Scientia Iranica D (2017) 24(3), 1335–1343

Sharif University of Technology
Scientia Iranica
Transactions D: Computer Science & Engineering and Electrical Engineering
www.scientiairanica.com

Research Note

Time complexity of two disjoint simple paths

M. Razzazi[a] and A. Sepahvand[a,b,*]

a. *Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, P.O. Box 15875-4413, Iran.*
b. *School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, P.O. Box 19395-5746, Iran.*

Received 26 April 2015; received in revised form 22 February 2016; accepted 14 May 2016

KEYWORDS
Hamiltonian path;
NP-complete;
Planar graph;
Simple path.

Abstract. Finding two disjoint simple paths on two given sets of points is a geometric problem introduced by Jeff Erickson. This problem has various applications in computational geometry, e.g. robot motion planning, generating polygon, etc. We will present a reduction from planar Hamiltonian path to this problem, and prove that it is NP-complete. To the best of our knowledge, no study has considered its complexity up until now. We also present a reduction from planar Hamiltonian path problem to the problem of "finding a path on given points in the presence of arbitrary obstacles" and prove that it is also NP-complete. Also, we present a heuristic algorithm with time complexity of $O(n^4)$ to solve this problem. The proposed algorithm first calculates the convex hull for each of the entry points and then produces two simple paths on the two entry point sets.

1. Introduction

This problem has various applications in path planning, VLSI, etc. Assume there are two pairs of sets of robots R_1 and R_2 where robots in R_1 and R_2 give sets of services s_1 and s_2, respectively; R and B sites (points) in a set need the sets of services s_1 and s_2, respectively. The amount of time which each robot spends to give a service is not fixed. We want to find a simple path within each set of R and B so that these two paths are disjoint and the robots of one set (R or B) can be stationed at one end point of the related path to start offering the services. By choosing two simple and disjoint paths, we avoid collision of robots.

In the mathematical field of graph theory, the Hamiltonian path problem and the Hamiltonian cycle problem are problems of determining whether a Hamiltonian path (a path is an undirected or directed graph that visits each vertex exactly once) or a Hamiltonian cycle exists in a given graph (whether directed or undirected). Both problems are NP-complete [1,2].

There is a simple relation between the problems of finding a Hamiltonian path and a Hamiltonian cycle. In one point of view, the Hamiltonian path problem for graph G is equivalent to the Hamiltonian cycle problem in a graph H obtained from G by adding a new vertex and connecting it to all vertices of G. Thus, finding a Hamiltonian path cannot be significantly slower (in the worst case, as a function of the number of vertices) than finding a Hamiltonian cycle. In another point of view, a graph G has a Hamiltonian cycle using edge uv if and only if the graph H obtained from G by replacing the edge by a pair of vertices of degree 1, one connected to u and one connected to v, has a Hamiltonian path. Therefore, by trying this replacement for all edges incident to some chosen vertex of G, the Hamiltonian cycle problem can be solved by at most n Hamiltonian path computations, where n is the number of vertices in the graph [1].

The Hamiltonian cycle problem is also a special case of the travelling salesman problem, obtained by setting the distance between two cities to one if they are

*. *Corresponding author.*
E-mail address: A.sepahvand@aut.ac.ir (A. Sepahvand)

adjacent and to two otherwise, and verifying that the total distance travelled is equal to n (if so, the route is a Hamiltonian circuit; if there is no Hamiltonian circuit, then the shortest route will be longer) [3].

There are several different definitions of path; simple path is a sequence of points connected to each other with line segments such that the segments do not intersect each other. Self-intersecting path is like a simple path but segments intersect each other. A close path is a simple or self-intersecting path where there exists a line segment between the first and the last points of the path.

1.1. Drawing two disjoint simple paths on two sets of points

Jeff Erickson in [4] introduced the problem of finding two simple paths that had no intersection together. In technical terms, given two sets of points X, and Y in the plane, how we can find two disjoint simple paths from the whole points of each set or report that no such paths exist. Figure 1 shows the problem. The points are in general position. A simple path may also be called *polygonal chain, polygonal curve [5], polygonal path [6], polyline [7]*, or piecewise *linear curve [7]*.

This problem has many potential applications in computational geometry, e.g. navigation, VLSI, robot motion planning, network design, etc.

This paper is organized as follows. Section 2 will review related studies; Section 3 will present an NP-complete proof for the problem and reduction details; in Section 4, we propose a heuristic algorithm for the problem mentioned above; and in Section 5, the conclusion and suggestions for future works will be presented.

2. Related works

Let X be a finite set of points and $x_0, x_1, ..., x_k \in X$ be some of the points of X; then, L is called an (s, X, t)-path. If there exists a path which starts at $s = x_0$, goes through vertices $x_1, ..., x_{(k-1)}$, and ends at $t = x_k$; and if Φ is any subset of the plane, then, we say that L avoids Φ if L does not intersect Φ, except for possibly at points s and t. Cheng, Chrobak, and Sundaram [8] presented an NP-complete proof for the problem of computing a simple (s, X, t)-path that avoided ϕ.

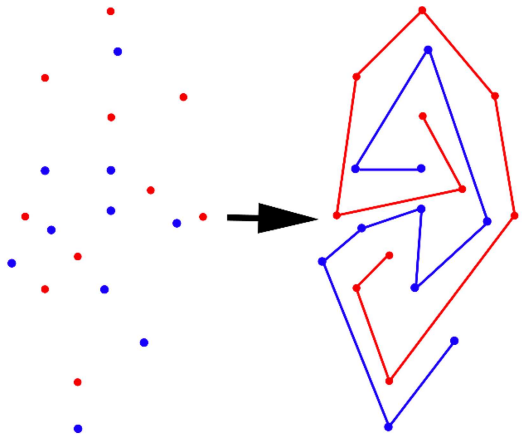

Figure 1. Given two sets of red and blue points, find a path from red points (L1) and a path from blue points (L2) such that L1 and L2 are disjoint [7].

Qi Cheng et al. in [8] also showed that the problem was solvable in polynomial time in a special case. Given a set of points, X, inside a polygonal region P, and two distinguished points $s, t \in X$, they studied the problem of finding the simple polygonal paths that turn only at the points of X and avoid the boundary of P, from s to t. Qi Cheng et al. in [8] presented an $O(m^2n^2)$ time and space algorithm. Xuehou Tan and Bo Jiang in [9] reviewed this problem and showed that it can be solved by $O((n^2 + m)logm)$ time, $O(n^2 + m)$ space algorithm for computing a simple path or reporting that no such path exists, where n is the number of points of X and m is the number of vertices of P.

Sometimes we may wish to generate a polygon that uses all points of X, not just a subset. This naturally leads to the problem of computing simple Hamiltonian (s, X, t)-paths (that is, simple (s, X, t)-paths that visit all points of X) that avoid Φ. It is easy to see that the problem is NP-complete for arbitrary obstacles and so is when we restrict our attention to the case where $\Phi = P$ is a simple polygon and X is inside (or outside) P. If P is convex, a simple Hamiltonian (s, X, t)-path that avoids P always exists and can be computed in time $O(nlogn)$, by using angular orderings of the points in X in an appropriate fashion. However, the status of this problem remains open when P is an arbitrary simple polygon [8].

Alsuwaiyel and Lee [10] showed that finding a Hamiltonian (s, X, t)-path (not necessarily simple) in a simple polygon P is NP-complete. Their proof works even in the special case when X is restricted to be the vertex set of P. (Note that the boundary of P is not a feasible solution if s and t are not consecutive.)

Erickson and LaValle in [11] presented an NP-Hard motion planning problem, which included path planning in situations where crossing an obstacle was costly but not impossible, to find the path that crossed the fewest obstacles. There are, not closely related, problems, including [12-14], which consider different versions of finding disjoint paths inside a set of sources and a set of targets.

3. Complexity result

In this section, we will prove that *drawing two disjoint simple paths on two sets of points* (defined in Section 1.1) is NP-complete. At first, we will present the proof idea of reduction, then will prove the mentioned problem.

3.1. Proof idea

A planar graph with fixed planar embedding is called a plane graph. To prove that our problem is NP-complete, first, we prove that the problem of 'finding Hamiltonian path in straight-line plane graph' is NP-complete and, then, we reduce this special case of Hamiltonian path problem to our own problem in polynomial time.

Theorem 1. Finding Hamiltonian path in any (directed or undirected) planar graph is NP-complete [15].

Theorem 2. Planarity testing can be conducted in linear time [16-17].

Theorem 3. In linear time, it is possible to find planar embedding from a planar graph [17-19].

Theorem 4. Any plane graph in linear time can be converted to straight-line embedding of the graph [20-22].

We call the straight-line embedding of a plane graph Straight-Line Plane Graph (SLPG). Algorithms for constructing planar line segment grid drawings, where the edges had integer coordinates, were developed by de Fraysseix, Pach, and Pollack [20] (shift method) and by Schnyder [21] (realizer method). They independently showed that every n-vertex planar graph had a planar line segment grid drawn with the height $O(n)$ and the width $O(n)$, resulting in the area of $O(n^2)$. Fraysseix et al. [20] conjectured that its complexity could be improved to $O(n)$. This bound was in fact achieved a few years later by Chrobak and Payne in [22].

Theorem 5. The problem of finding Hamiltonian path in a straight-line plane graph is NP-complete.

Proof. Using Theorems 1 to 4, we can simply conclude that finding Hamiltonian path in SLPG is NP-complete.□

We then call "Hamiltonian path problem in SLPG" ham-path problem.

Lemma 1. Any planar graph $G = (V, E)$ can be converted to two sets of points $U = V$, and $W = \{w_1, w_2, ..., w_k\}$ (w_is are points) in the plan such that $\forall u, v \in V$; if $(u, v) \in E$, then $u, v \in U$ are visible together; else $\exists w_i$ such that u, v, w_i are collinear and w_i is between u, v; it means, $u, v \in U$ are not visible together because w_i blocks their visibility as an obstacle.

Proof. According to Theorems 2 to 4, any planar graph can be converted to an SLPG and any SLPG $G = (V, E)$ can be converted to two sets of points $U = V$, $W = \{w_1, w_2, ..., w_k\}$ with the following algorithm:

Convert SLPG
Input: graph $G = (V, E)$
Output: two sets of points U, W with the mentioned condition in Lemma 1.
Begin
  Set $U = V$
  Set $k = 0$
  Set $w = \phi$
  Define $H = (U, E')$ a complete graph
  For each $(u, v) \in E'$
  Begin
    If $(u, v) \notin E$
    Begin
      Define w_k a point between u, v
      Set $W = W \cup w_k$
      Set $k = k + 1$
    End
  End
  Return U, W
End

Execution of the above algorithm on a sample input and its output is shown in Figure 2 (as set of red and blue points).

Clearly the condition mentioned in Lemma 1 is satisfied.□

Observation 1. Consider W as obstacles (blue points in Figure 2); if we can find a path containing all the points in U in such a way that the path does not cross the obstacles, clearly, we can find a Hamiltonian path in G, because two vertices are visible in U if there exists an edge between them in G.

Theorem 6. Finding a path on the given points and arbitrary obstacles in the plane is NP-complete.

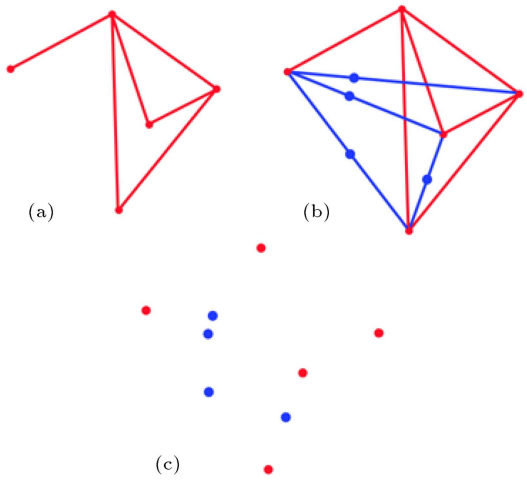

Figure 2. (a) Input graph SLPG $G = (V, E)$. (b) Complete graph $H = (U, E')$. Blue edges are not in G. (c) Output of the algorithm that includes two sets of points with the conditions mentioned in Lemma 1.

Proof. Directly concluded from lemma 1 and observation 1.□

Until now we have not proven that our defined problem is NP-complete, but in Section 3.2 we will present a reduction from *ham-path* using the above-mentioned idea.

3.2. Details of reduction

To prove that the problem of "drawing two disjoint simple paths on two sets of points" (*disjoint path* for short) is NP-complete, we will reduce *ham-path* problem to it as follows.

Given a planar graph $G = (V, E)$ with line segments as edges, with the function below, we can make two sets of points U, W such that $U = V$; and if there exists a path containing the points of U, there exists a path containing the points of W too.

Definition 1. For each vertex $v \in V$, extending each edge connected to v is called an *extending edge of* v. These edges are called extended edges. This extension will divide the plane into some regions (Figure 3).

Convert Function 2

```
1  Input: graph G = (V, E)
2  Output: two sets of points with the mentioned
          condition in Lemma 1 (and some more
          points as described later)
3  Begin
4      Set U = V
5      Set k = 0
6      Set w = φ
7      Define H = (U, E') a complete graph
8      For each (u, v) ∈ E'
9      Begin
10         If (u, v) ∉ E
11         Begin
12             Define w_k, w_{k+1} two points
               between u, v with ||w_k − u|| = ε
               and ||w_{k+1} − v|| = ε
13             Set W = W ∪ {w_k, w_{k+1}}
14             Set k = k + 2
15         End
16     End
17     For each vertex v ∈ V
18     Begin
19         Extend edges of v
20         For each region r_i of v
21         Begin
22             If ∄ w_i ∈ W in region r_i
                with ||w_i − v|| = ε
23             Begin
24                 Define w_k a point in
                   region r_i with ||w_k − v|| = ε
25                 Set W = W ∪ w_k
26                 Set k = k + 1
27             End
28         End
29         For each extended edge e of v
30             Begin
31                 Define w_k a point on e
                   with ||w_k − v|| = ε
32                 Set W = W ∪ w_k
33                 Set k = k + 1
34             End
35     End
36     Return U, W
37 End
```

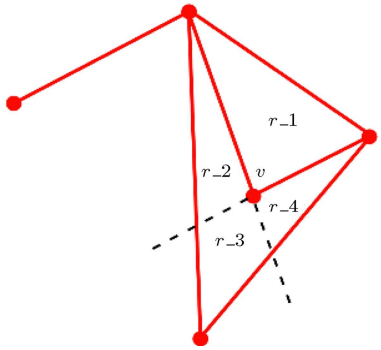

Figure 3. Extending edges of vertex v and respective regions. Dash lines are extended edges.

The output of *Convert Function 2* includes two sets of points, U and W, such that $U = V$ and W contains some points that satisfy the condition of Lemma 1 and some other points. Points inserted into W are shown in Figure 4. These points are added in such a way that all of them can be connected to each other as a cycle if there exists a path containing the points of U.

Claim 1. If there exists a path containing the points of U, there exists a cycle containing the points of W.

Proof. Points added to W are added in such a way that guarantee the above claim. If there exists a simple path containing U, we add the points to W such that we can have a cycle just moving near the path with about epsilon distance from it (Figure 5).

To explore more, let $L = \{p_1, p_2, ..., p_k\}$ be the path that contains the points of U such that p_1, and p_k are the end points (just one edge is connected to them). In the above algorithm, we inserted at least three points in W with epsilon distance from these end points (Figure 6(a)).

Knowing this, we can connect these points (three or more) together as it is shown in Figures 6(b) and 5. We need these properties to make a simple path containing the points of W.

For $v = p_i, 1 < i < k$, there exists at least five points in W that are on the concave side of point p_i with distance epsilon from p_i (Figure 7(a)), and there exists at least one point in W that is on the convex side of the point p_i with distance epsilon from p_i (Figure 7(a)).

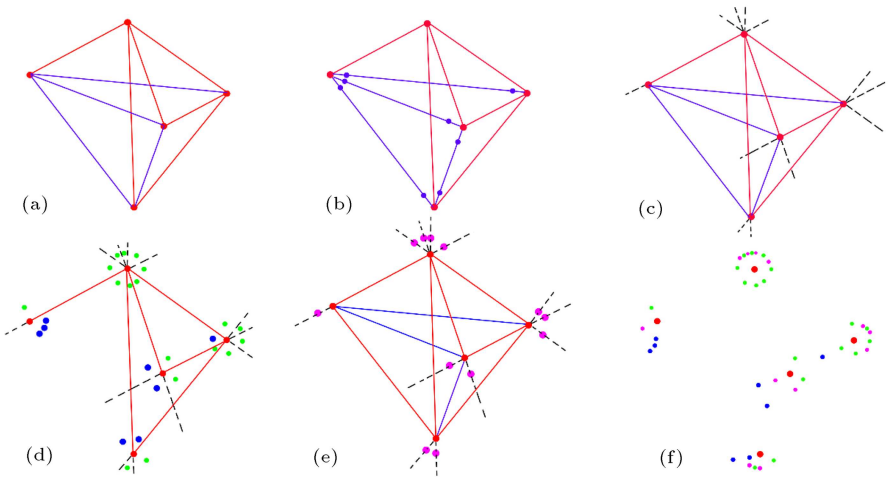

Figure 4. The output of Convert Function 2: (a) Red points and red lines represent input graph $G = (V, E)$ and all line segments (blue and red) represent the complete graph $H = (U, E')$, (b) lines 8-16 of the function will add blue points to W, (c) black lines are extended edges of $v \in V$ (line 19 of the function), (d) lines 20-28 of the function will add green points to W, (e) lines 29-34 of the function will add purple points to W, and (f) output of the algorithm: red points are in U and other points are in W (different colors for points are used for well understanding; all points in W play the same role).

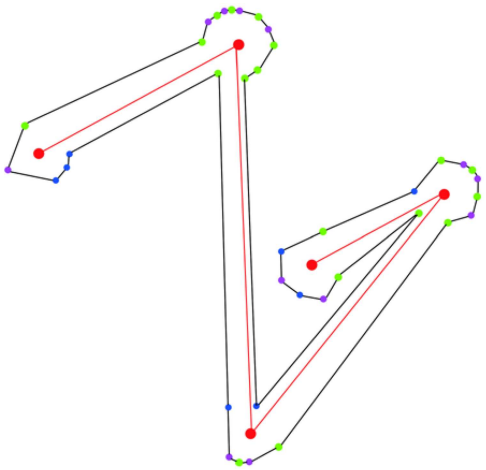

Figure 5. Red line segments are a path containing the points of U. Black line segments are a cycle containing the points of W.

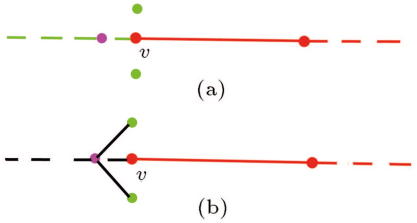

Figure 6. (a) There are at least three points in W with distance epsilon from the end point v. (b) The points can easily be connected with epsilon distance from v.

We can easily connect all such points on the concave side as shown in Figure 7(b). This trick is useful for turning around $p_i, 1 < i < k$ and to build a simple path containing all the points of W. By continuing these connecting points (as mentioned), there will be a chain containing the points of W.□

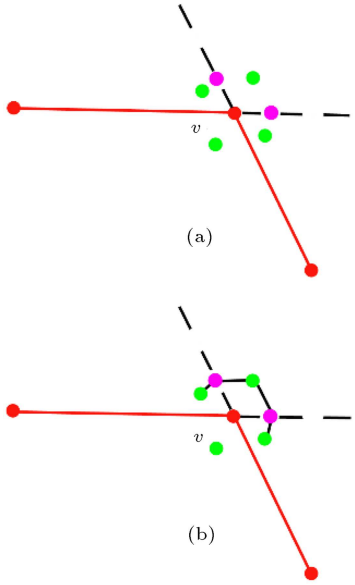

Figure 7. There exist at least five points on the concave side of vertex v and at least one point on its convex side.

As mentioned in Observation 1 and Claim 1, we can find a path from U if and only if we can find a Hamiltonian path from V. Thus, we have the following theorem.

Theorem 7. "Drawing two disjoint simple paths on two sets of points" is NP-complete.

Proof: Clearly, if we have two disjoint simple paths, simply, we can verify their disjointness in polynomial time; thus, the problem is NP. Because of the reduction mentioned in this section, we can conclude that the problem is NP-complete.□

4. The proposed algorithm

In this section, we present a heuristic algorithm in order to find two simple and distinct Paths from the set of entry points, namely, red (R) and blue (B). The objective is to minimize the intersection points between the two obtained paths.

First, we separately generate the convex hull for each R and B sets and we call the sets of edges of these convex hulls RCH and BCH, respectively (Figure 8(b)). By randomly removing an edge from RCH and BCH so that they have the highest numbers of intersections with the other CH, we obtain two simple red and blue paths (Figure 8(c)). Then, according to Algorithm 1, we add to each path, depending on the color of the path, those points in R or B sets that are not members of the path. Assume p is any given point and q is one of the two end points of a path x; we say q is visible from p (similarly p is visible from q) if pq (qp) does not intersect path x, except at $q(p)$. The point p is visible form edge (u, v) of path x, if neither edge (p, u) nor (p, v) intersect with path x (also edge (u, v) is visible from point p). For example, in Figure 8(d), point rp is visible from e_1, e_4, p_1, and p_2 of the red path, and point rp is not visible form e_2 and e_3 of the red path.

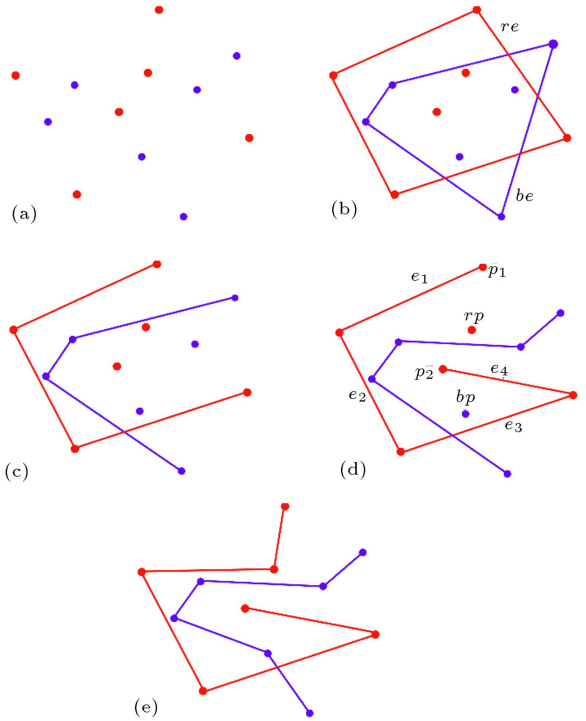

Figure 8. (a) The entry points sets. (b) Convex hulls of red and blue points sets are generated and the re edge is selected randomly from the convex hull of the red points set, and the be edge from the convex hull of blue points set in order to be removed. (c) Omission of re and be edges from convex hull. (d) The rp point is selected to be added to the red path and is *visible* from e_1, e_4, p_2, and p_1. (e) The output of applying the algorithm.

In Algorithm 1 we first generate the convex hull of two sets. Then, we remove one of the edges which has the most intersections with the other convex hull (lines 5-8). We start to add free we call a red (blue) point free if it is not connected to the red (blue) chain yet. red and blue points to corresponding chains (lines 9-51). Lines 10-29 add the free red points to the red chain and lines 30-50 add the free blue points to the blue chain. A red point rp is added to a red path x by doing the following.

rp is connected to any endpoint of the path x which is visible from rp; otherwise, it finds an edge (u, v) of path x which is visible from rp, removes (u, v) from path x, and adds edges (rp, u) and (rp, v) to path x. Similar steps are taken for adding a blue point to the blue path. In rare cases that such an endpoint or edge cannot be found (see Section 4.2), the algorithm is restarted. For example, Figure 8 shows these operations on the given sample red and blue points.

4.1. Analysis of the proposed algorithm

Assuming that $n = \max\{$the number of blue and red points$\}$, calculating the convex hull by using Graham's algorithm takes $O(nlogn)$. Checking whether the two edges have any intersection with each other or not takes a constant time. Steps 5-8 at most take the time $O(n^2)$. Most of the time needed for adding a point p to a path belong to finding the visible edges of a path from the point p; this takes the time $O(n^3)$. We add n points to the path so that time complexity of the entire algorithm becomes $O(n^4)$. In finding the above complexity, we use naive algorithms. Clearly, there are more efficient algorithms for visibility and finding intersections of two convex hulls, by using which complexity could be reduced significantly.

4.2. A special case of the algorithm

A condition may occur such that none of the remaining points from a point set is visible from any of the edges or points at the two ends of the path. If this special case occurs, we execute Algorithm 1 from the beginning. Figure 9 shows an example of this special case.

We executed the proposed algorithm 100,000 times on a point set which included the set point of

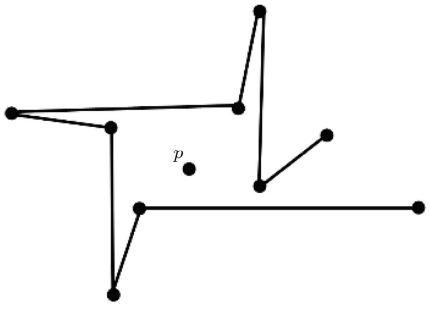

Figure 9. The point p is not visible from any of the edges or end points of a path.

Input: two sets of red (R) and blue (B) points.
Output: Two simple paths from each set.

Generate the convex hull for each R and B sets (Figure 8(b)). We call the sets of edges of these convex hulls RCH and BCH, respectively.
Let $V(RCH)$ and $V(BCH)$ be sets of endpoints of edges in RCH and BCH, respectively.
Initialize the set $M = \phi$, $N = \phi$, $V = \phi$, and $W = \phi$.
Add edges from RCH to M such that they have the most crossing points with BCH.
Add edges from BCH to N such that they have the most crossing points with RCH.
Choose edge $(u_r, v_r) \in M$ randomly and remove (u_r, v_r) from RCH.
Choose edge $(u_b, v_b) \in N$ randomly and remove (u_b, v_b) from BCH.
Do {
  If $(R - V(RCH) \neq \phi)$
  {
    Choose point $rp \in R - V(RCH)$ randomly.
    Add u_r, v_r and all edges from RCH to W where they are visible from rp.
    If u_r, v_r and all edges from RCH are not visible from rp and there exist unprocessed points in $R - V(RCH)$
      Go to Step 12 and select another point.
    Else
      Restart the program.
    Choose $w \in W$ randomly. (*it can be either a point or an edge*).
    If $(w \in \{u_r, v_r\})$ {
      Connect rp to w and add edge (rp, w) to RCH.
      If $(w = u_r)$
        $u_r = rp$ (*u_r is now the new endpoint of the red path*).
      Else
        $v_r = rp$ (*v_r is now the new endpoint of the red path*).
    } **Else** (*an edge is chosen*)
      Remove $w = (w_1, w_2)$ from RCH and insert (w_1, rp) and (w_2, rp) in RCH.
    Set $W = \phi$.
  }
  If $(B - V(BCH) \neq \phi)$
  {
    Choose point $bp \in B - V(BCH)$ randomly.
    Add u_b, v_b and all edges from BCH to V where they are visible from bp.
    If u_b, v_b and all edges from BCH are not visible from bp and there exist unprocessed points in $B - V(BCH)$
      Go to Step 32 and select another point.
    Else
      Restart the program.
    Choose $v \in V$ randomly. (*it can be either a point or an edge*).

    If $(v \in \{u_b, v_b\})$ {
      Connect bp to v and add edge (bp, v) to BCH.
      If $(v = u_b)$
        $u_b = bp$ (*u_b is now the new endpoint of the blue path*).
      Else
        $v_b = bp$ (*v_b is now the new endpoint of the blue path*).
    } **Else** (* an edge is chosen*)
      Remove $v = (v_1, v_2)$ from BCH and insert (v_1, bp) and (v_2, bp) in BCH.
    Set $V = \phi$.
  }
} **While** $(R - RCH \neq \phi$ or $B - BCH \neq \phi)$.

Algorithm 1. Computing two disjoint simple paths from two sets of red and blue points.

Figure 9, and in 99,742 executions the algorithm successfully produced a simple path in the first run. Also, in order to successfully test the proposed algorithm, we used point-set cardinalities: 10, 20, 30,..., 100, 130, 160, 190, and 220. For each cardinality, we randomly generated 1000 pairs of point sets, one for the set of red points and the other for the set of blue points. The proposed algorithm was executed 10,000 times on each pair of point sets. Figure 10 shows the probability of the algorithm's success at the first run.

4.3. The proposed algorithm test

In order to test the proposed algorithm, we used point-set cardinalities: 10, 20, 30,..., 90, 100. For each cardinality, we randomly generated 100 pairs of point sets, one for the set of red points and the other for the set of blue points. We executed the proposed algorithm 1000 times on each pair of point sets. Table 1 shows the obtained results.

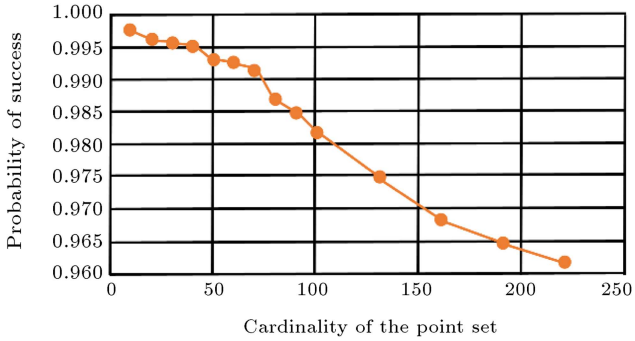

Figure 10. The probability of success of the proposed algorithm at the first run on the set of entry points.

We designed an exact algorithm with exponential time complexity to obtain the optimum solution. Using this exact algorithm, we carried out the following tests: we used point-set cardinalities: $5, 6, 7, \ldots, 11, 12$; for each cardinality, we randomly generated 100 pairs of

Table 1. The results obtained from the implementation of the proposed algorithm.

Number of entry points from each color	The minimum number of created intersections by the proposed algorithm	The maximum number of created intersections by the proposed algorithm	The average number of created intersections by the proposed algorithm
10	0	7	1.95
20	0	19	5.41
30	1	27	10.63
40	1	35	15.16
50	4	47	19.10
60	8	53	24.87
70	11	60	29.59
80	16	77	33.77
90	17	81	37.63
100	22	86	37.54

Table 2. The results of executing the optimal algorithm and the proposed algorithm.

Number of entry points from each color	The minimum number of created intersections by the proposed algorithm	The maximum number of created intersections by the proposed algorithm	The average number of created intersections by the proposed algorithm
5	3	0.34	0.64
6	5	0.63	0.83
7	4	0.91	1.17
8	6	0.98	1.34
9	5	1.11	1.5
10	5	1.13	1.91
11	6	1.43	2.13
12	9	1.57	2.51

point sets, each pair consisting of a set of red points and a set of blue points, with both sets having the same cardinality. For each pair of point sets, we ran the proposed algorithm 1000 times and calculated the average and the maximum number of intersections. Table 2 shows the results.

5. Conclusion and future works

In this paper, we presented proof of NP-completeness for finding two disjoint simple paths on two given sets of points. Also, we proved that finding a path on a given set of points in presence of arbitrary obstacles is NP-complete. These proofs were done by reduction from planar Hamiltonian path problem. Finding two disjoint paths on two given sets of points has application in robots motion planning, polygon generation, etc. Also, we proposed a heuristic algorithm to solve this problem in polynomial time, the objective of which was to minimize the number of intersection points between the two paths.

We discussed the problem in the two-dimensional space. As a future work, this problem can be generalized to higher dimensions. Another interesting problem is finding a path on the given points while a path as an obstacle exists. By solving this problem, some problems mentioned in [8] may be easily solved.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. This research was in part supported by a grant from IPM (No. CS1396-5-01).

References

1. Michael, R.G and David, S.J., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, pp. 199-200, W.H. Freeman, US (1979).
2. Karp, R. "Reducibility among combinatorial prob-

lems", In *Complexity of Computer Computations*, R. Miller, J. Thatcher and J. Bohlinger, pp. 85-103, Springer, US (1972).

3. Gupta, P. and Varshney, M., *Design and Analysis of Algorithms*, p. 374, PHI Learning, New Dehli (2012).
4. Erickson, J. "Generating random simple polygons", (1999, 2001). From http://jeffe.cs.illinois.edu/open/randompoly.html.
5. Gomes, J., Velho, L. and Sousa, M.C., *Computer Graphics: Theory and Practice*, p. 186, CRC Press, New York (2012).
6. Cheney, W., *Analysis for Applied Mathematics*, Springer Science & Business Media, New York, p. 13 (2001).
7. Boissonnat, J.-D. and Teillaud, M. (eds.)., *Effective Computational Geometry for Curves and Surfaces*, p. 34, Springer, Berlin (2007).
8. Cheng, Q., Chrobak, M. and Sundaram, G. "Computing simple paths among obstacles", *Computational Geometry,* **16**(4), pp. 223-233 (2000).
9. Tan, X. and Jiang, B. "Finding Simple Paths on Given Points in a Polygonal Region", In *Frontiers in Algorithmics*, J. Chen., J. Hopcroft. and J. Wang., pp. 229-239, Springer International Publishing, China (2014).
10. Alsuwaiyel, M.H. and Lee, D.T. "Minimal link visibility paths inside a simple polygon", *Computational Geometry*, **3**(1), pp. 1-25 (1993).
11. Erickson, L. and Lavalle, S.M. "A simple, but NP-Hard, motion planning problem", *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, North America (2013).
12. Bermond, J.C., David, C., Gianlorenzo, D. and Fatima Zahra, M. "Finding disjoint paths in networks with star shared risk link groups", *Theoretical Computer Science*, **579**, pp. 74-87 (2015).
13. Park, J.-H., Joonsoo, C. and Hyeong-Seok, L. "Algorithms for finding disjoint path covers in unit interval graphs", *Discrete Applied Mathematics*, **205**, pp. 132-149 (2016).
14. Xi, W., Jianxi, F., Xiaohua, J. and Cheng-Kuan, L. "An efficient algorithm to construct disjoint path covers of DCell networks", *Theoretical Computer Science*, **609**(1), pp. 197-210 (2016).
15. Garey, M.R., Johnson, D.S. and Tarjan, R.E. "The planar Hamiltonian circuit problem is NP-complete", *SIAM Journal on Computing*, **5**(4), pp. 704-714 (1976).
16. Hopcroft, J. and Tarjan, R.E. "Efficient planarity testing", *J. ACM*, **21**(4), pp. 549-568 (1974).
17. Shih, W.K. and Hsu, W.-L. "A new planarity test", *Theoretical Computer Science*, **223**(1), pp. 179-191 (1999).
18. Mehlhorn, K. and Mutzel, P. "On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm", *Algorithmica*, **16**(2), pp. 233-242 (1996).
19. Chiba, N., Nishizeki, T., Abe, S. and Ozawa, T. "A linear algorithm for embedding planar graphs using PQ-trees", *Journal of Computer and System Sciences*, **30**(1), pp. 54-76 (1985).
20. De Fraysseix, H., Pach, J. and Pollack, R. "How to draw a planar graph on a grid", *Combinatorica*, **10**(1), pp. 41-51 (1990).
21. Schnyder, W. "Embedding planar graphs on the grid", *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, pp. 138-147 (1990).
22. Chrobak, M. and Payne, T.H. "A linear-time algorithm for drawing a planar graph on a grid", *Information Processing Letters*, **54**(4), pp. 241-246 (1995).

Biographies

Mohammadreza Razzazi received the MS degree in Computer Science from Stanford University and the PhD degree in Computer Science from the University of California at Santa Barbara. Currently, he is an Professor in Computer Engineering & IT Department at Amirkabir University of Technology (AUT) in Tehran. His primary areas of research are computational geometry, robotics, and computer graphics.

Abdolah Sepahvand received the BSc degree in Computer Engineering in 2013 from Shahid Rajaee Teacher Training University (SRTTU) and MSc degree in Computer Engineering in 2015 from Amirkabir University of Technology (AUT). His research interests include computational geometry and approximation algorithms.