An Improved A* Algorithm and Implementation for Pathfinding in Symmetrical Circumstances

Gaifang Dong, Xueliang Fu, Heru Xue

*Corresponding author
College of Computer and Information Engineering
Inner Mongolia Agricultural University, Hohhot, China
donggf@imau.edu.cn, fuxl@imau.edu.cn, xuehr@imau.edu.cn

Abstract

The A* pathfinding algorithm is considered one of the best graph searching algorithms in the world and thus has been widely used in many applications, particularly computer games. However, the implementation of the A* algorithm has been based on the general procedure that searches through the whole search space to get the best path regardless of the circumstances of the search space. In case of symmetrical circumstances where the obstacle is regular and the starting point and the target are symmetrical in relation to the obstacle, the searching still needs to be carried out in the redundant space as well. By analyzing the A* algorithm, in this paper we propose a modified process and implementation of A* algorithm, which, in two-dimensional graph search, can save about half of the searching time in the symmetrical circumstances.

Keywords: Graph search, A* Algorithm, Symmetrical Circumstances, Implementation

1. Introduction

The A* algorithm is widely used in the pathfinding or graph traversal process by plotting an efficiently traversable path between the starting point (initial node) and target point (target node). The A* algorithm was first described in 1968 by Hart et al [1] as a result of an extension of the Dijkstra algorithm proposed in 1959 [2].

The A* algorithm uses a distance-plus-cost heuristic function to determine the order in which the search visits nodes during searching. As A* traverses a graph, it follows a path of the lowest known heuristic cost, keeping a sorted priority queue of alternate path segments along the way. In the end, the least-cost path from the initial node to the target node out of the sorted priority queue is chosen as the best path (or one of the best paths) for the searching problem [3].

Since the publication of the A* algorithm in 1968, efforts on improving the performance and expanding applications of the A* algorithm have been made in various publications [3-12]. However, most implementations of the A* algorithm were based on the general procedure that searches through the whole search space regardless of the circumstances of the search space. In case of symmetrical circumstances, the searching still goes through the redundant space. In this paper, we propose a modified process and implementation of A* in two-dimensional graph search, which can save about half of the searching time in the symmetrical circumstances.

In the following sections, we introduce the general procedure of A* in two-dimensional graph search in symmetrical circumstances, from which the weakness of current implementations of the A* algorithm can be exposed. We then present a modified procedure that can better deal with pathfinding using the A* algorithm in the symmetrical circumstances. The performance of the implementation based on this modified procedure is analyzed at last.

2. The A* Algorithm

2.1 Components of the A* algorithm

The A* algorithm represents a 2D graph using a two-dimensional grid. The algorithm uses the following three data structures and an evaluation function to facilitate the searching process:
OPEN_LIST: this open list being used to store vertices that will be assessed during searching;
CLOSE_LIST: this closed list being used to store the vertices that have been assessed during searching;
FATHER (n): representing the parent vertex of vertex n, or the vertex from which the search immediately moved to vertex n.
Cost function \( F(n) = G(n) + H(n) \): \( G(n) \) representing the mobile cost from the start vertex to a given vertex n along the favorable search path; \( H(n) \) estimating the cost of moving from vertex n to the target vertex.

To show how the cost function works, we assume the current point being at vertex A (red) and the target point being at vertex B (blue) on the grid (Figure 1), and let the cost of every horizontal or vertical movement be 10 and that of diagonal movement be 14. For the neighbor in the upper-left corner of A to move to the target B, it requires 8 horizontal moves and 1 vertical move to get there. This means the H value of this neighbor vertex of A is 90. This neighbor is also situated diagonally to the NW of A so the G cost of this neighbor is 14. Combining both the G and H values together, the cost F of this neighbor of A is 104. The common notation for G, H and F values of a given vertex is shown in Figure 1: G value shown at the bottom left corner, H value shown at the bottom right corner, and F value shown at up left corner.

2.2 Operational processes of the A* algorithm

Assuming the start vertex being A and the target vertex being B, initially A as the first vertex is put into OPEN_LIST for assessment. Since A is the only vertex in OPEN_LIST, A is removed from OPEN_LIST and then put into CLOSE_LIST.

Next, put all the neighbor vertices of A into OPEN_LIST for assessment and set A as FATHER of these neighbor vertices.

Calculate F values for all the vertices in OPEN_LIST, and transfer the vertex (for example N₁) with the smallest F value from OPEN_LIST into CLOSE_LIST; remove N₁ from OPEN_LIST.

Put those neighbor vertices of N₁ that are not in OPEN_LIST into OPEN_LIST for assessment and set N₁ as FATHER of these new neighbor vertices.

Calculate F values of these new vertices in OPEN_LIST, and transfer the vertex (for example N₂) with the smallest F value from OPEN_LIST into CLOSE_LIST; remove other N₂ from OPEN_LIST.

The process is repeated until the target vertex B is added to the OPEN_LIST as one of the neighbor vertices of the latest vertex in CLOSE_LIST; the path in CLOSE_LIST is the solution for this search.
2.3 Operational processes of the A* algorithm in symmetrical circumstances

In symmetrical circumstances where both the start vertex A and target vertex B are of symmetry in relation to the obstacle (Figure 2), according to the A* algorithm, the eight neighbor vertices of A, noted as N₁–N₈, are put into OPEN_LIST for assessment, i.e. OPEN_LIST = {N₁, N₂, N₃, N₄, N₅, N₆, N₇, N₈}.

![Figure 2](image)

Figure 2. The neighbor vertices of the start point and location of the target vertex

Calculate G, H and F values for all the vertices in OPEN_LIST. The calculated values for each of the eight vertices are given in Table 1 and shown in Figure 3.

<table>
<thead>
<tr>
<th>vertex</th>
<th>N₁</th>
<th>N₂</th>
<th>N₃</th>
<th>N₄</th>
<th>N₅</th>
<th>N₆</th>
<th>N₇</th>
<th>N₈</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>14</td>
<td>10</td>
<td>14</td>
<td>10</td>
<td>14</td>
<td>10</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>H</td>
<td>180</td>
<td>170</td>
<td>160</td>
<td>170</td>
<td>150</td>
<td>180</td>
<td>170</td>
<td>160</td>
</tr>
<tr>
<td>F</td>
<td>194</td>
<td>180</td>
<td>174</td>
<td>180</td>
<td>194</td>
<td>180</td>
<td>174</td>
<td>174</td>
</tr>
</tbody>
</table>

Figure 3. Initial status of vertices in OPEN_LIST
Among the eight vertices, N₅ has the lowest F value so N₅ is transferred from OPEN_LIST to CLOSE_LIST.

Set FATHER = N₅ and add its neighbor vertices that are not an obstacle and not in both OPEN_LIST and CLOSE_LIST into OPEN_LIST. Vertices in OPEN_LIST should now include N₁, N₂, N₃, N₄, N₆, N₇, N₉, N₁₀, and N₁₁ as shown in Figure 4a.

Since G, H and F values of every vertex are calculated from the same start node, there is no need to recalculate the values for N₁ – N₄ and N₆ – N₈. The values for N₉ – N₁₁ are given in Table 2.

With the lowest F value, N₁₀ is transferred from OPEN_LIST to CLOSE_LIST. Similarly N₁₀ is set to FATHER and its qualified neighbor vertices, N₁₂ – N₁₄, are added into OPEN_LIST (Figure 4b).

Figure 4. Progression of the searching process
Repeat this process until reaching $N_{22}$ that has no qualified neighbor vertex to be added into OPEN_LIST (Figure 5). The values for all the vertices assessed during the search so far are given collectively in Tables 1-3.

**Table 2.** The G, H and F values of vertices in OPEN_LIST in progression

<table>
<thead>
<tr>
<th>vertex</th>
<th>$N_8$</th>
<th>$N_{10}$</th>
<th>$N_{11}$</th>
<th>$N_{12}$</th>
<th>$N_{13}$</th>
<th>$N_{14}$</th>
<th>$N_{15}$</th>
<th>$N_{16}$</th>
<th>$N_{17}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>24</td>
<td>20</td>
<td>24</td>
<td>34</td>
<td>30</td>
<td>34</td>
<td>44</td>
<td>40</td>
<td>44</td>
</tr>
<tr>
<td>H</td>
<td>150</td>
<td>140</td>
<td>150</td>
<td>140</td>
<td>130</td>
<td>140</td>
<td>130</td>
<td>120</td>
<td>130</td>
</tr>
<tr>
<td>F</td>
<td>174</td>
<td>160</td>
<td>174</td>
<td>174</td>
<td>160</td>
<td>174</td>
<td>174</td>
<td>160</td>
<td>174</td>
</tr>
</tbody>
</table>

Figure 5. Vertices in OPEN_LIST after reaching the obstacle

**Table 3.** The G, H and F values of vertices in OPEN_LIST approaching the obstacle

<table>
<thead>
<tr>
<th>Vertex</th>
<th>$N_{18}$</th>
<th>$N_{19}$</th>
<th>$N_{20}$</th>
<th>$N_{21}$</th>
<th>$N_{22}$</th>
<th>$N_{23}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>54</td>
<td>50</td>
<td>54</td>
<td>64</td>
<td>60</td>
<td>64</td>
</tr>
<tr>
<td>H</td>
<td>120</td>
<td>110</td>
<td>120</td>
<td>110</td>
<td>100</td>
<td>110</td>
</tr>
<tr>
<td>F</td>
<td>174</td>
<td>160</td>
<td>174</td>
<td>174</td>
<td>160</td>
<td>174</td>
</tr>
</tbody>
</table>

Remember that vertices $N_5, N_{10}, N_{13}, N_{16}, N_{19}$, and $N_{22}$ are in CLOSE_LIST so the smallest F value currently in OPEN_LIST is 174 shared by 12 vertices that form 6 pairs of symmetrical vertices in relation to $A$, i.e., $N_5–N_{18}, N_3–N_{11}, N_{12}–N_{14}, N_{15}–N_{17}, N_{18}–N_{20}$ and $N_{21}–N_{23}$. Due to the symmetry, either vertex from the pair should have the same cost of reaching the target. However, in the traditional A* algorithm, even dealing with symmetrical circumstances, searching must be carried out over the whole searching space, some examples being shown in Figure 6. Therefore, we could retain only one set of the paired vertices in OPEN_LIST so as to improve the performance of the A* algorithm in symmetrical circumstances.
3. Improving Performance of the A* Algorithm in Symmetrical Circumstances

Assuming in a symmetrical case where A is still the start vertex (Figure 2), let \( N_i \) and \( N_j \) be the two neighbors of A, not being part of the obstacle and not being added to OPEN_LIST and CLOSE_LIST yet. If \( G(N_i) = G(N_j) \) and \( H(N_i) = H(N_j) \), then \( N_i \) and \( N_j \) are the paired vertices and should have the same cost in reaching the target. To improve the performance of A* for searching through symmetrical spaces, a modified procedure is proposed as follows.

Figure 6. Examples of symmetrical searching by the A* algorithm
1. Put A into OPEN_LIST.
2. Repeat the following steps:
   2.1 Set the vertex with the lowest F value in OPEN_LIST to CurrentPoint.
   2.2 Remove CurrentPoint from OPEN_LIST and add CurrentPoint to CLOSE_LIST.
   2.3 Add all qualified neighbors of CurrentPoint into OPEN_LIST and number each NEIGH.
   2.4 Calculate G, H, and F values for all vertices in OPEN_LIST.
   2.5 Assess cost values of the neighbors; for all distinctive vertices and paired neighbors with the smallest cost value in OPEN_LIST:
      2.5.1 If NEIGH is an obstacle or in CLOSE_LIST, do nothing;
      2.5.2 If NEIGH is not in the OPEN_LIST and not part of obstacle, put NEIGH into OPEN_LIST;
      set CurrentPoint as NEIGH’s FATHER;
      calculate G(NEIGH) and H(NEIGH);
      2.5.3 If NEIGH is in OPEN_LIST,
      if G(CurrentPoint) + Distance(CurrentPoint, NEIGH) < G(NEIGH),
      set CurrentPoint as NEIGH’s FATHER;
      calculate G(NEIGH) and H(NEIGH);
   2.6 If B is in OPEN_LIST or OPEN_LIST is empty, End looping.
3. Obtain the solution by tracking backwards from target vertex B along FATHER.

We implemented the improved A* algorithm using C++ and applied this program to some 2D graphs. Results of two cases of graph searching are shown in Figure 7. For any of the symmetrical cases, the search space is halved and thus searching time is shortened by about a half compared to that of the traditional A* algorithm.

![Figure 7](image)

**Figure 7.** Examples of symmetrical searching by the improved A* algorithm
4. Results and Conclusion

We have run the programs of both traditional and improved A* algorithms for two symmetrical cases on the same platform and the results are given in Table 4. It is evident that the improved A* algorithm is capable of reducing the searching time by about the half compared with the traditional A* algorithm. However, this is only useful for searching symmetrical spaces. In practice, this improved algorithm should be embedded into the A* algorithm to deal with the special circumstances where the search space or part of the search space is symmetrical.

Table 4. Comparison of performances of the traditional and improved A* algorithm

<table>
<thead>
<tr>
<th>Instance</th>
<th>Starting point</th>
<th>End point</th>
<th>Obstacle</th>
<th>Traditional A*</th>
<th>Improved A*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Node searched</td>
<td>Time (ms)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Node searched</td>
<td>Time (ms)</td>
</tr>
</tbody>
</table>

6. References