Annales Mathematicae Silesianae **22** (2008), 69–82
Prace Naukowe Uniwersytetu Śląskiego nr 2703, Katowice

APPLICATION OF THE A* ALGORITHM TO PROBLEMS OF THE EUCLIDEAN SHORTEST PATHS IN THE PLANE WITH POLYGONAL OBSTACLES

Anna Wojak

Abstract. The Euclidean shortest path between two points s and t in the plane with the cellular decomposition in the presence of obstacles is considered. The A* algorithm for a visibility graph (VG) is used to avoid *widened obstacles*. Computational experiments show that the proposed algorithm is often faster and it analyzes fewer nodes than the classical Dijkstra algorithm.

1. Introduction

The computation of the shortest path between two nodes in a weighted graph has been studied in the graph theory and network optimization for over forty years. A lot of algorithms and their modifications have been proposed, e.g. D'Esopo–Pepe, Dial, Floyd–Warshall, Danzig, Bellman–Fordand and Dijkstra [1, 3, 9]. The comparison and analysis of their computational complexity were presented by Pallottino [10], who performed tests on various data structures. The optimal-time algorithm for the classical shortest path problem runs in the worst-case time $O(n \log n)$ and it requires $O(n \log n)$ space, where n is the total number of nodes in the graph.

Several years ago scientists started to work on heuristic algorithms for the determination of the shortest path. One of them is the A* algorithm (A STAR). Its main idea was proposed by Pohl [11] and Hart [4]. Hart

Received: 23.05.2005. Revised: 9.04.2008.
(2000) Mathematics Subject Classification: 68W05, 68U05.
Key words and phrases: shortest path, obstacles, visibility graph.

used the A* algorithm to solve the shortest path problem in static networks. His results were extended by Chabini and Lan [2] for the computation of the fastest paths in a dynamic network.

The Euclidean shortest path problem is an important problem in the computational geometry and it has a lot of applications in robotics, transportation and computer graphics. The best-known application is the geographic information system (GIS), which helps to locate an object, to plan a route or to avoid traffic.

The shortest path problem considered in a geometric domain, in contrast to graphs, is usually specified by geometric objects. Given a set of polygonal obstacles in the plane, the problem is to compute the shortest path between two points avoiding all the obstacles. There are two fundamental methods: Shortest Paths Map (SPM) and Visibility Graph (VG). SPM tries to solve a more general problem: build a map of the shortest paths from a given origin point to all points in the plane. Hershberger and Suri [5] presented an algorithm with the computational complexity $O(n \log n)$, where n is the total number of obstacle vertices. They proposed a special decomposition of the space called "the conforming subdivision". Their algorithm studies the propagation of the wave in the cells of this decomposition in order to avoid the obstacles. The wavefront moves among the edges of obstacles and does not enter the obstacles. SPM can be used to the answer single-shortest path queries in $O(\log n)$ time.

The method of Visibility Graph method uses a graph whose nodes are the vertices of the obstacles and whose edges are pairs of mutually visible vertices. O'Rourke [9] described the construction of the VG and computed the shortest path on this graph by running the Dijkstra algorithm. Kapoor, Maheshwari, Mitchell [7] and Mitchell [8] have shown that the Euclidean shortest path avoiding obstacles in the plane can be found in $O(n + h^2 \log n)$ time, where n is the total number of obstacle vertices and h is the number of obstacles in the plane. Ghosh and Mount [6] have shown that the optimal time algorithm is $O(n \log n + E)$, where E is the number of edges in the VG graph. In the worst case E is equal to $O(n^2)$, but they showed that E is usually much smaller than $\binom{n}{2}$.

In this paper we combine the VG method with the heuristic search in order to find the Euclidean shortest path between two points in the plane with the presence of obstacles. We show algorithms for the construction of obstacle vertices for a given *initial map* and for the determination of the Visibility Graph. We present the ideas underlying the A* algorithm and show how to use it to improve efficiency of the determination of the shortest path in the Visibility Graph.

The paper is organized as follows. In Section 2 we recall the fundamental information about the A* algorithm. We introduce the notation and describe the heuristic method for a simple graph [4, 11]. In Section 3 we propose an

algorithm for the determination of the obstacle vertices. We present a square cellular decomposition of the two-dimensional plane. For this decomposition we widen the original obstacles by marking the neighbouring cells. These marks allow us to determine the vertices of the obstacles. In Section 4 we present the implementation of the Euclidean A* algorithm in the geometric domain. In Section 5 we compare the classical Dijkstra algorithm with the Euclidean A* algorithm. Our numerical results show that the A* algorithm is usually faster and analyzes fewer vertices than the Dijkstra algorithm.

2. The basics of the A* algorithm

Consider a graph $G = (N, E)$, where N is the set of nodes and E is the set of edges. To each edge $(i, j) \in E$ in G is attached its cost $c_{ij} > 0$. Let $s \in N$ denote the origin node and $t \in N$ denote the destination node.

Similarly to the Dijkstra algorithm [9], the A* algorithm uses the node labelling method: labels are either *temporary* or *permanent*. A node i has a permanent label, denoted by $[d_i]$, if the shortest path from the origin s to i is known. The label stores the cost of this path and a node (predecessor) from which i was reached. A temporary label of the node i, denoted by (d'_i), contains the predecessor and the cost of a path given by formula:

$$d'_i = \begin{cases} \min\{[d_j] + c_{ji} + f_i : \text{ for all } \ j \in E(i)\}, & \text{if } j \neq s, \\ f_i, & \text{if } j = s, \end{cases} \tag{1}$$

where $E(i)$ is the set of all neighbours of the node i, which have permanent labels, and f_i is a heuristic value associated with the node j. This value tells how the node i is distant from the destination node t. It could be the shortest path, the minimal number of nodes or edges on the path or a metric distance between those two nodes. A temporary label stores a heuristic cost of the path from the origin s to the destination t running though the node i.

A* algorithm

Let S and $\overline{S}$ denote two sets consisting of nodes with permanent labels and all other nodes of the graph $G = (N, E)$, respectively. At the beginning of the A * algorithm (see Algorithm 1), the set $S = \emptyset$ and $\overline{S}$ is the set of all nodes of the graph G, i.e., $\overline{S} = N$. The value d_j is the minimal cost of the path from the node s to the node j, $pred(j)$ is the predecessor of the node j, c_{ij} is the cost of the edge $(i, j) \in E$ and f_j is a "distance" between the nodes

j and t. The value d'_j given by formula (1) is a heuristic cost of a path P_{st} via the node j.

Each iteration of Algorithm 1 we start by choosing the node i with a minimal heuristic value d'_j (line 3). This node is permanently labelled (line 4). In the first iteration it is origin node s. Next we establish all temporary labels for neighbouring nodes to i (lines 6–10). We iterate as long as we assign the node $i = t$ (line 3).

ALGORITHM 1. A* algorithm

```
1:  S := ∅; S̄ := V; d_s := 0; pred(s) := 0; d'_s = f_j;
2:  while |S| < V do
3:      let i ∈ S̄ be the node for which [d_i] := {d_j : min((d'_j) : j ∈ S̄)}
4:      S := S ∪ {i};
5:      S̄ := S̄ − {i};
6:      for all (i, j) ∈ E(i) do
7:          if d_j > [d_i] + c_ij then
8:              d_j := [d_i] + c_ij;
9:              (d'_j) := d_j + f_j;
10:             pred(j) := i;
```

It is easy to notice that the Algorithm 1 is similar to the Dijkstra algorithm presented in [1]. The Dijkstra algorithm chooses the node to be permanently labelled with respect to the minimal value d_j whereas the A* algorithm chooses the node, which has the minimal heuristic value d'_j from all temporarily labelled nodes. The upper bound of the computational complexity of the A* algorithm is $O(n^2)$, where n is the total number of nodes in the graph G. In Section 5 we show that for the same costs of paths the A* algorithm is faster and analyses fewer nodes than the classical Dijkstra algorithm.

3. The definition of map and the determination of vertices of polygonal obstacles

Consider a two-dimensional plane K with a square cellular decomposition. Each square cell is described by coordinates $\overline{x}_{i,j} = (x_i, y_j)$ of its center for $i, j = 1, ..., n$. This cell is called *the point* of the plane K. The set of all square cells we denote by $\overline{X}$, and $\overline{X} = \{\overline{x}_{i,j} : \overline{x}_{i,j} \in K\}$. Let $U(\overline{x}_{i,j}) = \{\overline{x}_{i+1,j}, \overline{x}_{i-1,j}, \overline{x}_{i,j+1}, \overline{x}_{i,j-1}\}$ be a four-neighbourhood of the point $\overline{x}_{i,j}$ (see Figure 1).

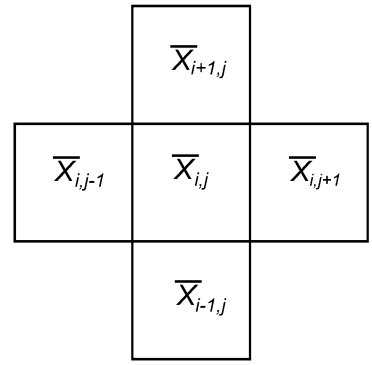

Figure 1. The surrounding of the square cell $\overline{x}_{i,j}$

Let O be any subset of $\overline{X}$, and let us define a set B as follows:

$$B = \{\overline{x}_{i,j} : \overline{x}_{i,j} \in \overline{X} \setminus O \text{ and } U(\overline{x}_{i,j}) \cap O \neq \emptyset\}. \tag{2}$$

Consider *a marking function* $C\colon \overline{X} \longrightarrow \{1, 0, -1\}$ and

$$C(\overline{x}_{i,j}) = \begin{cases} 1 & \overline{x}_{i,j} \in O, \\ 0 & \overline{x}_{i,j} \in B, \\ -1 & \overline{x}_{i,j} \in \overline{X} \setminus (O \cup B), \end{cases} \tag{3}$$

which is used to define the obstacle vertices.

Definition 1. The cell $\overline{x}_{i,j} \in B$, such that

$$C(\overline{x}_{i,j+1}) + C(\overline{x}_{i,j-1}) \neq C(\overline{x}_{i+1,j}) + C(\overline{x}_{i-1,j}) \tag{4}$$

is called an *obstacle vertex*.

The set O is called the set of *obstacle points*. It can be presented as a sum of finite disjoint polygonal obstacles $O = \bigcup_{i=1}^{n} O_i$, where O_i consists of obstacle points in a form of the polygonal, $O_i \cap O_j = \emptyset$ for $i \neq j$. The corresponding set B is called a set of *obstacle boundary points*. The set $\overline{X} \setminus (O \cup B)$ is called a set of *free points of the plane* K. The map with the square cellular decomposition with these three kinds of points is called *the initial map* **IM**.

3.1. The determination of obstacle vertices

Consider a map with the square cellular decomposition where every square has a side a. For this map we mark above mentioned three kinds of points according to formula (3). The maximal connected set of obstacle points and its boundary points is called *widened obstacle.* The edges of the widened obstacle consist of the set of obstacle boundary points and obstacle vertices. The boundary points should enclose obstacle points (see Figure 4d, obstacle III). If obstacle points are connected with sides of **IM**, then the boundary points enclose a part of obstacle points (see Figure 4d, obstacles I and II). The four - neighbourhood of boundary points with obstacle points reduces the number of vertices of the original obstacle, which consists of obstacle points. This reduction let us efficiently avoid the obstacles using edges and vertices of widened obstacles and it decreases the number of obstacle vertices. Now we define a new map called *a real map* (**RM**). This **RM** consists of the initial map **IM** and two arbitrarily chosen points: the origin s and the destination t. In addition we assume that the points s and t are not obstacle points $s, t \notin O$. According to the marking function the obstacle points, the boundary points and free space points have the values 1, 0 and -1, respectively. In Algorithm 2 we present the procedure of the vertex determination (VD).

Algorithm 2. VD procedure

```
1: for all x̄_{i,j} ∈ X̄ do
2:     if x̄_{i,j} ∈ B then
3:         if C(x̄_{i,j+1}) + C(x̄_{i,j-1}) ≠ C(x̄_{i+1,j}) + C(x̄_{i-1,j}) then
4:             x̄_{i,j} is a obstacle vertex;
```

We look through a **RM** starting in the left-top point of the map using the methodology of raster-curve and we check every point $\overline{x}_{i,j} \in \overline{X}$ according to the formula (2) if it is a boundary point. If $\overline{x}_{i,j} \in B$ and the formula (4) holds then $\overline{x}_{i,j}$ is the obstacle vertex according to Definition 1. The computed obstacle vertices are stored in a table (see e.g. Table 1), which is an input to our algorithm. The structure of this table is described in details in Section 4.

As an example, we consider the initial map with the square cellular decomposition for $a = 10$ (our plane is covered by 10^2 squares) with one obstacle and two points: the origin and the destination, respectively, in Figure 2. This obstacle is widened according to the four-neighbourhood and the boundary points have a lighter colour than the obstacle points. The numbers in squares denote the order in which the vertices were computed. Two points with number 0 and 10 denote: the origin and the destination, respectively.

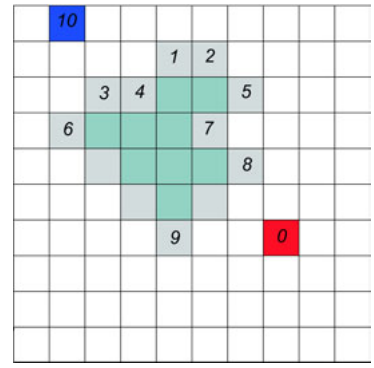

Figure 2. The determination of obstacle vertices

4. Implementation of the Euclidean A* algorithm

To find the Euclidean shortest path in the map between the nodes s and t avoiding obstacles we use the Visibility Graph method [6–9]. Nodes of VG are the obstacle vertices and edges are the pairs of their mutually visible vertices. To compute the set of VG edges we use a simple algorithm. We look at every edge to see if it blocks or interferes with a given pair of vertices. If not the two vertices are visible to each other. Of course, to construct the entire visibility graph, the procedure loops through every pair of vertices. There are $O(n^2)$ pairs of vertices and there are $O(n)$ edges for each vertex so the total time is $O(n^3)$. The algorithm requires $O(n)$ space [9]. To compute the Euclidean shortest path we combine the methodology of a VG construction with the A* algorithm (see Algorithm 1). For the shortest path problem in the geometric domain we call it the Euclidean A* algorithm. First we compute the obstacle vertices which become the VG nodes. For each node j the following information is considered: the node number, coordinates of the point on the plane K associated with the node j, the marker C, the cost d_j for the Euclidean shortest path P_{sj}, the predecessor of the node j, the heuristic value f_j and the value d'_j, respectively (as an example, see Table 1 for Figure 2). The marker C describes nodes with temporary label $C = 0$, nodes with the permanent labels $C = 1$ and the destination node t for $C = 10$. The Euclidean metric is used to determine the cost c_{ij} of the edge between two nodes i and j

$$c_{ij} = \sqrt{(i_x - j_x)^2 + (i_y - j_y)^2} \tag{5}$$

and the value f_j, which is the Euclidean distance between j and t

$$f_j = \sqrt{(j_x - t_x)^2 + (j_y - t_y)^2}, \tag{6}$$

where (i_x, i_y), (j_x, j_y), (t_x, t_y) are coordinates of points i, j and t, respectively.

The algorithm runs through two stages. The first one is a preprocessing phase. In this phase we compute the lists $A(j)$ of visible neighbours of each node $j \in N$. The second phase is the determination of the shortest path using the Euclidean A* algorithm (see Algorithm 3). Input of our algorithm is the table T.

ALGORITHM 3. Euclidean A* algorithm

```
 1: min: = 0; d_s := 0; d'_s := f_s;
 2: i = s
 3: while C(i) ≠ 10 do
 4:     for all j ∈ A(i) do
 5:         if C(j) ≠ 1 then
 6:             if d_j > min + c_ij then
 7:                 d_j := min + c_ij;
 8:                 d'_j := d_j + f_j
 9:                 pred(j) := i
10:     i :=MIN(d'_i) ;
11:     min := d_i;
```

Algorithm 3 consist of two parts: a path extention procedure (lines 4-9) and MIN($\cdot$) function (line 10). From the set of temporarily labelled nodes we choose the one which has the minimal heuristic value d'_i. Its label becomes permanent (line 10) and we start the path extention procedure. We extend the path P_{si} to the nodes which are visible from i (line 4). For each of these nodes we establish the label. For the nodes which has not been labelled we create new labels and for the labelled nodes we update costs if it is necessary (line 7). The values c_{ij} and f_j are computed according to the formulas (5) and (6), respectively. We iterate as long as $C(i) \neq 10$. The path extention procedure requires $O(n^2)$ time while the MIN function $O(n)$ [3].

5. Comparison of Euclidean A * and Dijkstra algorithms

In this section we compare the performance of the A* algorithm with the classical Dijkstra algorithm applied to a geometric problem. Our results show that the A* algorithm outperforms the classical Dijkstra algorithm.

First, we consider the map in Figure 2, which consists of one obstacle and two points: the origin and the destination. The vertices of the obstacle are numbered (1–9), the origin point is numerated by 0 and the destination by 10.

In the VD procedure (Algorithm 2) we compute the obstacle vertices, which are stored in the table T (see Table 1). This vertices together with the origin and the destination create the set of VG nodes. For each VG node we computed the visible nodes. Next we apply the Euclidean A* algorithm (see Algorithm 3) and the classical Dijkstra algorithm [1]. Final results of both algorithms are shown in the table T_A (see Table 2) and T_D (see Table 3), respectively.

Table 1. Table T contains the obstacle vertices for Figure 2

Vertex	x_j	y_j	C	d_j	$pred(j)$	f_i	d'_j
0	375	325	1	0	0	0	0
1	225	75	0	0	0	0	0
2	275	75	0	0	0	0	0
3	125	125	0	0	0	0	0
4	175	125	0	0	0	0	0
5	325	125	0	0	0	0	0
6	75	175	0	0	0	0	0
7	275	175	0	0	0	0	0
8	325	225	0	0	0	0	0
9	225	325	0	0	0	0	0
10	75	25	10	0	0	0	0

We recall that the columns in Table 1 denote: the node j, x and y coordinates of the obstacle vertex j, the marker C, the cost d_j of a path P_{sj}, the predecessor of the node j, the cost of an Euclidean shortest path f_j from the node j to the node t, and the heuristic value d'_j, respectively. In table T_D we remove columns associated with the heuristic values d'_j and f_j because they are not used in the Dijkstra algorithm.

Table 2. The table T_A for results of the Euclidean A* algorithm

Vertex	x_j	y_j	C	d_j	$pred(j)$	f_j	d'_j
0	375	325	1	0	0	0	0
1	225	75	0	326	2	158	484
2	275	75	1	276	5	206	482
3	125	125	0	434	2	111	545
4	175	125	0	0	0	0	0
5	325	125	1	206	0	269	475
6	75	175	0	0	0	0	0
7	275	175	1	181	8	250	431
8	325	225	1	111	0	320	431
9	225	325	0	150	0	335	485
10	75	25	10	482	2	0	482

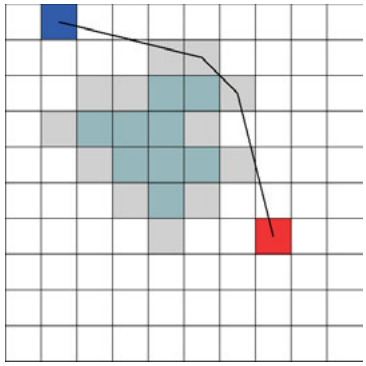

Figure 3. The shortest path computed by A* and Dijkstra algorithms for Figure 2

Analyzing the marker column C of the table T_A we notice that there are points for which the marker $C = 0$. It means that not all points were permanently labelled. On the other hand, in the table T_D all vertices have the marker $C = 1$, so we infer that the Euclidean A* algorithm analyzes fewer vertices than the Dijkstra algorithm and both of them give the same shortest path (see Figure 3). To prove it, let us consider more complicated examples of real maps with four types of decompositions for $a = \{10, 30, 50, 100\}$.

Table 3. The table T_D for results of the Dijkstra algorithm

Vertex	x_j	y_j	C	d_j	$pred(j)$
0	375	325	1	0	0
1	225	75	1	326	2
2	275	75	1	276	5
3	125	125	1	432	6
4	175	125	1	396	1
5	325	125	1	206	0
6	75	175	1	362	9
7	275	175	1	181	8
8	325	225	1	111	0
9	225	325	1	150	0
10	75	25	10	482	2

Our maps are approximated by a^2 squares. Examples are shown in Figure 4 and Figure 6. We compare the algorithms with respect to the number of obstacles and the square decomposition. The total number of nodes for the visibility graph $|N|$, the cost of the path $d(P_{st})$, the total number of analysed nodes θ and the algorithm running time τ were the objects of our analysis. The results for the example maps are presented in Table 4. Consider the map for $a = 30$ with three obstacles (Figure 4d). The results obtained are: the total number of vertices are 70, the Euclidean A* algorithm analyzed 55 vertices in time 0,09 seconds while the Dijkstra algorithm analyzed 67 vertices in time 0,12 seconds (9–10 rows in Table 4).

Table 4. Result of the Euclidean A* and Dijkstra algorithms for the a – square decomposition. The value $|N|$ denote the total number of nodes in the visibility graph, $d(P_{st})$ – the cost of the shortest path from the origin to the destination, θ – the total number of analyzed nodes, τ – the algorithm running time in seconds.

Algorithm	Number of obstacles	a	$d(P_{st})$	$\|N\|$	θ	τ	Map
A*	2	10	674	15	5	0,03	Figure 4a
Dijkstra	2	10	674	15	12	0,05	Figure 4a
A*	3	10	423	16	5	0,02	Figure 4b
Dijkstra	3	10	423	16	15	0,05	Figure 4b
A*	4	10	628	23	11	0,05	Figure 4c
Dijkstra	4	10	628	23	21	0,06	Figure 4c
A*	2	30	973	78	60	0,1	Figure 4e
Dijkstra	2	30	973	78	76	0,18	Figure 4e
A*	3	30	1162	70	55	0,09	Figure 4d
Dijkstra	3	30	1162	70	67	0,12	Figure 4d
A*	3	30	617	44	20	0,13	Figure 5a
Dijkstra	3	30	617	44	42	0,17	Figure 5a
A*	4	30	741	63	24	0,15	Figure 5b
Dijkstra	4	30	741	63	58	0,25	Figure 5b
A*	5	30	1066	91	81	0,47	Figure 5c
Dijkstra	5	30	1066	91	89	0,5	Figure 5c
A*	6	30	681	61	27	0,19	Figure 5d
Dijkstra	6	30	681	61	60	0,27	Figure 5d
A*	2	50	878	304	198	1,59	Figure 5e
Dijkstra	2	50	878	304	287	2,77	Figure 5e
A*	2	50	1318	70	60	0,08	Figure 4f
Dijkstra	2	50	1318	70	69	0,11	Figure 4f
A*	3	50	579	76	41	0,23	Figure 5f
Dijkstra	3	50	579	76	72	0,33	Figure 5f
A*	4	50	908	318	208	3,11	Figure 5g
Dijkstra	4	50	908	318	289	3,42	Figure 5g
A*	5	50	681	291	111	1,77	Figure 5h
Dijkstra	5	50	681	291	259	2,47	Figure 5h
A*	6	50	1075	537	271	6,07	Figure 5i
Dijkstra	6	50	1075	537	507	7,01	Figure 5i
A*	6	100	907	639	377	7,37	—
Dijkstra	6	100	907	639	699	9,01	—

A similar analysis may be performed for all entries in this table. The computational tests show that in problems of computing the shortest path with the presence of polygonal obstacles, the Euclidean A* algorithm is faster and analyzes fewer nodes than the classical Dijkstra algorithm.

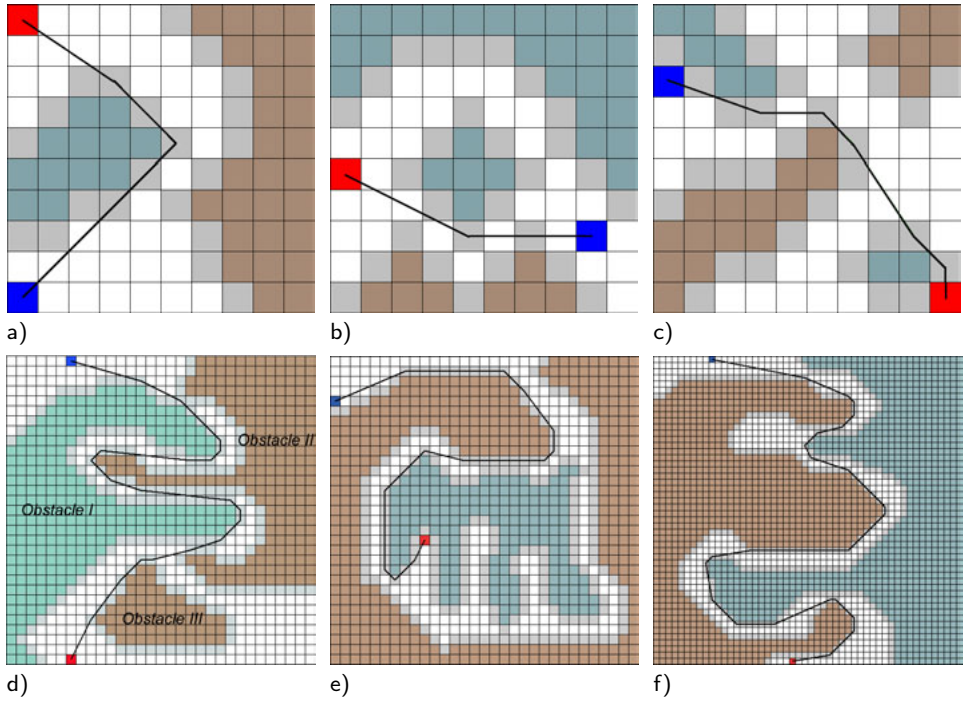

Figure 4. The shortest paths computed by Dijkstra and A* algorithms with tree types of the square decomposition: a–c) for $a = 10$; d–e) for $a = 30$; f) for $a = 50$, respectively

6. Conclusions

In the present paper the Euclidean shortest path problem on the plane with the presence of polygonal obstacles is considered [5, 7, 8]. We combine a heuristic search (the A* algorithm [4, 11]) with the Visibility Graph method [6, 9]. For a map with obstacles we propose the procedure to determine the Visibility Graph: we use a square decomposition of the plane and a marking function to find obstacle vertices, the nodes and the edges of the VG. To such prepared data we apply two algorithms: Euclidean A* and Dijkstra's. Numerical tests (the analysis refers to the computation time, the number of visited nodes and the cost of the paths) show that the Euclidean A* algorithm outperforms the classical Dijkstra algorithm: it analyzes fewer vertices and has a shorter running time.

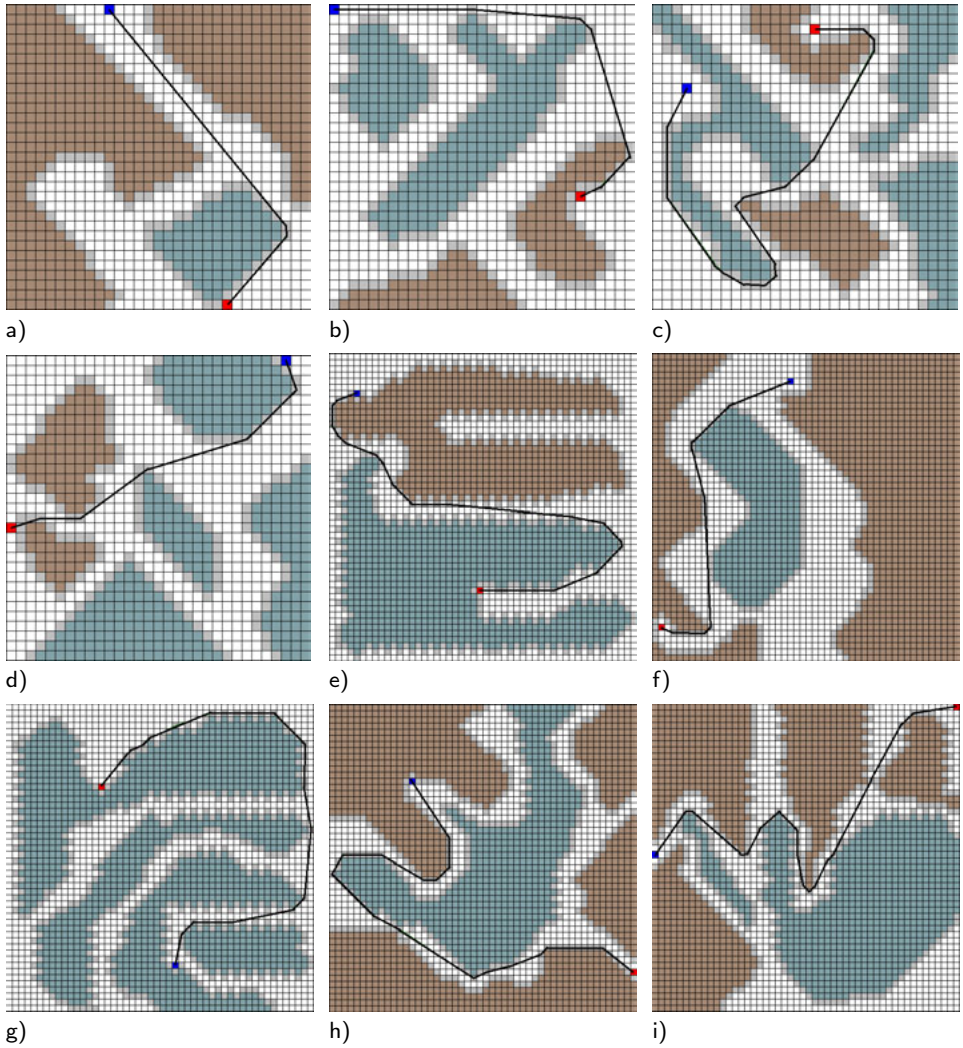

Figure 5. The shortest paths computed by Dijkstra and A* algorithms with tree types of the square decomposition: a–d) for $a = 30$; e–i) for $a = 50$, respectively

References

[1] Ahuja R.K., Magnanti T.M., Orlin J.B., *Network Flows – Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] Chabini I., Lan S., *Adaptation of the A* algorithm for computation of faster paths in deterministic discrete-time dynamic networks*, IEEE. Trans. on Intel. Trans. Syst. **3** (2002), 60–74.

[3] Cormen T.H., Leiserson C.E., Rivest R.L., *Introduction of Algorithms*, Second Edition, The MIT Press, 2001.
[4] Hart E.P., Nilson N.J., Raphael B., *A formal basis for the heuristic determination of minimum cost path*, IEEE Trans. Syst. Sci. Cybern. **4** (1968), 100–107.
[5] Hersberger J., Suri S., *On computing Euclidean shortest paths in the plane*, Proc. 34 Annu. IEEE. Sympos. Found, Comput. Sci. (1993), 508–517.
[6] Ghosh S.K., Mount D.M., *An output-sensitive algorithm for computing visibility graphs*, SIAM J. Computing **20** (1991), 888–910.
[7] Kapoor S., Maheshwari S.N., Mitchell J.S.B., *An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane*, Discrete Comput. Geom. **18** (1997), 377–383.
[8] Mitchell J.S.B., *Geometric Shortest Paths and Network Optimization*, In: Handbook of Computational Geometry, Elsevier Science (J.-R. Sack and J. Urrutia, eds.), 2000, pp. 633–701.
[9] O'Rourke J., *Computational Geometry in C*, Cambridge University Press, New York, 1995.
[10] Pallottino S., *Shortest paths Method – Complexity Interrelations and New Propositions*, Consiglio Nazionale delle Ricerche, Istituto per le Aplicazioni del Calcolo "Mauro Picone", Roma, **14** (1984), 257–267.
[11] Pohl I., *Heuristic search viewed as path finding in a graph*, Artif. Intell. **1** (1970), 193–204.

Institute of Mathematics
Silesian University
Bankowa 14
40-007 Katowice
Poland
and
Institute of Physics
Silesian University
Uniwersytecka 4
40-007 Katowice
Poland
e-mail: annawojak@gmail.com