Chinese Journal of Aeronautics, (2016), 29(5): 1385–1396

Chinese Society of Aeronautics and Astronautics
& Beihang University

Chinese Journal of Aeronautics

cja@buaa.edu.cn
www.sciencedirect.com

Rectangle expansion A* pathfinding for grid maps

Zhang An [a,b,*], Li Chong [b], Bi Wenhao [b]

[a] School of Aeronautics, Northwestern Polytechnical University, Xian 710129, China
[b] School of Electronics and Information, Northwestern Polytechnical University, Xian 710129, China

Received 25 December 2015; revised 2 March 2016; accepted 22 March 2016
Available online 27 August 2016

KEYWORDS

Breaking path symmetries;
Grid;
Heuristic algorithms;
Path search;
Variant of A*

Abstract Search speed, quality of resulting paths and the cost of pre-processing are the principle evaluation metrics of a pathfinding algorithm. In this paper, a new algorithm for grid-based maps, rectangle expansion A* (REA*), is presented that improves the performance of A* significantly. REA* explores maps in units of unblocked rectangles. All unnecessary points inside the rectangles are pruned and boundaries of the rectangles (instead of individual points within those boundaries) are used as search nodes. This makes the algorithm plot fewer points and have a much shorter open list than A*. REA* returns jump and grid-optimal path points, but since the line of sight between jump points is protected by the unblocked rectangles, the resulting path of REA* is usually better than grid-optimal. The algorithm is entirely online and requires no offline pre-processing. Experimental results for typical benchmark problem sets show that REA* can speed up a highly optimized A* by an order of magnitude and more while preserving completeness and optimality. This new algorithm is competitive with other highly successful variants of A*.

1. Introduction

Pathfinding is a basic problem in many domains, particularly robotics, artificial intelligence planning, navigation and commercial computer games. A great deal of research has been carried out to improve the quality of resulting paths while keeping costs low. The A* algorithm[1] is now regarded as the gold standard for search algorithms because of its completeness, optimality and effectiveness, and many of the current state-of-the-art pathfinding algorithms are variants of A*.

Search speed, quality of resulting paths and cost of pre-processing are the main evaluation metrics of a pathfinding algorithm. Typical speed enhancements for pathfinding usually involve either trading away optimality for speed or offline pre-processing. The former makes it difficult to guarantee the quality of results. The latter requires extra memory and pre-computing time as seen in hierarchical path-finding A* (HPA*),[2] swamp A*,[3] compressed path databases (CPDs),[4–7] subgoal graphs A* (SUB),[8] and TRANSIT A*.[9–11] Compared with exclusively online algorithms, those employing pre-processing will usually cause poorer performance in non-static environments. On the other hand, algorithms such as theta A*,[12,13] lazy theta A*,[12,14] and A* with post-smoothed paths (A* PS)[15] try to straighten the path during search or as

* Corresponding author. Tel.: +86 29 88431269.
E-mail address: zhangan@nwpu.edu.cn (A. Zhang).

Peer review under responsibility of Editorial Committee of CJA.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.cja.2016.04.023

part of the post-processing step to obtain better than grid-optimal results and avoid constraining paths to grid edges. This involves expensive visibility tests and sacrifices speed.

In this paper a new optimal algorithm, rectangle expansion A* (REA*), is introduced that explores maps units of unblocked rectangles. Without any pre-processing, REA* can speed up a highly optimized A* by an order of magnitude and more on typical benchmark problem sets. REA* executes purely grid-optimal pathfinding, but the additional benefit of rectangle expansion usually makes the final path better than grid-optimal. This paper makes the following contributions:

(1) A detailed description of the REA* algorithm for 8-connected grid maps.
(2) A theoretical proof for the completeness, optimality of REA* and analysis of its effectiveness.
(3) Trade-off measures to further enhance the method.
(4) Experimental results of REA* and comparisons with standard A* and some of its highly successful variants.

2. Related work

The concept of eliminating unnecessary symmetry paths on grid maps has been seen in literature in the last few years, and REA* is inspired by some of the recent successes. The idea of improving a path by reducing unnecessary heading changes also benefits REA*.

REA* is similar to rectangular symmetry reduction (RSR)[16–18] because both divide grid maps into several rectangles free of interior obstacles, and all interior nodes of each rectangle will be pruned during path search. However, they are essentially different. RSR replaces interior nodes with a series of macro-edges, and nodes are visited and expanded individually during search. In REA*, each rectangle is operated in as a whole, ignoring all unnecessary interior connections. Outer perimeter nodes, continuously free of obstacles, will be considered "search nodes", which means fewer and quicker list operations. What's more, REA* is carried out entirely online while RSR requires offline pre-processing to divide grid maps into rectangles and assign nodes.

Block A*[19] is another algorithm that searches using rectangles. Information about all possible distances across a block of grid cells ($m \times n$ region of nodes) is pre-computed and stored in a new type of database, the local distance database (LDDB). Distances between boundary points are queried from the LDDB during an online A* search. The number of entries in the LDDB increases extremely fast, so the size of blocks cannot be large (no more than 5×5 in Ref. 19). Block A* requires to pre-compute the LDDB and, though different LDDBs can be chosen to improve the resulting path, it is only guaranteed to be optimal in 4-connected grid maps.

Jump point search (JPS)[20,21] is the current state-of-the-art online algorithm. JPS identifies and selectively expands only certain nodes in a grid map called jump points. All intermediate nodes on paths connecting two jump points are never expanded. The algorithm can speed up A* greatly and, just like REA*, returns skipped path points. However, while JPS guarantees grid-optimal paths, REA* returns paths better than grid-optimal.

Anya A*[22] is a recently proposed algorithm for grid maps that uses contiguous sets of nodes in horizontal or vertical lines as search nodes. Anya A* expands search nodes from each line to a neighboring line with frequent, expensive line-of-sight tests while REA* expands search nodes in units of rectangles, a much cheaper operation. But, Anya A* offers any-angle optimal pathfinding that is not artificially constrained to the points of a grid.

3. Rectangle expansion A* algorithm

A grid map is one of the most popular types of maps used to represent realistic terrain in literature. Assuming that (in 8-connected grid maps) an agent operates on a grid map with obstacles consisting of blocked cells and traversable areas consisting of unblocked cells, it can move from any unblocked grid center to another cardinally or diagonally if both adjacent cardinal directions are unblocked.

REA* is a variant of standard A*. Pseudo-code of REA* is shown in Table 1. The octile distance is used to estimate the distance between two cells heuristically. The respective lengths of cardinal and diagonal moves are 1 and 1.414. A matrix the same size as the map is used to store all the grid points. Point (x, y) represents the point at the intersection of xth column and yth row in the grid map with (1, 1) as the point in the upper left corner. The octile distance between $p(x, y)$ and $p'(x', y')$ is:

$$\text{octile}(p, p') = 1.414 \times \min(\Delta x, \Delta y) + |\Delta x - \Delta y| \quad (1)$$

where $\Delta x = |x - x'|$ and $\Delta y = |y - y'|$.

Definition 1. A grid interval I is a set of contiguous points in the same row or column of the grid. If all points in I are unblocked, I is an unblocked interval. Each interval can be defined in terms of its endpoints a and b, written as $[a, b]$.

Search nodes in REA* are not individual cells in the grid map but unblocked intervals of the map. To distinguish them from our search nodes, traditional individual grid cells will be called cells or points in this paper for simplicity. The key idea of REA* is that, when exploring a grid map, REA* doesn't visit individual cells one by one. Instead, a linear search node will expand an unblocked rectangle until stopped by obstacle cells. Only interior boundary cells of the expanded unblocked rectangle will be visited by parent cells from the original search node, and unblocked points inside the rectangle will be pruned. Then, interior boundaries of the rectangle (except the original

Table 1 REA*() algorithm.

Require:	S: the start point, G: the goal point;
1:	Initialize();
2:	**if** InsertS() **then**
3:	**return** "path is found";
4:	**While** (Openlist! = $\varnothing$)
5:	CBN:= the current best search node;
6:	**if** Expand (CBN) **then**
7:	**return** "path is found";
8:	**return** "no path is found";

search node) are "pushed outward" to generate successor search nodes, which are unblocked intervals from outer boundaries of the original rectangle. Successor search nodes will be inserted into the open list with the minimum f value of cells in the search node as its priority. The current best search node is fetched and the above process is repeated until an optimal path is found. Unblocked rectangles in REA* can be seen as rooms protecting the path from obstacles, and successor search nodes are the doors to next rooms.

Fig. 1(a) shows the result of A* in a simple instance with red representing closed points and gray representing open points. Fig. 1(b) shows the status of REA* after inserting the start point onto the map. Fig. 1(c) shows the result of expanding beyond the first unblocked rectangle. Fig. 1(d) is the final result of REA* in the same instance. All points in REA* are newpoints when a new pathfinding task begins, and member parameters (for example, the *gval* and *hval*) of a newpoint are regarded as invalid. The red points in Fig. 1(b)–(d) are those that constitute search nodes of REA*, which are called *hpoints*. The gray are those visited by REA* but not contained by any search node, which are called *gpoints*. Since most points visited by REA* will not ultimately become *hpoints*, in most instances (except for the original rectangle expanded by the start point), the *gval* of a point can be calculated through addition. REA* assigns a *gpoint* with a *gval* only while the *hval*, which involves multiplication in octile distance, is omitted for efficiency.

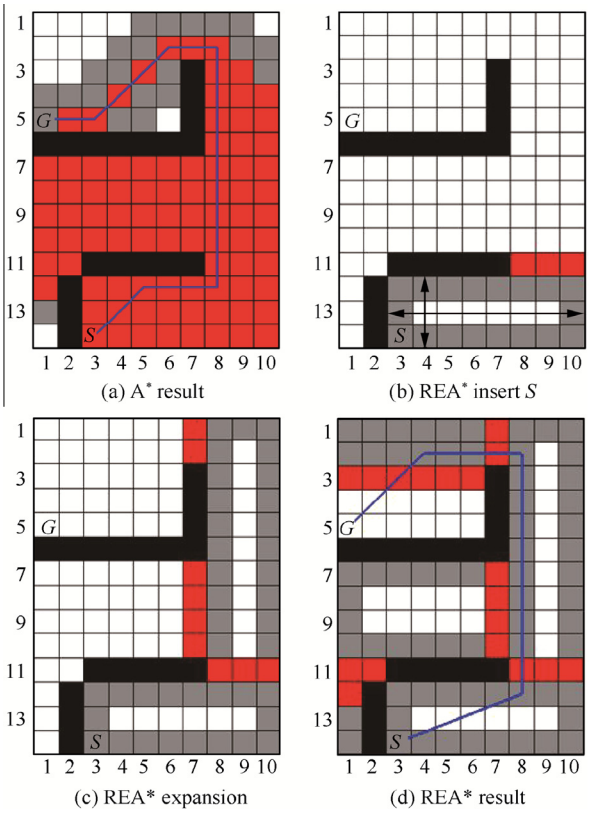


Fig. 1 Simple example of A* and REA*.

3.1. Insert start point on map

Pseudo-code to insert start point on map is shown in Table 2. The start point S can be seen as the first search node of REA* and expands in both the horizontal and vertical directions to generate the original unblocked rectangle. For example, in Fig. 1(b) S first expands vertically until stopped by the map boundary and blocked cells in row 11, then the vertical axis scans the map horizontally until stopped by the map boundary and blocked cells in column 2. The unblocked rectangle [(3, 12), (10, 12), (10, 14), (3, 14)] is the original rectangle of REA*. The order of vertical expansion and horizontal expansion may affect the original rectangle, but the completeness and optimality of the algorithm is impervious.

If the goal point G is within the original rectangle, the optimal path will be the line-of-sight between S and G, and REA* will terminate successfully.

Otherwise, all points in the interior boundaries will become *gpoints*, with S as the parent and the octile distance from S as the *gval*. Then all the boundaries (as intervals, not the individual grid cells) will be used to generate the first successor search nodes (details in Section 3.2) whose expansion directions are just the relative position of their parent boundaries in the parent rectangle. For example, in Fig. 1(b) the North boundary [(3, 12), (10, 12)] and West boundary [(3, 12), (3, 14)] will be the parents of the first search nodes. Notice that the point (3,12) is contained in both intervals. The map boundaries [(3, 14), (10, 14)] and [(10, 12), (10, 14)] are abandoned because their successor search nodes traverse beyond the map.

3.2. Generate successor search node

Definition 2. The extend neighbor interval (ENI) of interval I is calculated as Eq. (2), where corner points of I are NW for northwest, NE for northeast, SW for southwest and SE for southeast. DI is the expansion direction of I.

$$\mathrm{ENI} = \begin{cases} [\mathrm{NW}+(-1,-1), \mathrm{NE}+(1,-1)] & \text{if DI} = \text{North} \\ [\mathrm{SW}+(-1,1), \mathrm{SE}+(1,1)] & \text{if DI} = \text{South} \\ [\mathrm{NW}+(-1,-1), \mathrm{SW}+(-1,1)] & \text{if DI} = \text{West} \\ [\mathrm{NE}+(1,-1), \mathrm{SE}+(1,1)] & \text{if DI} = \text{East} \end{cases} \tag{2}$$

Table 2 InsertS() algorithm.

Require:	S: the start point, G:the goal point;
1:	RECT:= original rectangle expanded by S;
2:	**if** $G \in$ RECT **then**
3:	**return** "path is found";
4:	**for all** point $p \in$ boundaries of RECT **do**
5:	$p.gval$:= octile(p,S);
6:	$p.mode$:= *gpoint*;
7:	**for all** valid boundaries of RECT **do**
8:	**if** Successor(boundary) **then**
9:	**return** "path is found";
10:	**return** NULL;

Definition 3. Possible blocked member points may divide an ENI into several unblocked subintervals that are neither adjacent nor overlap. Each of these unblocked subintervals is a free subinterval (FSI) of the ENI. If an ENI is totally unblocked, it is the only free subinterval of itself.

Definition 4. A search node (*I*, *dir*, *minfval*) is a tuple where *I* is an unblocked interval and *dir* is the expansion direction of *I* and *minfavl* is the minimum *fval* of points in *I*.

Pseudo-code to generate successor search node is shown in Table 3. Assume that the ENI is of a valid boundary (interval PB) of the parent rectangles in given expansion direction, and the FSI is a free subinterval of ENI. Then, for each point $p \in$ FSI, if p is newpoint or if there exists a shorter path from PB to p, that is, $p.gval >$ pb.$gval$ + 1(or 1.414 according to the relative position of p and pb; pb is a reachable neighbor point of p in PB), p will be updated by the smaller *gval* and a new parent point pb.

If any point in an FSI is updated, a new successor search node PN will be generated and inserted into the open list. The FSI will be the unblocked interval of PN, and all points in the FSI will become *hpoints*, with octile distance octile(p, G) as the *hval*:

$$p.fval = p.gval + p.hval = p.gval + \text{octile}(p, G) \tag{3}$$

Expansion direction of PN depends on the position of PB in the parent rectangle, and the minimum *fval* of points in an FSI will be the *minfval* of the new search node PN. Notice that the *minfval* represents the best condition of points in the FSI when PN is generated. Points in an FSI can also be updated by other search nodes, and when an FSI is expanded it will always use the latest and current best status of points. If an FSI does not exist, or if none of the member points in an FSI can be updated, no successor search node will be generated.

A search node is in fact the entrance of a new unblocked rectangle. Fig. 2 shows how the original unblocked rectangle generates the first search nodes in Section 3.1. Since point (11,11) and point (2,15) are off the map, interval [(2,11), (10,11)] will be the extend neighbor interval of the north boundary with free subintervals [(2,11), (2,11)] and [(8,11), (10,11)], and interval [(2,11), (2,14)] will be the extend neighbor interval of the west boundary with free subinterval [(2,11), (2,11)]. Since point (2,11) is not reachable directly because of the obstacles, subinterval [(8,11), (10,11)] will be the only successor search node of the original unblocked rectangle with local optimal parents from the north boundary.

If goal point G belongs to ENI and is updated with a *fval* not greater than the *minfval* of the search node PP, which expands the parent rectangle, REA* will terminate successfully with the optimal path.

Table 3 Successor() algorithm.

Require:	PB: boundary of parent rectangle; PP: parent search node of PB; PN: a new search node;
1:	ENI: = extend neighbor interval of PB;
2:	**for all** free subintervals FSI of ENI **do**
3:	**for all** point $p \in$ FSI **do**
4:	Try to update p with PB
5:	$p.mode$: = *hpoint*;
6:	$p.hval$: = octile(p,*G*);
7:	$p.fval$: = $p.hval$ + $p.gval$;
8:	**if** $G \in$ FSI and
9:	$G.fval \leqslant$ PP.*minval* **then**
10:	**return** "path is found";
11:	**if** $\exists\ p \in$ FSI updated by PB **then**
12:	PN.*I*: = FSI;
13:	PN.*dir*: = direction of PB;
14:	PN.*minfval*: = $\min_{pf \in FSI} pf.gval$;
15:	Openlist.Insert(PN);
16:	**return** NULL;

3.3. Expand unblocked rectangle

The search node with minimum *minfval* is called the current best search node (CBN). The CBN is fetched from the open list and expands the new unblocked rectangle in its direction until stopped by blocked cells. The current best search node can be seen as the entrance of the new rectangle (or room) and the other three interior boundaries can be seen as the walls.

If the goal point G belongs to the CBN or G is within an unblocked rectangle, the point with the minimum *fval* in the CBN will be the parent of G and an optimal path will be found. Otherwise REA* only calculates the local minimum *gval* for points on the walls, and all interior points of the room will be pruned.

To get the local minimum *gval* of a point on the wall, REA* does not need to calculate the octile distances between the point and all entrance points. Fig. 3 shows an example when CBN.*dir* = North and blocked points are not shown for simplicity. The red interval in Fig. 3 is the CBN and gray points are the wall points. REA* will only consider the potential parents from the CBN, which are between the southeast diagonal and the northwest diagonal of the wall points.

Assume W is the western interior boundary of the unblocked rectangle. The letters in the center of points in Fig. 3(a) are the order that REA* deals with points in W. First, to determine the local minimum *gval* and local optimal parent, point A examines its southern neighbor point (1,5) with vertical distance 1 and its southeast diagonal point (2,5) with initial diagonal octile distance 1.414. Then, point B examines its own southern neighbor point A with vertical distance 1 and its southeast diagonal point (3,5) with diagonal octile distance 1.414 larger than the last diagonal octile distance. Since point A is local optimal and an octile path or equal octile path must pass point A for all points west of point (3,5), it is easy to understand that point B will be local optimal. This process repeats and potential parents of each point in W are shown with arrows in Fig. 3(a). Until all points in W are visited, the

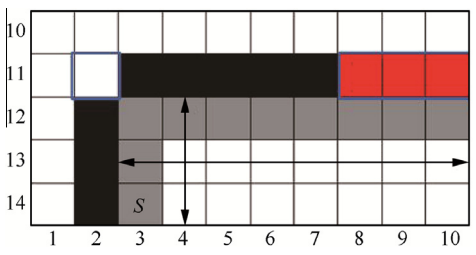
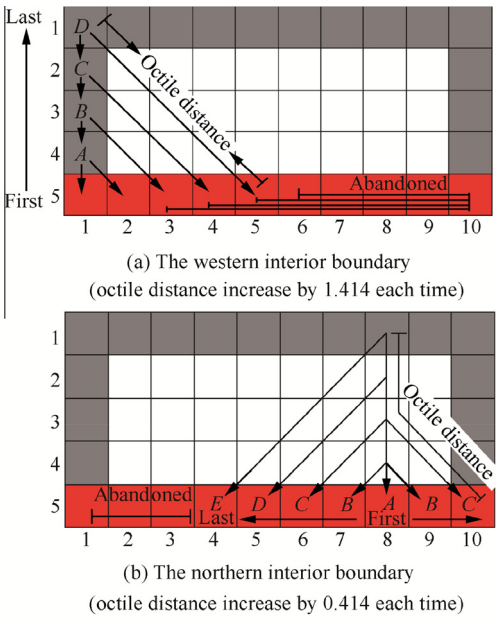


Fig. 2 Generating the first search nodes in Fig. 1.



(a) The western interior boundary
(octile distance increase by 1.414 each time)

(b) The northern interior boundary
(octile distance increase by 0.414 each time)

Fig. 3 Example of expanded unblocked rectangle when CBN. *dir* = North.

current wall point will move to the north by one point and increase the diagonal octile distance by 1.414 each time if the southeast diagonal point is not beyond the CBN. All points in W will be local optimal if points east of the southeast diagonal point of the current wall point are abandoned. The absence of an eastern part of the CBN will not destroy the optimality of REA* (proved in Section 4) and this process will need no multiplication.

A wall point will follow its parents from itself to S and choose the last point within the current unblocked rectangle as its immediate parent, which will skip unnecessary in-between points and straighten the path. The process to deal with the eastern interior boundary of the unblocked rectangle is similar and omitted here.

Assume N is the northern interior boundary of the unblocked rectangle. Point (8, 1) is a point in N, as seen in Fig. 3(b). Point (8, 1) examines the southern boundary point (8, 5), which is in the same column, and the initial octile distance is the distance from N to the CBN (5–1 = 4). Then, the potential parents move west and east by one point (if not beyond the CBN) and the octile distance increases by 0.414 each time until all potential parents between the southeast diagonal and the northwest diagonal of point (8, 1) have been examined. This process will also not need multiplication.

Points in the walls will become *gpoints* if they are originally not *hpoints* before this expansion, and the three interior boundaries (walls) will generate new search nodes. The successor search nodes generated can be seen as the exit or outer boundary of the current rectangle, which will also be the entrance of a new unblocked rectangle. Pseudo-code to expand unblocked rectangle is shown in Table 4.

3.4. Terminal conditions

After start point S is inserted into the map and the first search nodes are generated, the main loop of REA* will fetch the current best search node CBN in the open list, expand the new unblocked rectangle, and generate successor search nodes in each cycle.

If the open list is empty and there is no new CBN, no traversable grid path between S and G exists. Otherwise an optimal path will be found and REA* will terminate in one of these three conditions:

(1) The goal point G belongs to the CBN.
(2) The goal point G belongs to the unblocked rectangle of the CBN. The point with the minimum *fval* in the CBN will be the parent of G in this condition.
(3) The goal point G belongs to a successor search node of the CBN and $G.fval \leqslant$ CBN.*minfval*.

REA* will follow the parents from G to S to retrieve the optimal path. REA* guarantees grid-optimal path points, but

Table 4 Expand() algorithm.

```
Require: CBN: the current best search node
1:  if G ∈ CBN then
2:    return "path is found";
3:  /*only for CBN.dir = North, and other case is omitted here*/
4:  RECT:= rectangle expanded by CBN;
5:  if G ∈ RECT then
6:    return "path is found";
7:  W:= western interior boundary of RECT;
8:  pw:= the most west point in CBN;
9:  diagonal:= 1.414;
10: p:= pw+(0, −1);
11: pv′:= pw;
12: pd′:= pw+(1, 0);
13: while (p ∈ W)
14:   octile(p, pv′):= 1;
15:   octile(p, pd′):= diagonal;
16:   Try to update p with pv′ and pd′:
17:   if p.mode≠hpoint then
18:     p.mode:= gpoint;
19:   diagonal:= diagonal + 1.414;
20:   p:= p+(0, −1);
21:   pv′:= pv′ + (0, −1);
22:   pd′:= pd′+(1, 0);
23: if Successor(W) then
24:   return "path is found";
25: /*eastern interior boundary is omitted here*/
26: N:= north interior boundary of RECT;
27: for all p ∈ N do
28:   dis:= CBN.row-N.row;
29:   p1′:= p2′:=p+(0, CBN.row-N.row);
30:   while(dis ⩽ 1.414 × (CBN.row-N.row))
31:     octile(p, p1′):= dis;
32:     octile(p, p2′):= dis;
33:     Tyr to update p with p1′ and p2′;
34:     if p.mode≠hpoint then
35:       p.mode:=gpoint;
36:     dis:= dis + 0.414;
37:     p1′:= p1′+(−1,0);
38:     p2′:= p2′+(1,0);
39: if Successor(N) then
40:   return "path is found";
41: return NULL;
42:
```

since path points of REA* are either adjacent or skipped but protected by an unblocked rectangle, the straight lines between path points will be safe and shorter than grid-optimal without any extra cost.

4. Optimality

Assuming path $\pi = \{p0, p1, \ldots, pn\}$ is an optimal grid path with $p0 = S$ and $pn = G$ and SN_0 is the search node generated by the original rectangle and from which π leaves the original rectangle.

Definition 5. A point p is optimal if $p.gval$ equals the length of the grid-optimal path from start point S to p. A search node SN is optimal if there exists a point $p \in SN$ satisfying:

(1) $p \in \pi$.
(2) p is optimal.
(3) $|p.gval - p'.gval| \leqslant octile(p, p') \forall p' \in SN$.

It is obvious that SN_0 is optimal, because for any p and p' in SN_0, $p.gval = octile(S,p)$, $p'.gval = octile(S,p')$ and $|octile(S,p) - octile(S,p')| \leqslant octile(p,p')$ is satisfied according to triangle inequality.

Lemma 1. *Whenever an optimal search node is expanded, the optimal path will be found or a new optimal search node will be generated whose minfval will be not greater than π.length. The optimal point in the new optimal search node has greater gval than that in the old optimal search node.*

Proof. Assume SN is an optimal search node with pi the optimal point in the optimal path π and SN.dir = North. Assume also that RECT is the unblocked rectangle of SN and π leaves RECT from a wall point pj and enters into the next unblocked rectangle at point pk. NW is the northwest corner point of SN and NE is the northeast. PSN is the interval [NW + (1,1), NE + (−1,1)] and it is obvious that PSN is the minimal area in the north boundary of the SN parent rectangle and PSN is totally unblocked according to Definition 2. □

Assuming pp is the parent point of pi, pp must belong to the interval [NW + (−1,1), NE + (1,1)]. pk cannot belong to [NW + (0,1), NE + (0,1)] because the path from pp to pk through pi must be non-optimal when PSN is unblocked. Since RECT is surrounded by [NW + (0,1), NE + (0,1)] and the successor search nodes of SN, pk must belong to one of the successor search nodes of SN.

Fig. 4 shows an example in which SN.I = [(2,8), (11,8)], pi = (8,8) and PSN = [(3,9), (10,9)]. Considering that all points painted gray or red are undoubtedly unblocked, the optimal path π (the blue line) must leave the RECT from a point in one of the successor search nodes. In Fig. 4, pj is (4,4), pk is (3,3) and pp is (9,9).

Assuming p is a point on the west wall of RECT, if pi belongs to the interval between the southeast and the southwest diagonal of p, $p.gval \leqslant$ p$i.gval$ + octile(p,pi) is guaranteed according to Section 3.3. This example is shown in Fig. 4 as p = (11,4).

Otherwise, assuming pd is the southeast diagonal intersection of p, $p.gval \leqslant$ pd.$gval$ + octile(pd,p) is guaranteed according to Section 3.3. Since pd.$gval \leqslant$ p$i.gval$ + octile(pi,pd) according to Definition 5 and octile(pi,p) equals octile(pi,pd) + octile(pd,p) in this condition, $p.gal \leqslant$ pd.$gval$ + octile(pd, p) $\leqslant$ p$i.gval$ + octile(pi,pd) + octile(pd,p) = p$i.gval$ + octile (pi,p) is still guaranteed. This is illustrated by p = (3,4) and pd = (7,8) in Fig. 4.

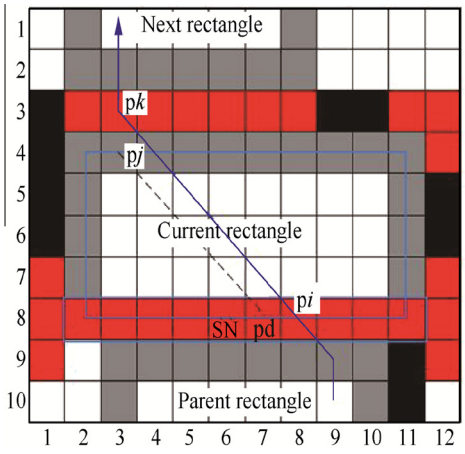


Fig. 4 A typical expansion process.

So, for each wall point p of RECT, $p.gval \leqslant$ p$i.gval$ + octile (p,pi) is always guaranteed.

p$j.gval \leqslant$ p$i.gval$ + octile(pi,pj) is then also guaranteed. Considering that pj is a downstream point of pi in π and pi is optimal, p$j.gval$ must equal p$i.gval$ + octile(pi,pj) because it is in fact the length of the optimal path from S to pj, thus pj will be optimal.

For any point p' in the same interior boundary with pj:

(1) $p'.gval \leqslant$ p$i.gval$ + octile(p',pi) $\leqslant$ p$i.gval$ + octile(pi, pj) + octile(pj, p') = p$j.gval$ + octile(pj, p') is guaranteed.
(2) $p'.gval <$ p$j.gval$ − octile(p', pj) is not possible because a path from S to pj passing p' will be shorter than pj.gval, which contradicts the optimality of pj.

So, the interior boundary that contains pj is optimal according to Definition 5.

Condition 1. The same process occurs when the interior boundary generates a successor search node that contains pk, as in Section 3.2. So, the new search node that contains pk is also optimal with its $minfval \leqslant$ p$k.fval \leqslant \pi.length$ according to Definition 4.

Condition 2. A new search node containing pk will not be generated because all points in the search node have not already had suboptimal parents and thus were not updated by SN. An equal optimal path π' from start point S to pk without passing pj must exist in this condition. Since Condition 2 will result in no children, pk will must have already been optimally assigned in π' by repeating Condition 1, and the process will not be interrupted by Condition 2. Assuming pk is assigned the optimal $gval$ when expanding an unblocked rectangle RECT', entrance of RECT' is certain by an optimal search node as a result of the last Condition 1. If pk is an outer boundary point of RECT', Condition 1 must have happened. Otherwise, π' must leave RECT' from another optimal outer

exit point p*m* passing p*k* with p*m*.*gval* > p*k*.*gval*. Since for any optimal point in π' the *gval* is finite (no greater than length of π'), the second assumption about p*k* and RECT' cannot loop infinitely. Therefore, Condition 2 will finally find the optimal path or end with Condition 1.

As a result, Lemma 1 is always guaranteed in both conditions.

Lemma 2. *If path* π_0 *is the path found by REA**, $\pi_0.length \leqslant CBN.minfval$ *is always guaranteed.*

Proof. Lemma 2 is satisfied in the third terminal condition of REA*. In the first two terminal conditions, the octile path between *G* and points in CBN is obviously traversable and a path with its length equal to CBN.*minfval* must exist. Since points in CBN may be updated before CBN is expanded, but *minfval* of the search node is never updated, $\pi_0.length \leqslant$ CBN.*minfval* is always guaranteed. □

Theorem. *REA** *will always return an optimal path (if there exists).*

Proof. If a path is found when inserting *S* into the map, it is obviously optimal and *G*.*gval* = octile(*S*,*G*). □

Otherwise, REA* will never end with an empty open list since an optimal search node is always waiting to be expanded in the open list according to Lemma 1. Assuming path π_0 is non-optimal and $\pi_0.length > \pi.length$, when π_0 is found, CBN.*minfval* $\geqslant \pi_0.length > \pi.length$ is guaranteed according to Lemma 2. Since a search node with its *minfval* greater than $\pi.length$ will never be expanded earlier than any optimal search node, π_0 will never be found earlier than π according to Lemma 1. So REA* will always end with an optimal path.

5. Efficiency

Compared with A*, the new method is unique and efficient because of the advantages brought by rectangle expansion.

Fig. 5 shows an example of A* and REA* on AR0020SR, a map from the benchmark problem sets used in the experiment, with *S* = (40,488) and *G* = (218,259).

5.1. Rectangle prune

REA* explores the map with unblocked rectangles and all interior points pruned. Many fewer points will be visited by REA* than A* and unnecessary multiplication and list operations will be omitted, so REA* can explore the map more efficiently.

5.2. Length of open list

Only the entrance interval of an unblocked rectangle will be considered as a search node and put into the open list, so there will be fewer search nodes in the REA* open list, meaning fewer and quicker list operations than A*.

5.3. Terminal conditions

REA* will terminate successfully whenever the goal point *G* is reachable from the current best search node by an octile path, which means unnecessary operations will be avoided in many cases. For example, if *G* is in the original rectangle expanded from start point *S*, REA* will reach the optimal path immediately. A* will always have to expand the entire optimal path.

5.4. Quality of resulting path

Though REA* returns grid-optimal path points, path points of REA* are either adjacent or skipped but protected by an unblocked rectangle. So, the straight lines between the resulting points form a path better than grid-optimal. For example, the path found by REA* in Fig. 1(d) has a shorter length (23.63) with less inflection than the grid-optimal path found by A* in Fig. 1(a) whose length is (24.07).

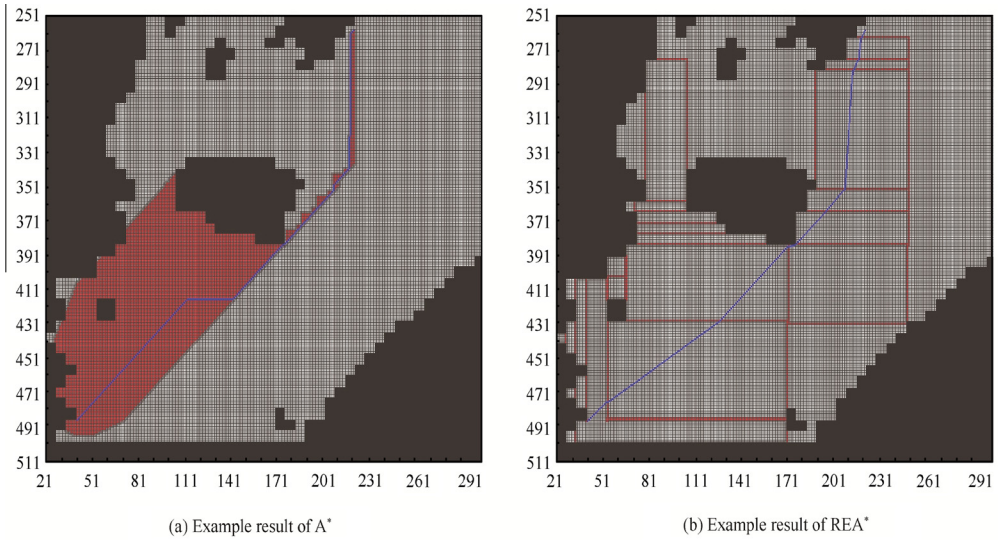


Fig. 5 Results of A* and REA* on a sample map.

6. Improvements and enhancement

Certain measures can be taken to further strengthen the basic REA*. They involve trade-offs that will make the algorithm more efficient in most conditions while preserving optimality.

6.1. Avoiding rescanning

An area may be scanned more than once by different search nodes. These rescans are usually unnecessary since latter search nodes will not update the scanned area in most situations. An improvement would be to assign wall points with an expansion direction. A new unblocked rectangle would then be stopped by not only blocked points, but also *gpoints* with the opposite expansion direction, which indicates the interior boundary of an old unblocked rectangle. Another enhancement measure is to shorten the length of search nodes by including only those points inside the updated parts of intervals in newly generated search nodes. For example, if CBN.*dir* = North, the new unblocked rectangle will be stopped by a *gpoint* whose expansion direction is South, Southeast or Southwest (corner points). If none of the points in south interior boundary of the old unblocked rectangle can be updated by CBN, the newly generated search node will avoid that rectangle and unnecessary rescanning.

6.2. Enlarge the original rectangle

Since boundary points of the original rectangle are always optimal, a large original rectangle will benefit REA*. A method to enlarge the original rectangle is to expand the start point S in both vertical and horizontal directions and prune the overlap to generate a cross rectangle. Another approach is to generate original rectangles for both S and G, and swap S and G if the original rectangle of G is larger than that of S.

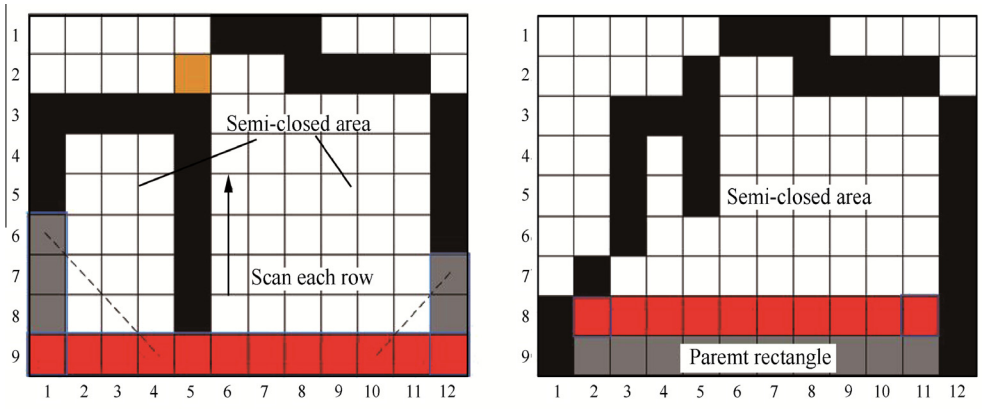
6.3. Identifying the semi-closed area

A semi-closed area can be identified when expanding a search node. For example, if CBN.*dir* = North, REA* will scan the map row by row from South to North. A column is blocked if there was a blocked point in this column during past scanning. When a row is being scanned, if a column is unblocked and one of its blocked neighbor columns has an unblocked cell in this row, the expanded area will not be narrowly semi-closed in the north. The scanning stops if all columns in the expanded area are blocked or the expanded area is identified to not be narrowly semi-closed. In Fig. 6(a), the current best search node is [(1,9), (12,9)] and the unblocked area will be semi-closed if point (5,2) is blocked. Otherwise the area will not be narrowly semi-closed since the unblocked column 6 has an unblocked neighbor point (5,2) when scanning the row 2. If all blocked points are not only north of the southeast diagonal of point (1,6), which is the northernmost point in the western boundary, but also north of the southwest diagonal of point (12,7), which is the northernmost point in the eastern boundary, calculations of *gval* for wall points will be unimpeded. Only the western boundary [(1,6), (1,9)] and eastern boundary [(12,7), (12,9)] will be used to generate new search nodes in Fig. 6(a) unless goal point G is within the expanded area. Fig. 6(b) shows another example in which no new search nodes will be generated.

7. Experimental results

Experimental verification is performed to prove the optimality of REA* and evaluate its effectiveness compared with standard A*. The A* in the experiments has been highly optimized; a matrix is used to store all map points and an ordered binary heap is used as the open list. These enhancements speed up list operations and decrease the disadvantages caused by the long open list in A*. Even so, REA* can still improve upon this highly optimized A* by an order of magnitude and more in the experiments.

Four typical benchmark problem sets with 917835 instances from 2015 Grid-Based Path Planning Competition[23,24] (GPPC, an international competition for grid based pathfinding) are selected, which are popularly used as standard benchmarks by many other researchers. All testing maps and instances (pairs of start points and goal points) are freely available from http://movingai.com. Fig. 7 shows sample maps from each of the four types.



(a) Example of new search nodes (b) Example without new search nodes

Fig. 6 Identifying a semi-closed area.

Dragon age origins (DAO) 156 maps and 159465 instances from the popular video game, with maps ranges from 22×28 to 1260×1104.

Baldur's gate II (BG2) 75 maps and 93160 instances from another popular role-playing game with all maps are scaled to 512×512. BG2 is representative because all maps have a distinctive 45° orientation.

Rooms 40 512×512 artificial maps and 78840 instances, with all maps divided into small rooms (8×8, 16×16, 32×32 or 64×64) connected by randomly placed entrances.

Mazes 60 artificial maze maps and 586370 instances, with all maps at 512×512 and having composed by corridor widths of 1, 2, 4, 8, 16 or 32.

All 917835 instances from GPPC are randomly generated while lengths of optimal paths in the same map are evenly distributed. The experiments are run on a Core i3 3.2 GHz PC with 2 GB of RAM, and the results are shown in Table 5.

For each map type, Table 5 shows the average length of the longest open list, the average total number of elements (open points for A* and search nodes for REA*) in the open list, the average length of the resulting path and the average time (by millisecond) to solve an instance for both A* and REA* in the experiments. Length of the path found by REA* is written as *gridlength*/*reallength*, and *reallength* is the sum of straight line distances between path points returned by REA*.

Table 5 shows that REA* can greatly speed up A* in all of the four types of maps and return the optimal path an order of magnitude faster than the highly optimized A*. The grid paths found by REA* always have the same length as the optimal path found by A* in all the 917835 instances, while the values of reallength are usually smaller than those of the grid-optimal paths.

MAZE1 is special in that REA* cannot shorten the path due to characteristics of this type of map. With the exception of MAZE1, REA* returns paths 97.64–99.65% of the grid-optimal length on average, and the total elements in open lists of REA* are 0.06–10.02% that of A*. The longest REA* open list is only 0.87–29.55% that of A*, which means fewer expansions and quicker list operations.

REA* performs best in the maps of ROOM64 and MAZE32, where REA*can increase A* speed by 23.22 and 26.27 times. This is because the existence of large open areas and regular shaped obstacles allows REA* to prune more unnecessary points. In maps of BG2 and DAO, a large proportion of obstacles with irregular edges and a distinctive 45° hypotenuse makes rectangular areas more piecemeal. This

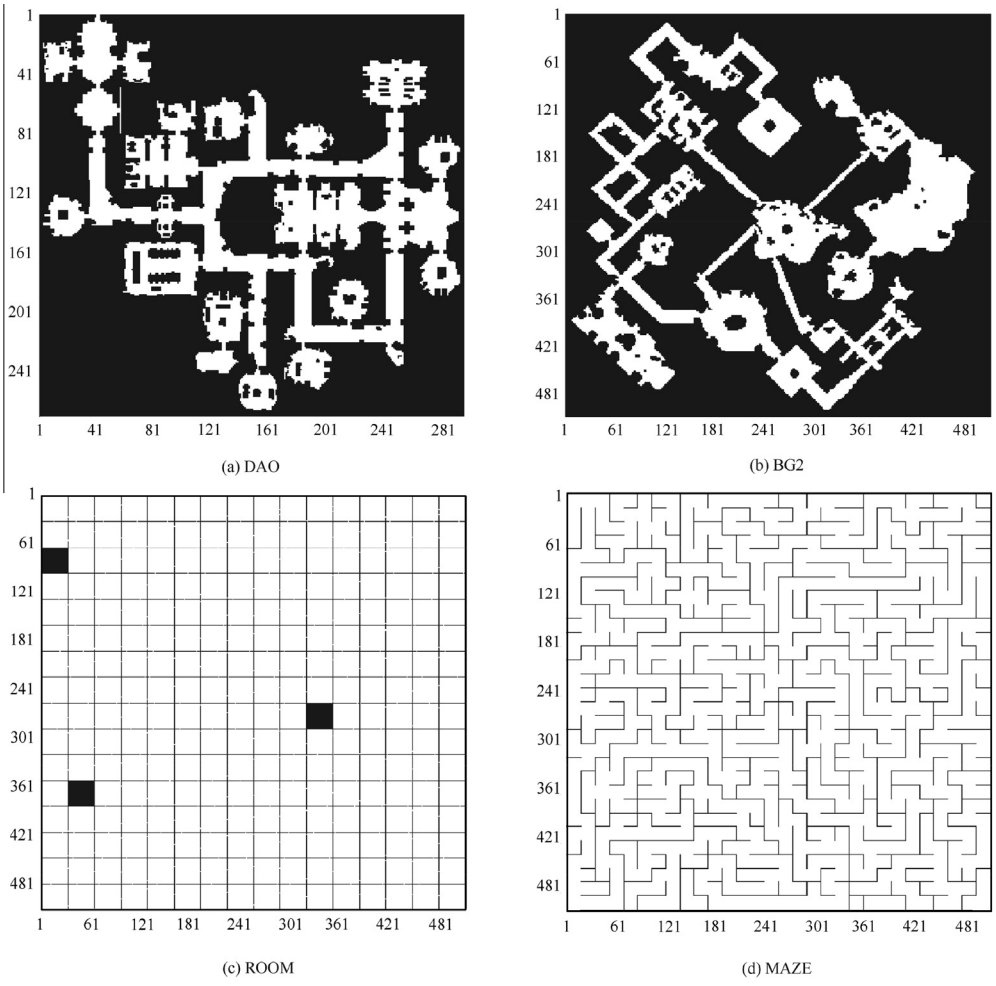


Fig. 7 Sample maps in the experiment.

Table 5 Experiment results on typical benchmark problem sets from GPPC.

Map type	Longest list		Total list element		Path length		Time (ms)	
	A*	REA*	A*	REA*	A*	REA*	A*	REA*
DAO	365	80	19662.2	953	418.57	418.57/413.00	14.43	3.25
BG2	488	71	16079.5	421	249.69	249.69/245.15	13.10	2.81
ROOM8	822	187	37953.5	3461	389.85	389.85/388.49	33.44	6.04
ROOM16	973	87	42649.9	986	385.66	385.66/381.94	37.19	3.01
ROOM32	982	40	56094.5	337	392.20	392.20/386.80	48.70	2.63
ROOM64	912	19	79885.6	131	426.37	426.37/419.00	68.50	2.95
MAZE1	74	76	64132.1	20202	2986.14	2986.14/2986.14	37.61	18.40
MAZE2	176	52	83425.0	8361	2193.1	2193.10/2156.53	54.86	10.95
MAZE4	296	32	104524.0	3104	1953.39	1953.39/1914.19	71.58	7.43
MAZE8	398	18	126648.0	1015	1927.86	1927.86/1879.90	87.05	5.33
MAZE16	514	10	145337.0	316	1730.85	1730.85/1690.04	102.24	4.29
MAZE32	688	6	149669.0	85	1246.84	1246.84/1218.26	110.28	4.20

decreases the efficiency of REA*, but it can still speed up A* by a factor of 4.67 in BG2 and 4.45 in DAO.

Table 6 compares REA* with the related algorithms on the three main evaluation metrics. In order to eliminate the effects of hardware differences, the speeds up factor over A*, instead of the absolute run time, is chosen to measure the search speed. Path length is the ratio of results for REA* to those of A*. 1 equates to grid-optimal, >1 is non-optimal and <1 means better than grid-optimal. The data for other algorithms is taken from the literature of their original authors. Data for RSR is from Refs.[16,17], data for Block A* is from Ref.[19] and data for JPS is from Refs.[20,21] They use the same benchmark problem sets from the GPPC except that not all map sets are used in all the original literature. Theta A*, lazy theta A*, A* PS and anya A* are all slower than A* and not shown in Table 6.

Though all three algorithms prune interior points of rectangles, REA* is more efficient than RSR and Block A* in all the three main evaluation metrics, even though the latter two algorithms pre-process the map. The reason is that RSR and Block A* deal with individual boundary points while REA* adopts linear search nodes to shorten the open list, which means fewer and quicker list operations. In addition, rectangles in RSR and REA* can be larger than the sedentary block in Block A* in many instances, which means higher efficiency.

Both JPS and REA* prune unnecessary points online. While REA* deals with entire unblocked rectangles, JPS considers only the jump points (necessary inflection points) on a map, which is more efficient. But, REA* can avoid rescanning of points and identify semi-closed areas while JPS cannot. So, the speed advantage of JPS is not absolute. Another advantage is that rectangle expansion makes it possible for REA* to return a path better than grid-optimal that is protected by unblocked rectangles, which JPS cannot do. In addition, REA* keeps more information about maps than JPS, which can prove useful in many instances, such when as allowing a

Table 6 Comparison with other variants of A*.

Algorithm	Speed-up factor		Path length	Pre-processing
REA*	DAO	4.45	<1	NO
	BG2	4.67		
	ROOMS	5.54–23.22		
	MAZES	2.04–26.27		
RSR	BG2	2.0–3.0	1	YES
	ROOMS	5.0–9.0		
Block A*	DAO	2.1	>1	YES
	BG2	2.3		
JPS	DAO	3.0–26.0	1	NO
	BG2	2.0–30.0		
	ROOMS	3.0–16.0		
SUB	DAO	14.1	1	YES
	BG2	35.0		
	ROOMS	22.5–562.3		
	MAZES	3.3–675.5		
JPS+	DAO	20.8–180.4	1	YES
CPDs	BG2	23.0–700.0	1	YES

moving target to reuse map information to avoid the unnecessary parts of a new search. Such cases are beyond the scope of this paper and will be discussed in the future.

Table 6 also shows the results of the highly successful offline algorithms SUB[8], JPS+[25] and CPDs[4–7], using data from their original literature. The grid-optimal version of SUB is selected from Ref.[8] and the data on JPS+ is converted based on the comparison of results with JPS in Ref.[25] By preprocessing the maps and sacrificing flexibility in a dynamic environment, offline algorithms can gain huge advantage in search speed. Generally, the tradeoff for search speed, quality of resulting path and the cost of pre-processing, is an important issue for researchers as it greatly affects the performance of algorithms. An algorithm that dominates all others in each of the three key metrics does not currently exist.

To classify all instances into different buckets according to the length of the optimal paths, Fig. 8 shows the detailed results in maps of MAZE, and similar results in other types of maps. The speed-up factor of REA* over A* increases with length of the optimal path in all maps. This is because, to prune unnecessary points, REA* requires additional operations to scan and identify maps, and while lengths of the optimal paths increase, the benefits brought by the additional operations (a shorter open list) also increase significantly. The upper limit of the REA* speed increase over A* is decided by the proportion of how many unnecessary points can be pruned, which depends on the characteristics of the map. For example, REA* performs better in MAZE32 than in MAZE1 because of the former's larger open areas.

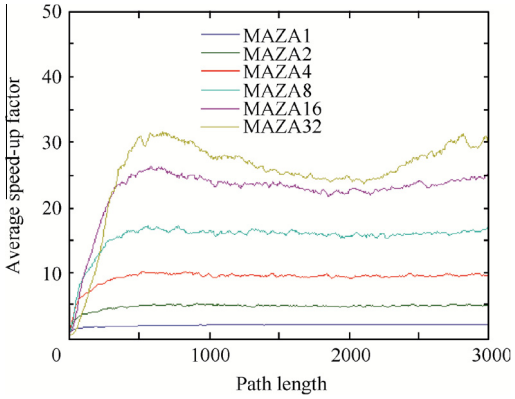


Fig. 8 Average speed-up of REA* with different path length in MAZE.

8. Conclusions

REA* is a fast, efficient, optimality preserving online algorithm that can always return grid-optimal path points with a real path length shorter than grid-optimal.

REA* can speed up a highly optimized A* by an order of magnitude and more in typical benchmark problem sets. The average speed-up factor increases with the optimal path length. The upper limit of the REA* speed increase over A* is decided by map characteristics. While REA* performs better in maps with large open areas and regular shaped obstacles, experiments show it can still speed up A* more than 4 times in maps with a high proportion of obstacles with irregular edges and a 45° hypotenuse. A comparison of results with those of related algorithms shows that REA* performs well in all three main evaluation metrics: search speed, quality of resulting path and cost of pre-processing. This new algorithm is competitive with other highly successful variants of A* and is not outmoded even by state-of-the-art JPS.

Future work should include the development of more efficient point pruning strategies and measures to enhance the performance of REA* on irregular maps. Another focus should be to extend the use of this method to more complex situations like moving targets.

Acknowledgement

This study was supported by the National Natural Science Foundation of China (No. 61573283).

References

1. Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transact Syst Sci Cyber* 1968;**4**(2):100–7.
2. Botea A, Müller M, Schaeffer J. Near optimal hierarchical pathfinding. *J Game Develop* 2004;**1**(7):7–28.
3. Pochter N, Zohar A, Rosenschein JS, Felner A. Search space reduction using swamp hierarchies. *Proceedings of the twenty-fourth AAAI conference on artificial intelligence*; 2010 July 11–15; Atlanta, USA. Menlo Park: AAAI Press; 2010. p. 155–6.
4. Botea A. Ultra-fast optimal pathfinding without runtime search. *Proceedings of the seventh AAAI conference on artificial intelligence and interactive digital entertainment*; 2011 October 11–14; Palo Alto, USA. Menlo Park: AAAI Press; 2011. p. 122–7.
5. Botea A. Fast, optimal pathfinding with compressed path databases. *Proceedings of the fifth annual symposium on combinatorial search*; 2012 July 19–21; Ontario, Canada. Menlo Park: AAAI Press; 2012. p. 204–5.
6. Botea A, Harabor D. Path planning with compressed all-pairs shortest paths data. *Proceedings of the twenty-third international conference on automated planning and scheduling*; 2013 June 10–14; Rome, Italy. Menlo Park: AAAI Press; 2013. p. 293–7.
7. Strasser B, Harabor D, Botea A. Fast first-move queries through run-length encoding. *Proceedings of the seventh annual symposium on combinatorial search*; 2014 August 15–17; Prague, Czech Republic. Menlo Park: AAAI Press; 2014. p. 157–65.
8. Uras T, Koenig S, Hernández C. Subgoal graphs for optimal pathfinding in eight-neighbor grids. *Proceedings of the twenty-third international conference on automated planning and scheduling*; 2013 June 10–14; Rome, Italy. Menlo Park: AAAI Press; 2013. p. 224–32.
9. Bast H, Funke S, Matijevic D. *TRANSIT: Ultrafast shortest-path queries with linear-time preprocessing*. New York: American Mathematical Society; 2009. p. 175–92.
10. Bast H, Funke S, Matijevic D, Sanders P. In transit to constant time shortest-path queries in road networks. *Proceedings of the ninth workshop on algorithm engineering and experiments*; 2007 January 6–9; New Orleans, USA. Philadelphia: Society for Industrial and Applied Mathematics Philadelphia; 2007. p. 46–59.
11. Antsfeld L, Harabor D, Kilby P, Walsh T. TRANSIT routing on video game maps. *Proceedings of the eighth AAAI conference on artificial intelligence and interactive digital entertainment*; 2012 October 8–12; Stanford, USA. Menlo Park: AAAI Press; 2012. p. 2–7.
12. Nash A, Koenig S. Any-angle path planning. *AI Magazine* 2013;**34**(4):85–107.

13. Daniel K, Nash A, Koenig S, Felner A. Theta*: any-angle path planning on grids. *J Artif Intel Res* 2010;**39**(1):533–79.
14. Nash A, Koenig S, Tovey C. Lazy theta*: any-angle path planning and path length analysis in 3D. *Proceedings of the third annual symposium on combinatorial search*; 2010 July 8–10; Atlanta, USA. Menlo Park: AAAI Press; 2010. p. 153–4.
15. Thorpe C, Matthies L. Path relaxation: path planning for a mobile robot. *Proceedings of the fourth national conference on artificial intelligence*; 1984 August 6–10; Austin, USA. Menlo Park: AAAI Press; 1984. p. 318–21.
16. Harabor D, Botea A, Kilby P. Path symmetries in undirected uniform-cost grids. *Proceedings of the ninth symposium on abstraction, reformulation, and approximation*; 2011 July 17–18; Catalonia, Spain. Menlo Park: AAAI Press; 2011. p. 58–61.
17. Harabor D, Botea A. Breaking path symmetries on 4-connected grid maps. *Proceedings of the sixth artificial intelligence and interactive digital entertainment conference*; 2010 October 11–13; Stanford, USA. Menlo Park: AAAI Press; 2010. p. 33–8.
18. Harabor D. Graph pruning and symmetry breaking on grid maps. *Proceedings of the twenty-second international joint conference on artificial intelligence*; 2011 July 16–22; Barcelona, Spain. Menlo Park: AAAI Press; 2011. p. 2816–7.
19. Yap P, Burch N, Holte R, Schaeffer J. Block A*: Database-driven search with applications in any-angle path-planning. *Proceedings of the twenty-fifth AAAI conference on artificial intelligence*; 2011 August 7–11; San Francisco, USA. Menlo Park: AAAI Press; 2011. p. 120–5.
20. Harabor D, Grastien A. Online graph pruning for pathfinding on grid maps. *Proceedings of the twenty-fifth AAAI conference on artificial intelligence*; 2011 August 7–11; San Francisco, USA. Menlo Park: AAAI Press; 2011. p. 1114–9.
21. Harabor D. *Fast pathfinding via symmetry breaking*. [Internet]. [cited 2015 July 1]; Available from: http://aigamedev.com/open/tutorial/symmetry-in-pathfinding/.
22. Harabor D, Grastien A. An optimal any-angle path finding algorithm. *Proceedings of the twenty-third international conference on automated planning and scheduling*; 2013 June 10–14; Rome, Italy. Menlo Park:AAAI Press; 2013. p. 308–11.
23. Sturtevant NR. The grid-based path planning competition. *AI Magazine* 2014;**35**(3):66–9.
24. Sturtevant NR. Benchmarks for grid-based pathfinding. *IEEE Transact Comput Intel AI Games* 2012;**4**(2):144–8.
25. Harabor D, Grastien A. Improving jump point search. *Proceedings of the twenty-fourth international conference on automated planning and scheduling*; 2014 June 21–26; Portsmouth, USA. Menlo Park: AAAI Press; 2014. p. 128–35.

Zhang An received his Ph.D. degree from Northwestern Polytechnical University. He is now a professor and doctoral advisor at Northwestern Polytechnical University. His main research interests are complex system modeling and effectiveness evaluation, and intelligent command and control engineering.

Li Chong is a Ph.D. student at Northwestern Polytechnical University. His research interests are autonomous decision-making control and collaborative planning of UAVs.

Bi Wenhao is a Ph.D. student at Northwestern Polytechnical University. His research interests are aircraft weapons fire-control technology and advanced aviation electronics integration technology platforms.