance transform [19, 20] to identify changes in the nodes; but for the binary occupancy grid such as the one used in this paper, the task reduces to identifying the nearest obstacle to the current node. The overall effect is reduction in the number of nodes needed to be explored, computation cost and increased search speed in a sparse uniform gridded world. The *'star'* in the name does not suggest that it always finds an optimal path, it is just intended to retain its resemblance toits namesake, $\boldsymbol{A^*}$. The *r* stands for radius (range) defined as the maximum allowable radius (in node distance) of a 'ball' of nodes that can be counted as the neighbors of a given node (see Figure 3). Thus, only *Level-R-Neighbors* are considered during the search where $1 \leq R \leq r$.

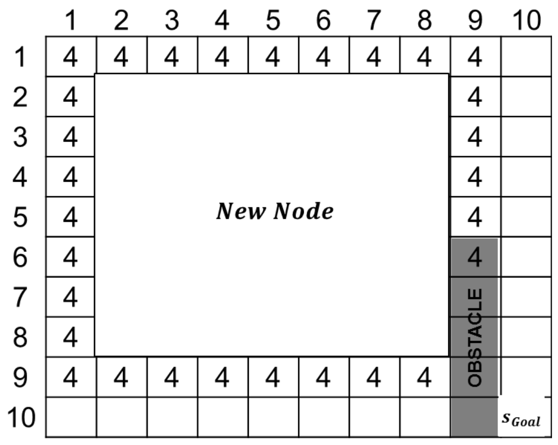


Figure 5: The nodes at Level-1 to Level-3 decimated to form one big node with the original *Level-4-Neighbors* of s_{Start} now forming the neighbors of the new node.

Let the node distance fromthe closestobstacle node to a given node $s$be $r_0(s)$, then at the end of the search: $r_0(s) \leq r \; \forall \; s \in S_{Open}$. Implementation-wise, this is achieved by searching for the minimum R such that at least one of the *Level-R-Neighbors* of a node being expanded belongs to the set of blocked nodes. Then all nodes in the neighborhood of s such that $R < r_0(s)$ are tagged as skip nodes ($s'.tag = Skip$) and nodes such that $R = r_0(s)$ are returned as the *Level-R-neighbors*.The pseudocode for the algorithm is similar to that of $\boldsymbol{A^*}$ with two modifications. The first modification is done to the *Expand subroutine* and the resulting algorithm is referred to as**Basic $\boldsymbol{A_r^*}$**. This is achieved by replacing the lines {21} to {28} in Algorithm 1 by the lines in Algorithm 2.

```
{21} $\textbf{Expand}(s, S_{Closed}, S_f, s_{goal}, r)$
{22}     $R := 1;$
{23}     $skipflag := true$
{24}     $while\ R \leq r$
{25}         $levelrneighbor(s) := \textbf{LRNG}(R, s)$
{26}         $\textbf{foreach}\ s' \in levelrneighbor(s)$
{27}             $\textbf{if}\ s' \notin S_f$
{28}                 $skipflag := false$
{29}                 $\textbf{continue}$
{30} $\textbf{Elseif}\ s'.tag \neq skip\ and\ s' \notin S_{Closed}$
{31} $lrneighbors := lrneighbors \cup s'$
{32}         $\textbf{if not}(skipflag)$
{33}             $\textbf{return}\ levelrneighbor(s)$
{34}         $\textbf{Foreach}\ s' \in levelrneighbor(s)$
{35}             $s'.\textbf{tag} := skip;$
{36}             $r++$
{37} $\textbf{return} lrneighbors$
{38} $\textbf{end}$
```

Algorithm 2: First modification resulting in the Basic $\mathbf{A_r^*}$

Note that nodes that are tagged skip will never make it to the $OPEN$ list (Algorithm 2 line {30}) but a node might make it to the $OPEN$ list before being tagged as skip. Thus, in the $\boldsymbol{Basic}\ \boldsymbol{A_r^*}$algorithm, tag *open* dominates/overwrites tag *skip*. The second modification is to switch the tag dominance to *skip*dominating *open*. Thus, nodes that made it to the $OPEN$ list from one node before being tagged as skip from another node will not be expanded (i.e., they will stay on the $OPEN$list till the algorithm terminates). The resulting algorithm after the last modification is called the $\boldsymbol{A_r^*}$ algorithm. The $\boldsymbol{A_r^*}$pseudocode can be derived from the $\boldsymbol{Basic}\boldsymbol{A_r^*}$ by inserting the lines in Algorithm 3 after the line {10} in Algorithm 1.

```
{11}        $\boldsymbol{if}\ s.\boldsymbol{tag} = skip\ \boldsymbol{then}$
{12}            $\boldsymbol{continue}$
```

Algorithm 3: Second modification resulting in the A_r^*

5.3 Finite Radius (r) vs. Infinite Radius (∞)

The choice of r can be fixed from the start of the algorithm to a finite value. For example, choosing $r = 1$ reduces the algorithm to $\boldsymbol{A_1^*}$ which is essentially$\boldsymbol{A}^*$. But choosing an appropriate finite value for r requires absolute knowledge of the environment since the choice of r affects the performance of the algorithm. The simplest solution is to allow the algorithm toevolve and discover the valueof r during the search. This is achieved by pegging the value of r at infinity (i.e., choosing $r = \infty$). This leads to what is referred to as the A_∞^* (A-Infinity-Star). Here, infinity (∞) is defined as $r \geq r'$ where r' is the radius of the biggest 'ball' of continuous free nodes available in the search space. It is trivial to derive that for an $M\ x\ M$ grid world, $r' < M/2$ always holds (strictly less because the node under consideration cannot be counted as part of the radius and r is undefined for $s \notin S$).

5.4 Choice of Level-R-Neighbors Generator (LRNG)

The***LRNG*** function is responsible for generating *Level-R-Neighbors*of a given node s. A good choice of the ***LRNG***function should return at every*Level-R*, all and only the nodes at radius R from s as the *Level-R-Neighbors* of s. This is a necessary condition for $\boldsymbol{A_r^*}$to be complete and correct. On square grids, choosing the ***LRNG***is a trivial task but this is not trivial when other geometric shapes are used for the gridding.

5.4.1 Theorem 1

If the *Level-R-Neighborhood*generation function of $\boldsymbol{A_r^*}$at every *Level-R*returns all and only the nodes at radius R from s for $R = 1, \dots, r$ then the $\boldsymbol{A_r^*}$ algorithm is complete and correct.

*Proof:*Let us assume the contrary that the path, $s_1, \dots, s_i, \dots, s_n$ returned by the algorithm is incorrect, thus there exists at least one s_i between s_1 ands_n, $1 < i < n$, such that $s_i \in S_b$. This will imply that a blocked node $s_i$got expanded by the algorithm which contradicts the line {30} of Algorithm 2 and therefore cannot be true. Similarly, let us assume that a path actually exists but $\boldsymbol{A_r^*}$did not find a path. This will imply at a certain *Level-R*, the algorithm failed to return a node $s \in S_f$and hence assumed there was not a path available. This is the necessary condition for a function to qualify as an ***LRNG*** and hence poses a contradiction that cannot surface.

6 PROPERTIES OF THE A-R-STAR ($\boldsymbol{A_r^*}$) ALGORITHM

6.1 Completeness

Like$\boldsymbol{A}^*$, the $\boldsymbol{A_r^*}$algorithm is complete meaning it will find a path if one exists between the start and the goal node. The condition for completeness solely depends on the $\boldsymbol{LRNG}$ as stated in **Theorem 1**.

6.2 Correctness

The correctness property holds for the $\boldsymbol{A_r^*}$algorithm. This implies that if $\boldsymbol{A_r^*}$ does return a path for a given starting and ending node, then that path is a truly unblocked path (that is, the path exists and is correct).

6.3 Termination in Finite Time

The use of the *CLOSED* list and the tagging ensure that $\boldsymbol{A_r^*}$ expands (or tags) every node once. Since the world is an $M\ x\ M$ grid, where M is finite, it is implicit that the algorithm will terminate in finite time.

6.4 Convergence to A*

The performance of the $\boldsymbol{A_r^*}$ (and for that matter$\boldsymbol{Basic}\ \boldsymbol{A_r^*}$) approaches that of $\boldsymbol{A}^*$for worlds with increasing clutter. Let us define a perfectly cluttered 2D world as a grid configuration such that every *Level-1-Neighbor* of $s\ \forall s\colon s \in S_f$ contains at least one $s : s \in S_b$. Given a perfectly cluttered world, the $\boldsymbol{A_r^*}$ will be forced not to tag any node as skip. Thus, $\boldsymbol{A_r^*}$ will implicitly operate as $\boldsymbol{A_1^*}$ which is essentially$\boldsymbol{A}^*$.

6.4.1 Theorem 2

In a perfectly cluttered world, $\boldsymbol{A_r^*}$ and $\boldsymbol{Basic}\ \boldsymbol{A_r^*}$ converge to $\boldsymbol{A}^*$ for all positive integer values of$r$.

*Proof:*Assume the set S is a perfectly cluttered 2D environment. Then every *Level-R-Neighborhood* of a free cell $s \in S_f$ will contain at least one $s'\colon s' \in S_b$ and thus $r_0(s) = 1\ \forall\ s \in S_f$; from subsection 5.2,$1 \leq R \leq r_0(s) \Longrightarrow R = 1$ for all nodes. But the A_r^* runs at *Level-R* $= 1$ throughout the search and so it is intuitive that after the search, $r_0(s) = 1\ \forall\ s \in S_{Open}$ and hence the proof.

6.5 Any Angle Path Planning

Most of the grid-based path planning algorithms that operate on a 2D world use discrete state transitions that are artificially constrained to a small set of possible headings angles (e.g.,$0, \frac{\pi}{4}, \frac{\pi}{2}$, etc.). The ramification is that the path returned by even the optimal grid planner will not be the shortest possible continuous path (see Figure 4). The $\boldsymbol{A_r^*}$ algorithm is not constrained to a finite set of angles. This means that it sometimes returns a more natural and smooth path than$\boldsymbol{A}^*$. The $\boldsymbol{A_r^*}$ algorithm under sparse conditions can therefore be considered as an *'any angle path planner'*.

6.6 Reaction to Obstacle

The $\boldsymbol{A_r^*}$ algorithm reacts to an obstacle by planning in small steps till it avoids the obstacle. This mimics intuitive navigating behavior. Much caution is taken when navigating close to an obstacle than when navigating far from an obstacle.

6.7 Definitions

Given two nodes, s_{Start} ands_{Goal}, a *node path*isdefined as any chain of nodes $s_1, s_2, \dots, s_n \in S_p$ such that everys_{i+1} belongs to the *Level-1-Neighbors* of s_i and $s_i = s_j iff\ i = j$and$s_i \notin$

$S_b \; \forall \, i,j \in [1,n]$. A continuous path is an unbroken curve drawn from the center of s to the center of s'' without passing through a blocked node. Let $\Gamma(s,s'')$ be the set of all continuous paths linking the centers of s and s'', and let $\Gamma_\mathrm{p}(s,s'')$ be a specific continuous path linking the centers of s ands''. Define $\Gamma_\mathrm{m}(s,s'') = \min_{cost(s,s'')} \Gamma(s,s'')$. Thus, for a given grid map, if $Line\ Of\ Sight, \boldsymbol{LOS}\ (s,s'') = true$ (i.e., there is an unobstructed straight line path from s to s'') then $\Gamma_\mathrm{m}(s,s'')$ = a straight line.

7 CHALLENGES OF THE A-R-STAR (A_r^*) ALGORITHM

7.1 Bulges

$\boldsymbol{A_r^*}$ usually returns a path with $bulges$ in it (see Figure 6). This is a major challenge to the performance of$\boldsymbol{A_r^*}$. Given two nodes, s and s'', if $\boldsymbol{LOS}\ (s,s'') = true$ and $\Gamma_p(s,s'') \neq \Gamma_\mathrm{m}(s,s'') =$ straight line, then $\Gamma_p(s,s'')$ is called a $bulged$ path and, in general, any $\Gamma_\mathrm{p}(s,s'') \in \Gamma(s,s''): \Gamma_\mathrm{p}(s,s'') \neq \Gamma_\mathrm{m}(s,s'')$ is said to be a *bulged* path. This definition implies that a *'kink'* is a type of *bulge* (see Figure 4). There are two main causes of *bulges* in $\boldsymbol{A_r^*}$; namely, *angular constraint (kinks)*and *premature tagging*. *Angular constraint* occurs around obstacles where the algorithm navigates in small steps; the navigation angles are thus artificially constrained to a small set of angles. *Premature tagging*occurs due to local minima. The greedy heuristic of $\boldsymbol{A_r^*}$is initially drawn into a local minimum. This is accompanied by the tagging of nodes as *skip*. When the algorithm bounces back from the local minimum, these nodes which were prematurely tagged as *skip* are not considered for expansion and this creates a *bulge*.

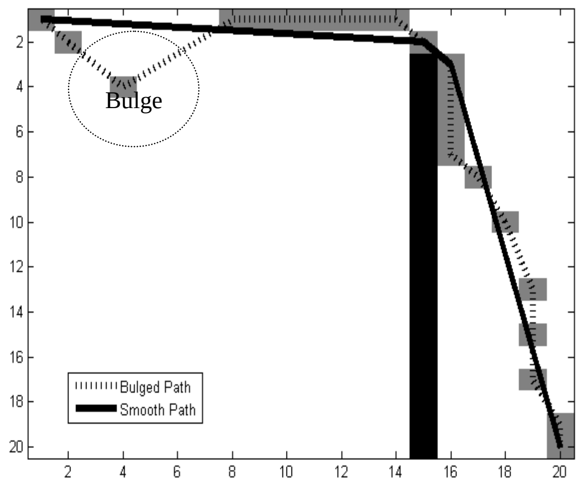


Figure 6: An example of a path planned by A_r^* highlighting the challenges posed by bulges and bulge elimination using post smoothing.

7.2 Non-optimality

Due to *bulges*, the path returned by$\boldsymbol{A_r^*}$ is not always optimal. *Bulges* introduce extra cost into the sorting function by increasing the estimated goal distance thereby placing some nodes at a disadvantage. The goal distance becomes dependent on the configuration of the obstacles in the environment. Consequently, $g(s_1) < g(s_2)$ does not always imply $g^*(s_1) < g^*(s_2)$ during the search (where $g^*(s_n)$ is the actual optimal path between s_{start} and s_n). Thus, unlike the $\boldsymbol{A^*}$ algorithm, $\boldsymbol{A_r^*}$ does not always guarantee an optimal path.

8 PROPOSED SOLUTIONS TO THE CHALLENGES

8.1 Bulge Removal: Post Dissociative Smoothing (PDS)

A Post Dissociation Smoothing (PDS) algorithm similar to that outlined in [9]has been developed and implemented to eliminate the *bulges* in the path returned by $\boldsymbol{A_r^*}$. This shortens the path and gets it closer to the shortest possible path. Algorithm 4 shows the pseudocode for the PDS. Both$\boldsymbol{LOS}$and $\boldsymbol{DissociatePath}$are derived from the Bresenham line drawing algorithm similar to that in [9].

```
{1} $\boldsymbol{Smooth}(Path)$
{2}  $DisPath = \boldsymbol{DissociatePath}(Path)$
{3}      $s_p = DisPath.\boldsymbol{pop}()$
{4}      $s_{old} = DisPath.\boldsymbol{pop}()$
{5}      $s_{cur} = DisPath.\boldsymbol{pop}()$
{6}      $smoothPath.\boldsymbol{push}(s_p)$
{7}      $\boldsymbol{while}\ Path \neq \emptyset$
{8}          $\boldsymbol{if}\ \boldsymbol{LOS}(s_p, s_{cur})\boldsymbol{then}$
{9}              $s_{old} = s_{cur}$
{10}             $s_{cur} = DisPath.\boldsymbol{pop}$
{11}             $\boldsymbol{continue}$
{12}         $smoothPath.push(s_p)$
{13}         $s_p = s_{old}$
{14} $smoothPath.\boldsymbol{push}(s_{cur})$
{15} $\boldsymbol{return}\ smoothPath$
{16} $\boldsymbol{end}$
```

Algorithm 4: Post Dissociative Smoothing Algorithm

8.2 Non-optimality: Interleave Smoothing with Post Dissociative Smoothing (IS-PDS)

Some path configurations cannot be smoothed into the shortest possible/optimal path. To increase the chances of returning a path that can be smoothed to optimal, the search has been interleaved with the smoothing algorithm. This is similar to the idea in [9]. The post dissociative smoothing is then applied to the path as a post process. Note that *PDS* can be applied iteratively from goal to start and vice versa until subsequent application does not shorten the path by a distance greater than δ (where δ is a user defined threshold). This results in *Interleave Smoothing with Iterative Post Dissociative Smoothing (IS-IPDS)*. To implement the interleave smoothing; replace the $\boldsymbol{UpdateNode}$ function of the $\boldsymbol{A_r^*}$algorithm with Algorithm 5.

9 SIMULATIONEXPERIMENT RESULTS

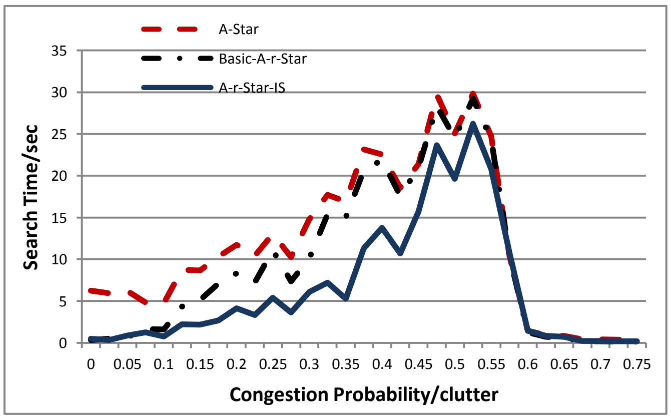
In this section,simulation experiments have been used to highlight the properties of the***Basic*** $\boldsymbol{A_r^*}$/$\boldsymbol{A_r^*}$ pathfinder and show its performance as compared to $\boldsymbol{A^*}$ on different world scenarios using both uniform and non-uniform gridding. The simulations were developed using MATLAB (2011b, The MathWorks) running on PC with the Windows 8 OS. The simulation world comprises a grid of size $256x256$ (which amounts to 65536 nodes). The performance parameters include: (a) ***Search Time***: the time it takes to plan a path; (b) ***Number of cells on OPEN list***: the total number of cells that ever made it to the OPEN list throughout the search; (c) ***Number of cells explored***: the number of cells that were actually explored before the goal was reached. In addition, example pathfinder applications to maze solving and indoor navigation are presented.

```
{38} $\mathbf{UpdateNode}(s,s')$
{39}     $\boldsymbol{if}\ LOS(parent(s),s')$
    {40} $\boldsymbol{if}\ g(parent(s)) + c(parent(s),s') < g(s')\ \boldsymbol{then}$
{41}     $g(s') := g(parent(s)) + c(parent(s),s');$
{42}         $parent(s') := parent(s);$
{43}         $f(s') := g(s') + c(s',s_{goal})$
{44}         $\boldsymbol{if}\ s' \in OPEN\ \boldsymbol{then}$
{45}             $OPEN.\mathbf{Remove}(s');$
{46}         $OPEN.\mathbf{Insert}(s',g(s'),f(s'));$
{47}     $\boldsymbol{else}$
{48} $\boldsymbol{if}\ g(s) + c(s,s') < g(s')\ \boldsymbol{then}$
{49}         $g(s') := g(s) + c(s,s');$
{50}         $parent(s') := s;$
{51}         $f(s') := g(s') + c(s',s_{goal})$
{52}         $\boldsymbol{if}\ s' \in OPEN\ \boldsymbol{then}$
{53}             $OPEN.\mathbf{Remove}(s');$
{54}         $OPEN.\mathbf{Insert}(s',g(s'),f(s'));$
```

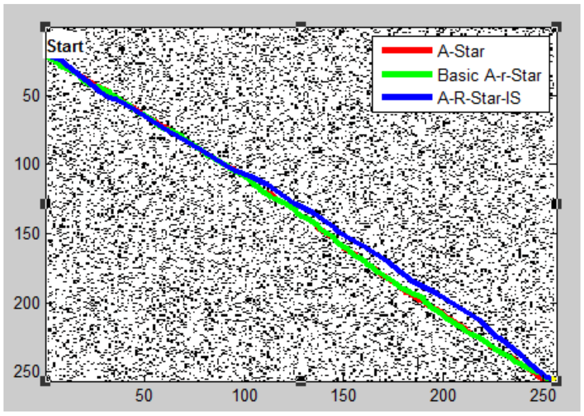
Algorithm 5: Interleave Smoothing (IS)

9.1 Effect of Congestion/Clutter on Performance of A-Star and Basic A-r-Star and A-r-Star

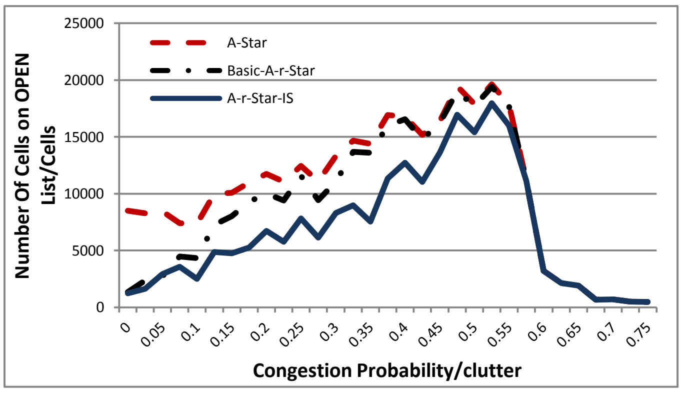
In the first experiment, the simulation environment was populated with obstacle nodes having congestion/clutter probability varied from 0 to 0.75, and with $s_{Start} = [1,20]$ and $s_{Goal} = [256,256]$. Results are shown in Figure 7. It was observed that no path existed beyond congestion probability of 0.6. Since over half of the nodes are occupied, it makes sense that searching from one extreme corner of the world to another will not have an unblocked path. Secondly, as the clutter increases, the number of free nodes decreases and this explains the sudden reduction in the graphs of performance parameter values after congestion probability of 0.55. The simulation results in Figure 7 demonstrate that***Basic*** $\boldsymbol{A_r^*}$and$\boldsymbol{A_r^*}$converge to $\boldsymbol{A^*}$beyond some degree of congestion (Theorem 2).



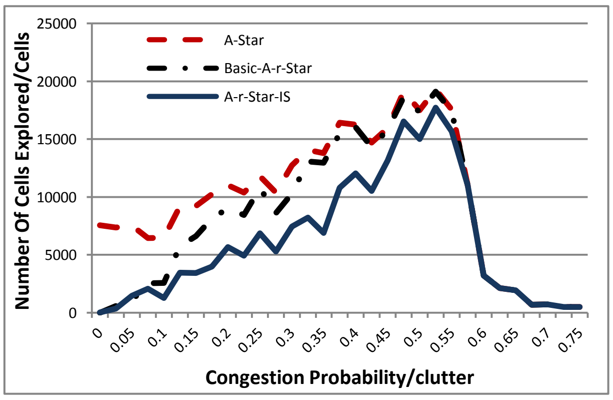
(a) Search Time



(b) Instance of the environment at clutter probability of 0.5



(c) Size of OPEN list

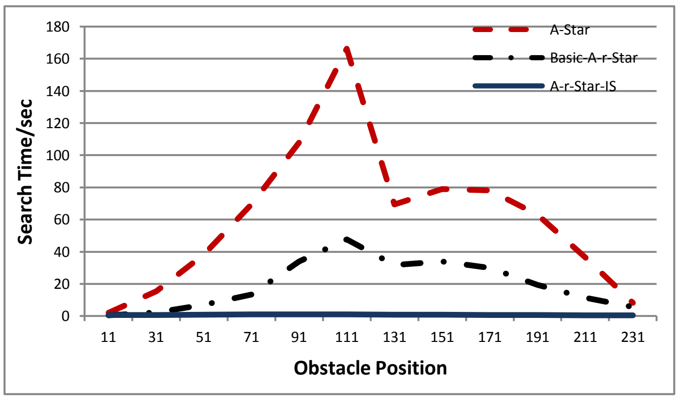


(d) Number of nodes explored

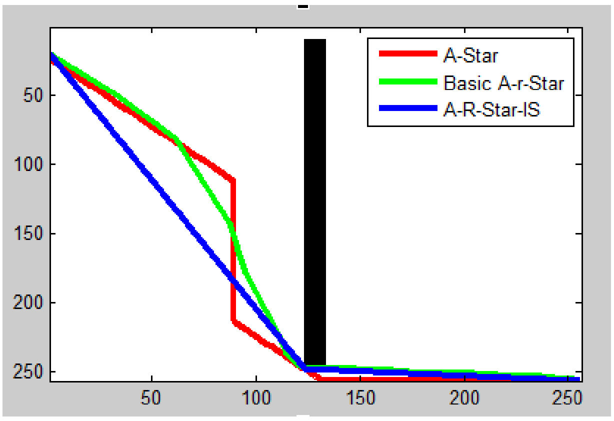
Figure 7: The effect of congestion/clutter on A^*, $\boldsymbol{Basic}\ A_r^*$ and A_r^* operating on a uniform grid world

9.2 Effect of Changing Obstacle Configuration on Performance of A-Star and Basic A-r-Star and A-r-Star (Sliding Obstacle)

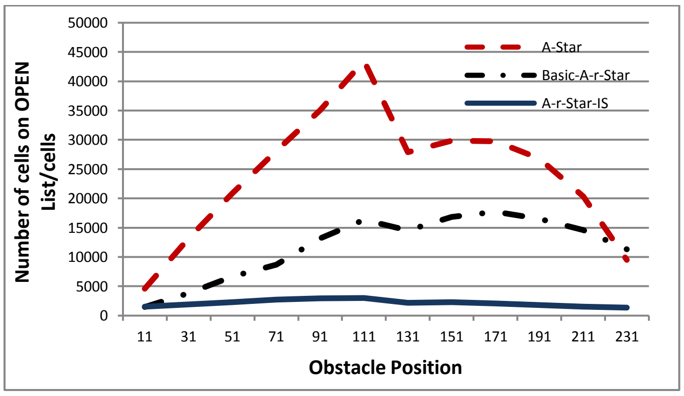
This simulation experiment shows thatchanging the obstacle configuration has little effect on $\boldsymbol{A_r^*}$performance whereas it can drastically degrade the performance of $\boldsymbol{A}^*$ operating in a sparse world such as the one shown in Figure 8 (b). The obstacle is assumed to be a long rigid wall in the environment separating the $s_{Start} = [1,20]$ and $s_{Goal} = [256,256]$. The horizontal position of this obstacle was varied from 11 to 231 and the performances of the pathfinders were recorded after each run.



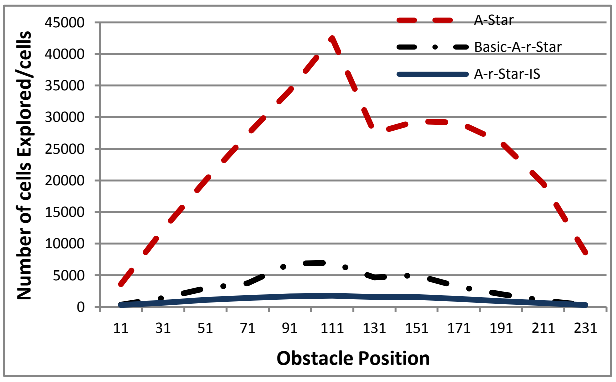
(a) Search Time



(b) Instance of the environment



(c) Size of OPEN list



(d) Number of nodes explored

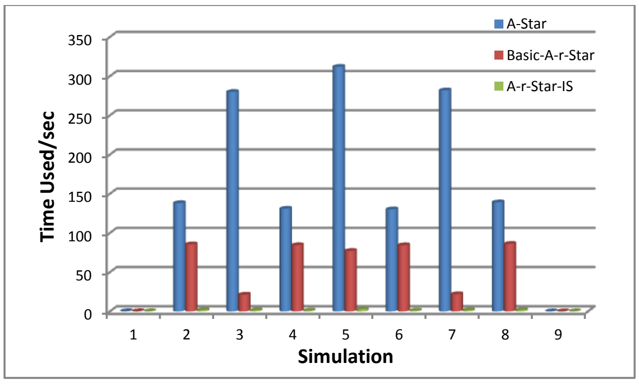
Figure 8: The effect of changing obstacle configuration on A^*, $\boldsymbol{Basic\ A_r^*}$ and $\boldsymbol{A_r^*}$ operating on a uniform grid world

9.3 Effect of Changing s_{Start} and s_{Goal} Configuration on Performance of A-Star and Basic A-r-Star and A-r-Star in the presence of a Concave Obstacle

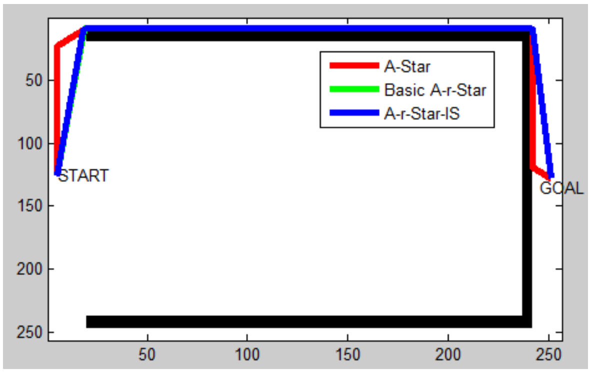
This simulation experiment shows that $\boldsymbol{A_r^*}$can better handle a large concave obstacle than$\boldsymbol{A}^*$. The obstacle is assumed to be a large rigid concave wall in the environment separating the s_{Start} and s_{Goal}. Table 1 shows the nine different combinations of s_{Start} and s_{Goal} used to generate the performance results shown in Figure 9. Different instances, where s_{Start} and s_{Goal} are either symmetrical or skewed towards one end of the obstacle were chosen as well as their combinations.

Table 1. The nine different Start and Goal combinations for the simulation in this section

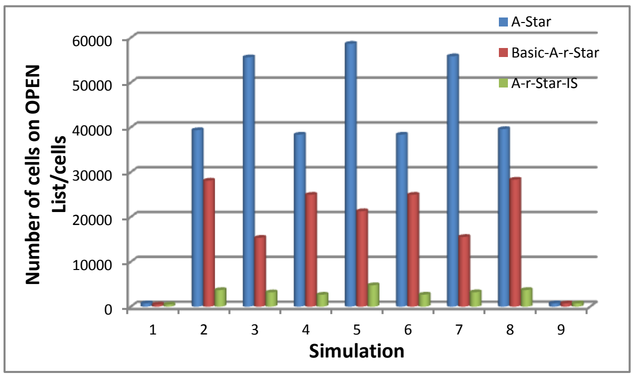
Simulation	START		GOAL	
	X	Y	X	Y
1	5	5	251	5
2	5	5	251	128
3	5	5	251	251
4	5	128	251	5
5	5	128	251	128
6	5	128	251	251
7	5	251	251	5
8	5	251	251	128
9	5	251	251	195



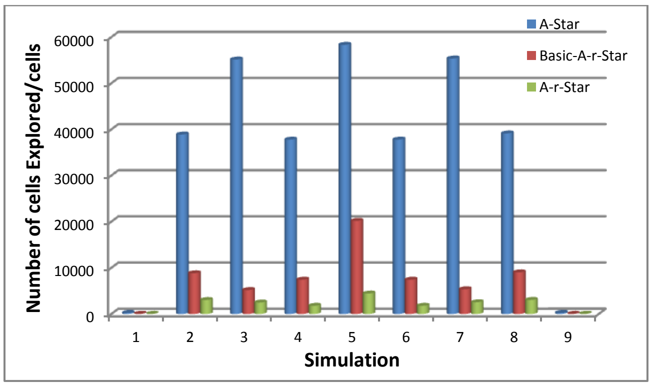
(a) Search Time



(b) Instance of the environment at simulation 5



(c) Size of OPEN list

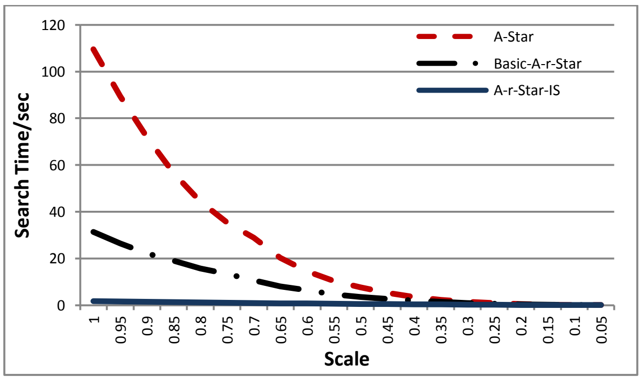


(d) Number of nodes explored

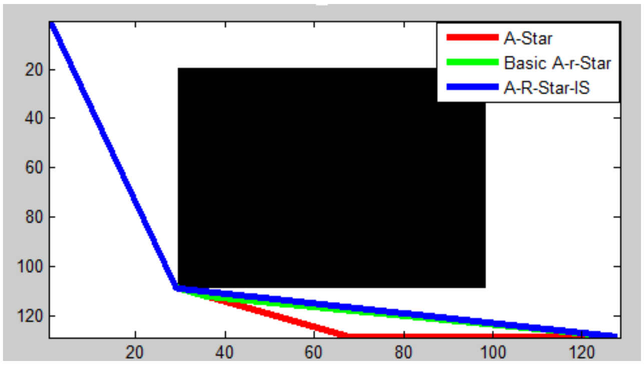
Figure 9: The effect of changing start and goal node position with respect to a large concave obstacle on A^*, $\boldsymbol{Basic\ A_r^*}$ and $\boldsymbol{A_r^*}$ operating on a uniform grid world

9.4 Effect of Increasing the Resolution of the Same Environment on Performance of A-Star, Basic A-r-Star and A-r-Star

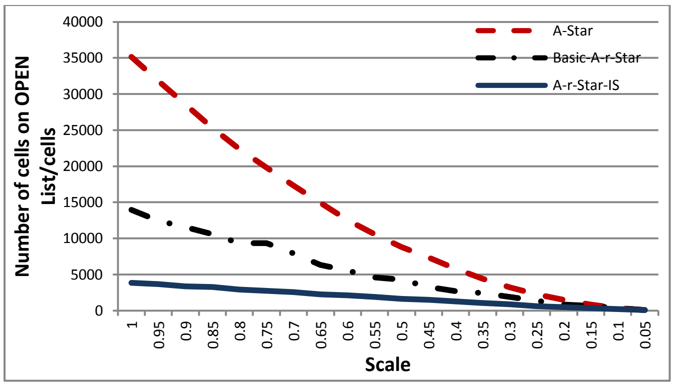
This simulation shows that increasing the resolution of the same environment degrades the performance of $\boldsymbol{A}^*$ exponentially but that of $\boldsymbol{A_r^*}$only degrades linearly. The obstacle is assumed to be a long rigid wall in the environment separating s_{Start} ands_{Goal}. The resolution of the grid was varied from $256x256$ to $10x10$. At each resolution, the performances of the pathfinders were recorded. This confirms the assertion in Section 2 that $\boldsymbol{A}^*$search time increases exponentially with increasing the grid size.



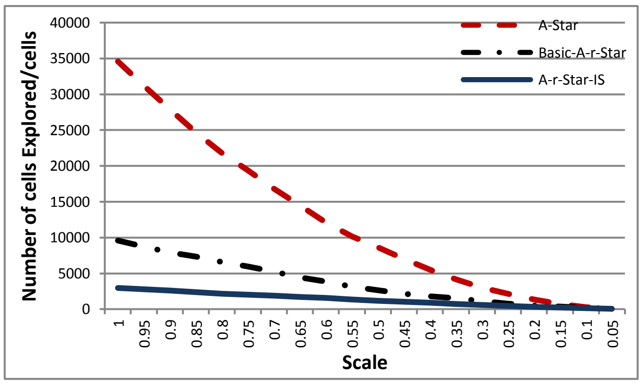
(a) Search Time



(b) Instance of the environment at resolution of $123x123$ (i.e. at scale 0.5)



(c) Size of OPEN list

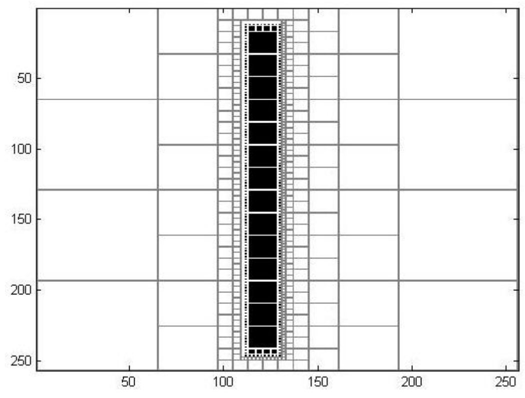


(d) Number of nodes explored

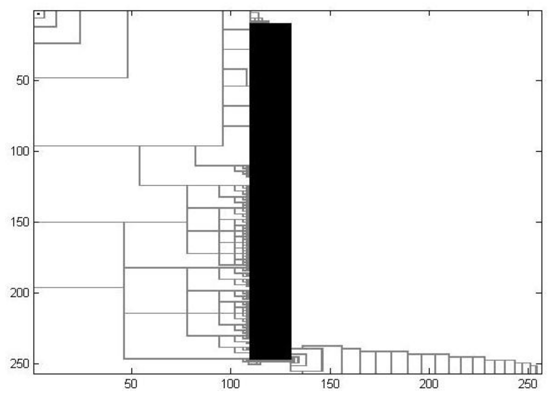
Figure 10: The effect of changing the gridding resolution of a given continuous world onA^*, $Basic\ A_r^*$ and A_r^* operating on a uniform grid world

9.5 Comparing the Performance of the A-r-Star with that of the A-Star Running on Quadtree

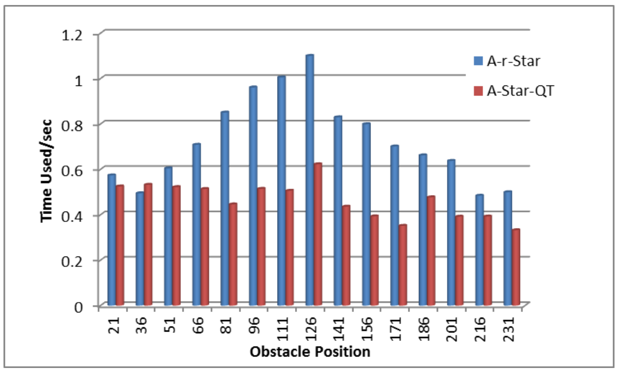
For this simulation experiment, an obstacle of size $(236\ x\ 20)\ nodes$ is placed between the start position $s_{Start} = [3,3]$ and the goal position $s_{Goal} = [254,254]$. An instance of the world after quadtree decomposition is shown in Figure 11 (a) and that after the A_r^*search in Figure 11 (b). The comparison in Figure 11 (c) and (d) shows that at certain obstacle configurations A^*running on an environment preprocessed into a quadtree almost always outperformsA_r^*;however, it must be noted that the preprocessing takes a longer time in the quadtree case. Besides, as highlighted above in Section 2 and in [4], the performance of the quadtree approach degrades drastically with increasing congestion.



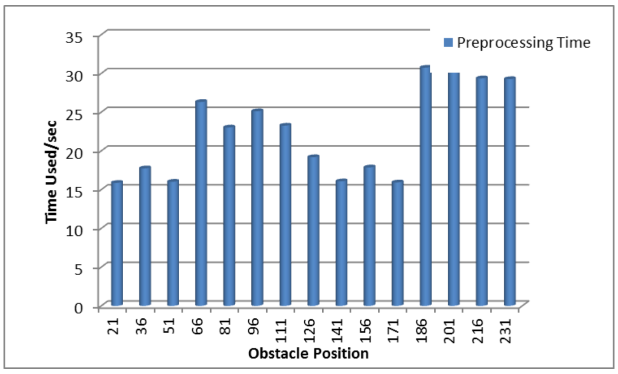
(a) The world after quadtree decomposition (preprocessing). White represents free nodes, gray represents node borders and black represents obstacle.



(b) The multi-resolution grid built by A-r-Star during the search. White represents free nodes, gray represents node borders and black represents obstacle.



(c) The search time comparison



(d) The preprocessing time for the quadtree

Figure 11: (a) Quadtree decomposition; (b) A_r^*–derived multi-resolution grid (c) performance comparison forA^* operating on a quadtree and A_r^* operating on a uniform grid of the same continuous environment; (d) associated quadtree processing time.

9.6 APPLICATION 1: Solving a Maze Problem with the A-Star and Basic A-r-Star and A-r-Star Algorithms

Artificial intelligence search algorithms are often used to solve maze problems that are common in tortuous games such as the *Pacman Maze Game*. Such maze problems are similar, and equivalent in some cases, to pathfinding problems faced by robots operating in maze-like environments such as building floorplans and underground mines, for example. The first application is to use the three pathfinder algorithms to solve a simple 256 x 256 maze problem. Figure 12 shows the maze and respective paths found between the indicated start and goal nodes. The performances of the algorithms are summarized in Table 2. Here, the path length is measure using the Euclidean distance. Note that the paths returned by $\boldsymbol{Basic\ A_r^*}$ and $\boldsymbol{A_r^*}$ have *bulges* that make the path suboptimal. These bulges can be eliminated using the techniques proposed in Section 9, as shown in Figure 6.

Table 2. The Performance Comparison of the Three Algorithms for the Maze Problem Solving Problem

Algorithm	**Time Used (sec)**	**Path Length (units)**	**Number of cells on Open List (cells)**	**Number of cells Explored (cells)**
A-Star	122.45	779.39	37378	37142
Basic-A-r-Star	102.62	795.77	33011	31581
A-r-Star-IS	17.47	789.56	14980	14675

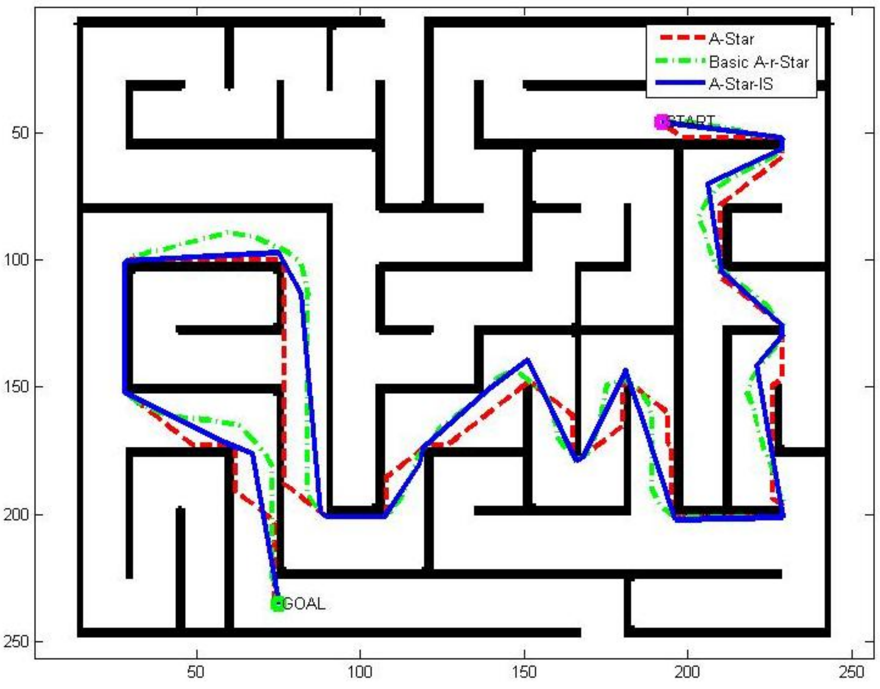
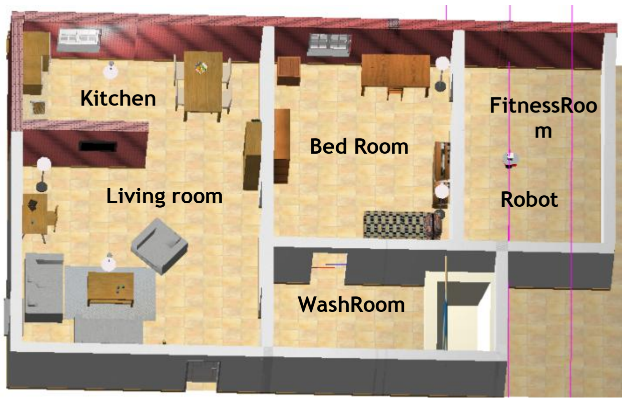


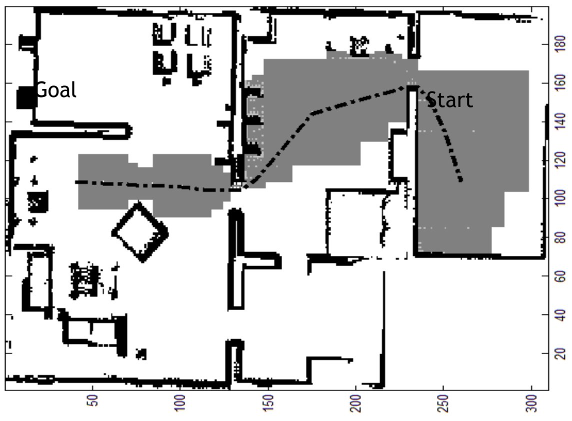
Figure 12: How A^*, $\boldsymbol{Basic\ A_r^*}$ and $\boldsymbol{A_r^*}$ operating on a uniform grid world solves a maze problem

9.7 APPLICATION 2: Path planning in a Simulated Home using A-r-Star Pathfinder

The next application involves using$\boldsymbol{A_r^*}$ for path finding in a simulated 3D home environment which was developed using Webots as shown in Figure 13 (a). Webots [21] is commercial software for robotic systems prototyping and simulation. A prototype of the Pioneer 2DX robot was run in this environment to build a binary occupancy grid map using a simulated SICK Laser Measurement Sensor (LMS) 200. (The prototype for the robot and sensor come with Webots.) A 2D map representing the floorplan of the home environment is then fed to $\boldsymbol{A_r^*}$ to plan a path from a point in the fitness room to a destination in the living room. The result is shown in Figure 13 (b).



(a) The Webots Prototype of the home environment



(b) A binary occupancy grid map of the environment showing the path (center line), the obstacle zone (black nodes), the skip zones (gray nodes) and the obstacle-free zone (white nodes).

Figure 13: Path planning in a prototype home environment using the A_r^* pathfinder

10 CONCLUSION AND FUTURE WORK

This paper presents a new pathfinder called $\boldsymbol{A_r^*}$ for offline path planning that outperforms the $\boldsymbol{A}^*$ pathfinder in a uniform gridded sparse world. It also presents various demonstrations of some of the desirable properties of this algorithm and provesthat the performance of this new algorithm matches that of the $\boldsymbol{A}^*$pathfinder running on a quadtree decomposed world. More research is being performed to extend the algorithm to operate on non-binary grid worlds and to develop an incremental version of this algorithm, and the results will be compared with the performance of the incremental pathfinders ($\boldsymbol{D}^*$)and its variants.

11 ACKNOWLEDGMENT

This material is based in part upon work supported by the National Science Foundation (NSF) under Cooperative Agreement No. DBI-0939454. Any opinions, findings and conclusions are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

12 REFERENCES

[1] Cikes, M., Dakulovic, M., and Petrovic, I. 2011. The path planning algorithms for a mobile robot based on the occupancy grid map of the environment; A comparative study.In Proceedings of the XXIII International Symposium on Information, Communication and Automation Technologies, 1-8.

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

[3] Dechter, R. and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. J. ACM 32 (3), 505-536.

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

[5] Verwer, B. J. H. 1990. A multiresolution work space, multiresolution configuration space approach to solve the path planning problem.In Proceedings of the IEEE International Conference on Robotics and Automation 2103, 2107-2112.

[6] Botea, A., Muller, M., and Schaeffer, J. 2004. Near optimal hierarchical path-finding. Computers and Games 3, 33–38.

[7] Fredman, M. L. and Willard, D. E. 1993. Surpassing the information theoretic bound with fusion trees. Journal of Computer and System Sciences 47 (3), 424-436.

[8] Cowlagi, R. V. and Tsiotras, P. 2012. Multi-resolution path planning: theoretical analysis, efficient implementation, and extensions to dynamic environments. In Proceedings of the49th IEEE Conference on Decision and Control. 1384-1390.

[9] Daniel, K., Nash, A., Koenig, S., and Felner, A. 2010. “Theta*: any-angle path planning on grids. Journal of Artificial Intelligence Research 39, 533-579.

[10] Stentz, A. 1995. Optimal and efficient path planning for unknown and dynamic environments. International Journal of Robotics & Automation 10 (3), 89-100.

[11] Stentz, A. 1994. Optimal and efficient path planning for partially-known environments.In Proceedings of the IEEE International Conference on Robotics and Automation 3314, 3310-3317.

[12] Stentz, A. 1995. The Focussed D* algorithm for real-time replanning.In Proceedings of the International Joint Conference on Artificial Intelligence.

[13] Koenig, S. and Likhachev, M. 2002. D*lite.In Proceedings of the Eighteenth National Conference on Artificial Intelligence, Edmonton, Alberta, Canada, 476-483.

[14] Ferguson, D. and Stentz, A. 2007. Field D*: an interpolation-based path planner and replanner. In Thrun,

S., Brooks, R. and Durrant-Whyte, H., eds., Robotics Research. Springer Tracts in Advanced Robotics,Springer Berlin Heidelberg, 239-253.

[15] Carsten, J., Ferguson, D., and Stentz, A. 2006. 3D Field D: improved path planning and replanning in three dimensions.In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. 3381-3386.

[16] Shih-Ying, C., Tsui-Ping, C., and Zhi-Hong, C. 2012. An efficient theta-join query processing algorithm on MapReduce framework.In Proceedings of the International Symposium on Computer, Consumer and Control . 686-689.

[17] Aizawa, K. and Tanaka, S. 2009. A constant-time algorithm for finding neighbors in quadtrees.IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (7), 1178-1183.

[18] Schroeder, W. J., Zarge, J. A., and Lorensen, W. E. 1992. Decimation of triangle meshes. SIGGRAPH Comput. Graph. 26(2), 65-70.

[19] Huang, C. T. and Mitchell, O. R. 1994. A Euclidean distance transform using grayscale morphology decomposition.IEEE Transactions on Pattern Analysis and Machine Intelligence 16 (4), 443-448.

[20] Chung, K.-L., Huang, H.-L., and Lu, H.-I. 2004. Efficient region segmentation on compressed gray images using quadtree and shading representation. Pattern Recognition 37 (8), 1591-1605.

[21] Michel, O. 2004. Webots: professional mobile robot simulation. International Journal of Advanced Robotic Systems 1 (1), 39-42.