Simultaneous Time-Space Upper Bounds for Red-Blue Path Problem in Planar DAGs*

Diptarka Chakraborty and Raghunath Tewari

Department of Computer Science & Engineering,
Indian Institute of Technology, Kanpur, India
{diptarka,rtewari}@cse.iitk.ac.in

Abstract. In this paper, we consider the RedBluePath problem, which states that given a graph G whose edges are colored either red or blue and two fixed vertices s and t in G, is there a path from s to t in G that alternates between red and blue edges. The RedBluePath problem in planar DAGs is NL-complete. We exhibit a polynomial time and $O(n^{\frac{1}{2}+\epsilon})$ space algorithm (for any $\epsilon > 0$) for the RedBluePath problem in planar DAG. We also consider a natural relaxation of RedBluePath problem, denoted as EvenPath problem. The EvenPath problem in DAGs is known to be NL-complete. We provide a polynomial time and $O(n^{\frac{1}{2}+\epsilon})$ space (for any $\epsilon > 0$) bound for EvenPath problem in planar DAGs.

1 Introduction

A fundamental problem in computer science is the problem of deciding reachability between two vertices in a directed graph. This problem characterizes the complexity class non-deterministic logspace (NL) and hence is an important problem in computational complexity theory. Polynomial time algorithms such as Breadth First Search (BFS) algorithm and Depth First Search (DFS) give a solution to this problem, however they require linear space as well. On the other hand, Savitch showed that reachability can be solved by an $O(\log^2 n)$ space algorithm, however that takes $\Theta(n^{\log n})$ time [1]. The readers may refer to a survey by Wigderson [2] to know more about the reachability problem.

It is an important open question whether these two bounds can be achieved by a single algorithm. In other words can we exhibit a polynomial time and $O(\log^k n)$ space algorithm for the reachability problem in directed graphs. Barnes, Buss, Ruzzo and Schieber made some progress in this direction by giving the first polynomial time and sub-linear space algorithm. They showed that directed reachability can be solved by an $O(n/2^{k\sqrt{\log n}})$ space and polynomial time algorithm [3], by cleverly combining BFS and Savitch's algorithm. Till now this is the best known simultaneous time-space bound known for the directed reachability problem in this direction. Recently, Imai et. al. [4] have improved the BBRS bound for the class of directed planar graph. They gave a polynomial

* Research supported by Research-I Foundation.

M.S. Rahman and E. Tomita (Eds.): WALCOM 2015, LNCS 8973, pp. 258–269, 2015.

time and $O(n^{\frac{1}{2}+\epsilon})$ space algorithm by efficiently constructing a *planar separator* and applying a divide and conquer strategy. In a recent work, their result has been extended to the class of *high-genus* and *H-minor-free* graphs [5].

An interesting generalization of the reachability problem, is the RedBluePath problem. Given a directed graph where each edge is colored either Red or Blue, the problem is to decide if there is a (simple) directed path between two specified vertices that alternates between red and blue edges. Kulkarni showed that the RedBluePath problem is NL-complete even when restricted to planar DAGs [6]. Unfortunately, no sublinear ($O(n^{1-\epsilon})$, for any $\epsilon > 0$) space and polynomial time algorithm is known for RedBluePath problem in planar DAGs.

A natural relaxation is the EvenPath problem, which asks if there is a (simple) directed path of even length between two specified vertices in a given directed graph. In general, EvenPath problem is NP-complete [7], but for planar graphs, it is known to be in P [8]. It is also known that for DAGs, this problem is NL-complete. Datta *et. al.* showed that for planar DAGs, EvenPath problem is in UL. However, no sublinear ($O(n^{1-\epsilon})$, for any $\epsilon > 0$) space and polynomial time algorithm is known for this problem also.

In this paper, we provide a sublinear space and polynomial time bound for both the RedBluePath and EvenPath problem. Our main idea is the use of a space efficient construction of separators for planar graphs [4]. We then devise a modified DFS approach on a smaller graph to solve the RedBluePath problem. As a consequence, we show that a similar bound exists for the directed reachability problem for a class of graphs that is a superset of planar graphs. Using similar approach, we design an algorithm to detect the presence of an odd length cycle in a directed planar graph, which serves as a building block to solve the EvenPath problem.

Our Contributions

The first contribution of this paper is to give an improved simultaneous time-space bound for the RedBluePath problem in planar DAGs.

Theorem 1. *For any constant $0 < \epsilon < \frac{1}{2}$, there is an algorithm that solves* RedBluePath *problem in planar DAGs in polynomial time and $O(n^{\frac{1}{2}+\epsilon})$ space.*

We first construct a separator for the underlying undirected graph and perform a DFS-like search on the separator vertices.

Using the reduction given in [6] and the algorithm stated in the above theorem, we get an algorithm to solve the directed reachability problem for a fairly large class of graphs as described in Section 3, that takes polynomial time and $O(n^{\frac{1}{2}+\epsilon})$ space. Thus we are able to beat the BBRS bound for such a class of graphs. One such class is all k-planar graphs, where $k = O(\log^c n)$, for some constant c and this is a strict superset of the set of planar graphs.

In this paper, we also establish a relation between EvenPath problem in a planar DAG and the problem of finding odd length cycle in a directed planar graph and thus we argue that both of these problems have the same simultaneous time-space complexity. We use two colors Red and Blue to color the vertices of the given graph

and then use the color assigned to the vertices of the separator to detect the odd length cycle. The conflicting assignment of color to the same vertex in the separator will lead to the presence of an odd length cycle. Here also we use the recursive approach to color the vertices and as a base case we use BFS to solve the problem of detecting odd length cycle in each small component. Thus we have the following result regarding the EvenPath problem.

Theorem 2. *For any constant $0 < \epsilon < \frac{1}{2}$, there is an algorithm that solves* EvenPath *problem in planar DAGs in polynomial time and $O(n^{\frac{1}{2}+\epsilon})$ space.*

The rest of the paper is organized as follows. In Section 2, we give some basic definitions and notations that we use in the rest of the paper. In Section 3, we prove our main results regarding the RedBluePath and EvenPath problem.

2 Preliminaries

A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E where each edge can be represented as an ordered pair (u, v) in case of directed graph and as an unordered pair $\{u, v\}$ in case of undirected graph, such that $u, v \in V$. Unless otherwise specified, G will denote a directed graph, where $|V| = n$. Given a graph G and a subset of vertices X, $G[X]$ denotes the subgraph of G induced by X and $V(G)$ denotes the set of vertices present in the graph G. Given a directed graph G, we denote the underlying undirected graph by $\hat{G}$. We follow the standard model of computation to discuss the complexity measures of the stated algorithms. In particular, we consider the computational model in which an input appears on a read-only tape and the output is produced on a write-only tape and we only consider the internal read-write tape in the measure of space complexity. Throughout this paper, by $\widetilde{O}(s(n))$, we mean $O(s(n)(\log n)^{O(1)})$.

The notions of separator and separator family defined below are crucial in this paper.

Definition 1. *A subset of vertices S of an undirected graph G is said to be a ρ-separator (for any constant ρ, $0 < \rho < 1$) if removal of S disconnects G into two sub-graphs A and B such that $|A|, |B| \leq \rho n$ and the size of the separator is the number of vertices in S.*

A subset of vertices $\overline{S}$ of an undirected graph G with n vertices is said to be a $r(n)$-separator family if the removal of $\overline{S}$ disconnects G into sub-graphs containing at most $r(n)$ vertices.

Now we restate the results and the main tools used in [4] to solve directed planar reachability problem and these results are extensively used in this paper. In [4], the authors construct a $\frac{8}{9}$-separator for a given undirected planar graph.

Theorem 3 ([4]). *(a) Given an undirected planar graph G with n vertices, there is an algorithm* PlanarSeparator *that outputs a $\frac{8}{9}$-separator of G in polynomial time and $\widetilde{O}(\sqrt{n})$ space.*
(b) For any $0 < \epsilon < 1/2$, there is an algorithm PlanarSeparatorFamily *that takes an undirected planar graph as input and outputs a $n^{1-\epsilon}$-separator family of size $O(n^{\frac{1}{2}+\epsilon})$ in polynomial time and $\widetilde{O}(n^{\frac{1}{2}+\epsilon})$ space.*

In [4], the above theorem was used to obtain a new algorithm for reachability in directed planar graph.

Theorem 4 ([4]). *For any constant $0 < \epsilon < 1/2$, there is an algorithm* `DirectedPlanarReach` *that, given a directed planar graph G and two vertices s and t, decides whether there is a path from s to t. This algorithm runs in time $n^{O(1/\epsilon)}$ and uses $O(n^{1/2+\epsilon})$ space, where n is the number of vertices of G.*

3 Red-Blue Path Problem

3.1 Deciding Red-Blue Path in Planar DAGs

Given a directed graph G with each edge colored either Red or Blue and two vertices s and t, a *red-blue path* denotes a path that alternate between Red and Blue edges and the RedBluePath problem decides whether there exists a directed red-blue path from s to t such that the first edge is Red and last edge is Blue. The RedBluePath problem is a generalization of the reachability problem in graphs, however this problem is NL-complete even when restricted to planar DAGs [6]. This makes it an interesting problem in the area of space bounded complexity as to the best of our knowledge, this is the only "reachabililty-like" problem in planar graphs that is hard for NL. We will now give a proof of Theorem 1.

Proof (of Theorem 1). Consider a planar DAG G. Let $\overline{S}$ be the $n^{(1-\epsilon)}$-separator family computed by `PlanarSeparatorFamily` on $\hat{G}$ and let $S = \overline{S} \cup \{s, t\}$. For the sake of convenience, we associate two numerical values to the edge colors – 0 to Red and 1 to Blue. We run the subroutine `RedBluePathDetect` (Algorithm 3) with the input $(G, s, t, n, 0, 1)$ and if the returned value is true, then there is a directed red-blue path from s to t such that the first edge is Red and last one is Blue. In Algorithm 2, we use the notation $(u, v) \in^{(init,temp)} \overline{E'}$ to decide whether there is a red-blue path from u to v that starts with an edge of color value *init* and ends with an edge of color value *temp*.

In Algorithm 1, we use general DFS type search to check the presence of a red-blue path between any two given vertices s' and t'. The only difference with DFS search is that here we explore edges such that color of the edges alternates between red and blue. If we start from a vertex s', then the for loop (Lines 3 – 8) explore the path starting from s' such that first edge of the path is of specified color. In the main algorithm (Algorithm 3), we use Algorithm 1 as a base case, i.e., when the input graph is small in size (is of size $n^{1/2}$). Otherwise, we first compute S and then run Algorithm 2 on the auxiliary graph $\overline{G} = (S, \overline{E})$. Algorithm 3 does not store the graph $\overline{G}$. Whenever it is queried with a pair of vertices to check if it forms an edge, it recursively runs Algorithm 3 on all the connected components of $G[V \setminus S]$ separately (Lines 10 – 14 of Algorithm 2) and produces an answer. Finally, we perform same DFS like search as in Algorithm 1 on $\overline{G}$ (Lines 1 – 9 of Algorithm 2).

In the base case, we use Algorithm 1 which takes linear space and polynomial time. Thus due to the restriction of the size of the graph in the base case, we

Input : $G' = (V', E'), s', t', init, final$
Output : "Yes" if there is a red-blue path from s' and t' starts with $init$ and ends with $final$
/* Use two sets- N_i, for $i = 0, 1$, to store all the vertices that have been explored with the color value i */
1 **if** $s' \notin N_{init}$ **then**
2 Add s' in N_{init};
3 **for** *each edge* $(s', v) \in E'$ *of color value init* **do**
4 **if** $v = t'$ *and* $init = final$ **then**
5 Return true;
6 **end**
7 Run ColoredDFS$(G', v, t', init + 1 (mod\ 2), final)$
8 **end**
9 **end**

Algorithm 1. Algorithm ColoredDFS: One of the Building Blocks of RedBluePath Detect

Input : $\overline{G'} = (\overline{V}', \overline{E'}), G', s', t', init, final$
Output : "Yes" if there is a red-blue path from s' and t' starts with $init$ and ends with $final$
/* Use two sets- R_i, for $i = 0, 1$, to store all the vertices that have been explored with the color value i */
1 **if** $s' \notin R_{init}$ **then**
2 Add s' in R_{init};
3 **for** *each* $(s', v) \in^{(init,temp)} \overline{E'}$ *for each* $temp \in \{0, 1\}$ **do**
4 **if** $v = t'$ *and* $temp = final$ **then**
5 Return true;
6 **end**
7 Run ModifiedColoredDFS$(\overline{G}', G', v, t', temp + 1 \text{ (mod 2)}, final)$
8 **end**
9 **end**
/* ''$(u, v) \in^{(init,temp)} \overline{E'}$?'' query will be solved using the following procedure */
10 **for** *every* $a \in V$ **do**
 /* V be the set of vertices of G' */
 /* V_a = the set of vertices of $\hat{H}$'s connected component containing a, where $H = G[V \setminus \overline{V}']$ */
11 **if** RedBluePathDetect*$(G[V_a \cup \overline{V}'], u, v, n, init, temp)$ is true* **then**
12 Return true for the query;
13 **end**
14 **end**
15 Return false for the query;
/* End of the query procedure */

Algorithm 2. Algorithm ModifiedColoredDFS: One of the Building Blocks of RedBluePathDetect

Input : $G', s', t', n, init, final$
Output : "Yes" if there is a red-blue path from s' and t' starts with $init$ and ends with $final$

if $n' \leq n^{\frac{1}{2}}$ **then**
  Run `ColoredDFS`$(G', s', t', init, final)$;
else
  `/* let` $r' = n'^{(1-\epsilon)}$ `*/`
  Run `PlanarSeparatorFamily` on $\hat{G'}$ to compute r'-separator family $\overline{S'}$;
  Run `ModifiedColoredDFS`$(\overline{G'} = (\overline{S'} \cup \{s', t'\}, \overline{E'}), G', s', t', init, final)$;
end

Algorithm 3. Algorithm `RedBluePathDetect`: Algorithm for Red-Blue Path in planar DAG

have $\widetilde{O}(n^{1/2})$ space and polynomial time complexity. The sets N_0 and N_1 of algorithm `ColoredDFS` only store all the vertices of the input graph and we run `ColoredDFS` on a graph with $n^{1/2}$ vertices and it visits all the edges of the input graph at most once which results in the polynomial time requirement.

Let $\mathcal{S}$ and $\mathcal{T}$ denote its space and time complexity functions. Since $(1-\epsilon)^k \leq \frac{1}{2}$ for $k = O(\frac{1}{\epsilon})$, the depth of the recursion is $O(\frac{1}{\epsilon})$. Also, $|V_a \cup \overline{S'}| \leq 2n'^{(1-\epsilon)}$. This gives us the following recurrence relation:

$$\mathcal{S}(n') = \begin{cases} \tilde{\mathrm{O}}(n'^{(\frac{1}{2}+\epsilon)}) + \mathcal{S}(2n'^{(1-\epsilon)}) & \text{if } n' > n^{\frac{1}{2}} \\ \tilde{\mathrm{O}}(n^{\frac{1}{2}}) & \text{otherwise} \end{cases}$$

Thus, $\mathcal{S}(n) = O(\frac{1}{\epsilon})\tilde{\mathrm{O}}(n^{\frac{1}{2}+\epsilon}) = \tilde{\mathrm{O}}(n^{\frac{1}{2}+\epsilon})$.
For time analysis, we get the following recurrence relation:

$$\mathcal{T}(n') = \begin{cases} q(n)(p_1(n')\mathcal{T}(2n'^{(1-\epsilon)}) + p_2(n')) & \text{if } n' > n^{\frac{1}{2}} \\ q(n)\tilde{\mathrm{O}}(n^{\frac{1}{2}}) & \text{otherwise} \end{cases}$$

As the recursion depth is bounded by $O(\frac{1}{\epsilon})$ (a constant), we have $\mathcal{T}(n) = p(n)^{O(\frac{1}{\epsilon})}$ for some polynomial $p(n)$.

Proof of correctness: We now give a brief idea about the correctness of this algorithm. In the base case, we use similar technique as DFS just by alternatively exploring Red and Blue edges and thus this process gives us a path where two consecutive edges are of different colors. Otherwise, we also do a DFS like search by alternatively viewing Red and Blue edges and we do this search on the graph $H = (\overline{S'} \cup \{s, t\}, \overline{E'})$. By this process, we decide on presence of a path in H from s to t such that two consecutive edges are of different colors in G and the edge coming out from s is Red and the edge going in at t is Blue. This is enough as each path P in G must be broken down into the parts $P_1, P_2, \cdots, P_k$ and each P_i must be a sequence of edges that starts and ends at some vertices of $\overline{S'} \cup \{s, t\}$

and also alternates in color. We find each such P_i, just by considering each connected component of $G(V' \setminus \overline{S'})$ and repeating the same steps recursively. □

Due to [6], we know that the reachability problem in directed graphs reduces to RedBluePath in planar DAGs. For the class of graphs in which this reduction results the sub-quadratic increase in the number of vertices, we have an algorithm for reachability problem that takes sublinear space and polynomial time. As a special case of this we can state the following theorem.

Theorem 5. *Given a directed acyclic graph $G = (V, E)$, where $|E| = \widetilde{O}(n)$, with a drawing in a plane such that the number of edge crossings is $\widetilde{O}(n)$ and two vertices s and t, then for any constant $0 < \epsilon < \frac{1}{2}$, there is an algorithm that decides whether there is a path from s to t or not. This algorithm runs in polynomial time and uses $O(n^{\frac{1}{2}+\epsilon})$ space, where n is the number of vertices of G.*

Proof. We consider a reduction similar to the reduction from directed reachability problem to RedBluePath problem in planar DAG given in [6]. We do the following: (i) insert new vertices in between edges of G so that in the resulting graph each edge takes part in only one crossing and (ii) replace each crossing of the resulting graph with a *planarizing gadget* as in Fig. 1 and also replace each edge without any crossing with two edges as shown in Fig. 1. Denote the resulting graph as G_{planar} and the corresponding vertices of s and t as s' and t'. It is easy to see that there is a bijection between $s - t$ paths in G and $s' - t'$ red-blue paths in G_{planar} that starting with a Red edge and ending with a Blue edge.

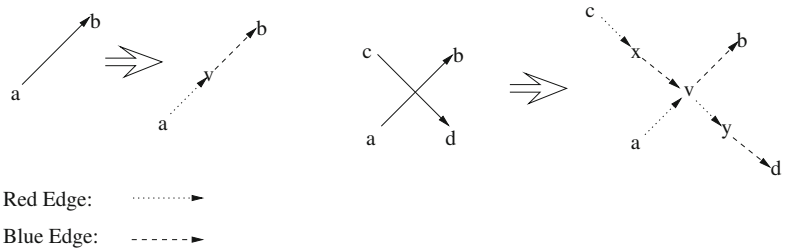

Fig. 1. Red-Blue Edge Gadget

If the drawing of the given graph G contains k edge crossings, then step (i) will introduce at most $2k$ many new vertices and say after this step the number of edges becomes m. Then step (ii) will introduce at most $(2m + 3k)$ many vertices. It is clear from the reduction itself that $m = \widetilde{O}(n)$ and thus the graph G_{planar} contains $\widetilde{O}(n)$ many vertices. Now by applying the algorithm `RedBluePathDetect` on G_{planar}, we get the desired result. □

A large class of graphs will satisfy the conditions specified in Theorem 5. We now explicitly give an example of one such class of graphs. Before that, we give

some definitions. *Crossing number* of a graph G, denoted as $cr(G)$, is the lowest number of edge crossings (or the crossing point of two edges) of a drawing of the graph G in a plane. A graph is said to be *k-planar* if it can be drawn on the plane in such a way that each edge has at most k crossing point (where it crosses a single edge). It is known from [9] that a k-planar graph with n vertices has at most $O(n\sqrt{k})$ many edges. Note that a k-planar graph has crossing number at most mk, where m is the number of edges. Now we can state the following corollary.

Corollary 1. *Given a directed acyclic graph, which is k-planar, where $k = O(\log^c n)$, for some constant c, with a drawing in a plane having minimum number of edge crossings and two vertices s and t, then for any constant $0 < \epsilon < \frac{1}{2}$, there is an algorithm that decides whether there is a path from s to t or not. This algorithm runs in polynomial time and uses $O(n^{\frac{1}{2}+\epsilon})$ space, where n is the number of vertices of the given graph.*

3.2 Deciding Even Path in Planar DAGs

Given directed graph G and two vertices s and t, EvenPath is the problem of deciding the presence of a (simple) directed path from s to t, that contains even number of edges. We can view this problem as a relaxation of RedBluePath problem as a path starting with Red edge and ending with Blue edge is always of even length. In this section, we establish a relation between EvenPath problem in planar DAG with detecting a odd length cycle in a directed planar graph with weight one (can also be viewed as an unweighted graph).

Lemma 1. *For directed planar graphs, for any constant $0 < \epsilon < \frac{1}{2}$, there is an algorithm that solves the problem of deciding the presence of odd length cycle in polynomial time and $O(n^{\frac{1}{2}+\epsilon})$ space, where n is the number of vertices of the given graph.*

The above lemma is true due to the fact that we can do BFS efficiently for undirected planar graph and it is enough to detect odd length cycle in each of the strong components of the undirected version of the given directed planar graph. For undirected graph, presence of odd length cycle can be detected using BFS algorithm and then put red and blue colors on the vertices such that vertices in the consecutive levels get the opposite colors. After coloring of vertices if there exists an monochromatic edge (edge where both vertices get the same color), then we can conclude that there is an odd length cycle in the graph otherwise there is no odd length cycle. But this is not the case for general directed graph. However, the following proposition will help us to detect odd length cycle in directed graph.

In the following proposition, we use $u \to v$ to denote a directed edge (u, v) and $x \xrightarrow{P} y$ to denote a directed path P from a vertex x to y.

Proposition 1. *A strongly connected directed graph contains an odd length cycle if and only if the underlying undirected graph contains an odd length cycle.*

Proof. The forward direction follows trivially. Now to prove the reverse direction, we will use the induction arguments on the length of the odd cycle in the undirected version of the graph. The base case is when the undirected version of the graph contains a 3-length cycle. If the undirected edges present in the undirected cycle also form directed cycle when we consider the corresponding edges in the directed graph, then there is nothing to prove. But if this is not the case, then the Fig. 2 will depict the possible scenarios. As the graph is strongly connected, so there must be a path P from t to s and if this path does not pass through the vertex x, then any one of the following two cycles $s \to t \xrightarrow{P} s$ or $s \to x \to t \xrightarrow{P} s$ must be of odd length. Now suppose P contains the vertex x and thus $P = P_1P_2$, where P_1 is the path from t to x and P_2 is the path from x to s. It is easy to see that all the three cycles $s \to t \xrightarrow{P} s$, $x \to t \xrightarrow{P_1} x$ and $s \to x \xrightarrow{P_2} s$ cannot be of even length.

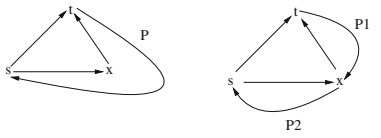

Fig. 2. For undirected cycle of length 3

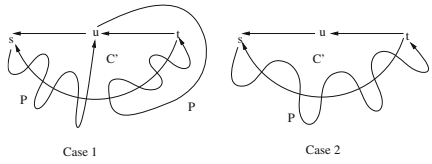

Fig. 3. For undirected cycle of length $(k+2)$

Now by induction hypothesis, assume that if the undirected version has a cycle of k-length (k odd), then there exists an odd length cycle in the original directed graph.

Now lets prove this induction hypothesis for any undirected cycle of length $(k + 2)$. Consider the corresponding edges in the directed graph and without loss of generality assume that this is not a directed cycle. As $(k + 2)$ is odd, so there must be one position at which two consecutive edges are in the same direction. Now contract these two edges in both directed and undirected version of the graph and consider the resulting k-length cycle in the undirected graph. So according to the induction hypothesis, there must be one odd length cycle C in the resulting directed graph. Now if C does not contain the vertex u (where we contract the two edges), then expanding the contracted edges will not destroy that cycle and we get our desired odd length cycle in the directed version of the graph. But if this is not the case, then consider C after expanding those two contracted edges($t \to u \to s$), say the resulting portion is C'. If C' is a cycle, then there is nothing more to do. But if not, then consider the path P from s to t (there must be such path as the graph is strongly connected). Now there will

be two possible cases: either P contains u or not. It is easy to see that for both the possible cases (case 1 and case 2 of Fig. 3 and in that figure every crossing of two paths denotes a vertex), all cycles generated by C' and P cannot be of even length. In case 1, if all the cycles generated by the paths $s \xrightarrow{P} u$ and $t \xrightarrow{C'} s$ and all the cycles generated by the paths $u \xrightarrow{P} t$ and $t \xrightarrow{C'} s$ are of even length, then as $t \xrightarrow{C'} s$ is of odd length, so the path $s \xrightarrow{P} u \xrightarrow{P} t$ must be of odd length. And then one of the following two cycles $s \xrightarrow{P} u \to s$ and $u \xrightarrow{P} t \to u$ is of odd length. Similarly in case 2, if all the cycles generated by $s \xrightarrow{P} t$ and $t \xrightarrow{C'} s$ are of odd length, then the path $s \xrightarrow{P} t$ is of odd length and so the cycle $s \xrightarrow{P} t \to u \to s$ is of odd length.

Proof (of Lemma 1). In a directed planar graph, any cycle cannot be part of two different strong component, so checking presence of odd cycle is same as checking presence of odd cycle in each of its strong components. Constructing strong components of a directed planar graph can be done by polynomial many times execution of `DirectedPlanarReach` algorithm (See Theorem 4), as a strong component will contain vertices x, y if and only if `DirectedPlanarReach` (G, x, y, n) and `DirectedPlanarReach` (G, y, x, n) both return "yes". And thus strong component construction step will take $\widetilde{O}(n^{\frac{1}{2}+\epsilon})$ space and polynomial time. After constructing strong components, it is enough to check presence of odd cycle in the underlying undirected graph (according to Proposition 1). So now on, without loss of generality, we can assume that the given graph G is strongly connected. Now execute `OddCycleUndirectedPlanar` $(\hat{G}, s, n)$ (Algorithm 4) after setting the color of s (any arbitrary vertex) to red. Here we adopt the well known technique used to find the presence of odd length cycle in a graph using BFS and coloring of vertices. In Algorithm 4, instead of storing color values for all the vertices, we only stores color values for the vertices present in the separator (Line 11) and we do the coloring recursively by considering the smaller connected components (Line 10). The algorithm is formally defined in Algorithm 4.

By doing the similar type of analysis as that of `RedBluePathDetect`, it can be shown that `OddCycleUndirectedPlanar` will take $O(n^{\frac{1}{2}+\epsilon})$ space and polynomial time and so over all space complexity of detecting odd length cycle in directed planar graph is $O(n^{\frac{1}{2}+\epsilon})$ and time complexity is polynomial in n.

Now we argue on the correctness of the algorithm `OddCycleUndirectedPlanar`. This algorithm will return "yes" in two cases. First case when there is a odd length cycle completely inside a small region ($n' \leq n^{\frac{1}{2}}$) and so there is nothing to prove for this case as it is an well known application of BFS algorithm. Now in the second case, a vertex v in the separator family will get two conflicting colors means that there exists at least one vertex u in the separator family such that there are two vertex disjoint odd as well as even length path from u to v and as a result, both of these paths together will form an odd length cycle. □

Now we are ready to prove the main theorem of this subsection.

Input : $G' = (V', E'), s', n$, where G' is an undirected graph
Output : "Yes" if there is an odd length cycle

if $n' \leq n^{\frac{1}{2}}$ **then**
  Run `BFS`(G' , s') and color the vertices with red and blue such that vertices in the alternate layer get the different color starting with a vertex that is already colored;
  if *there is a conflict between stored color of a vertex and the new color of that vertex or there is an edge between same colored vertices* **then**
    return "yes";
  end
else
  `/* let` $r' = n'^{(1-\epsilon)}$ `*/`
  Run `PlanarSeparatorFamily` on $\hat{G'}$ to compute r'-separator family $\overline{S'}$;
  Set $S' := \overline{S'} \cup \{s'\}$;
  for *every* $x \in V'$ **do**
    `/*` $V_x =$ `the set of vertices of` $\hat{H}$`'s connected component containing` x`, where` $H = G[V' \setminus S']$ `*/`
    Run `OddCycleUndirectedPlanar`($G[V_x \cup S'], s', n$);
    Store color of the vertices of S' in an array of size $|S'|$;
  end
end

Algorithm 4. Algorithm `OddCycleUndirectedPlanar`: Checking Presence of Odd Cycle in an Undirected Planar Graph

Proof (of Theorem 2). Given a planar DAG G and two vertices s and t, first report a path from s to t, say P, which can easily be done by polynomially many invocation of the algorithm `DirectedPlanarReach` mentioned in Theorem 4 and thus requires polynomial time and $O(n^{\frac{1}{2}+\epsilon})$ space. If the path P is not of even length, then construct a directed graph G' which has the same vertices and edges as G except the edges in path P, instead we do the following: if there is an edge (u, v) in P, then we add an edge (v, u) in G'. Now we can observe that the new graph G' is a directed planar graph.

Claim. G has an even length path if and only if G' has an odd length cycle.

Proof. Suppose G' has an odd length cycle, then that cycle must contains the reverse edges of P in G. Denote the reverse of the path P by P_{rev}. Now lets assume that the odd cycle C' contains a portion of P_{rev} (See Fig. 4). Assume that the cycle C' enters into P_{rev} at x (can be t) and leaves P_{rev} at y (can be s). Then in the original graph G, the path $s \xrightarrow{P} y \xrightarrow{C'} x \xrightarrow{P} t$ is of even length.
Now for the converse, lets assume that there exists an even length path P_1 from s to t in G. Both the paths P and P_1 may or may not share some edges and without loss of generality we can assume that they share some edges (See Fig. 5). Now if we consider all the cycles formed by P_{rev} and portions of P_1 in G', then it is easy to see that all the cycles cannot be of even length until length of

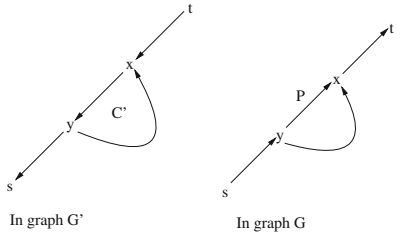

Fig. 4. When G' contains an odd length cycle

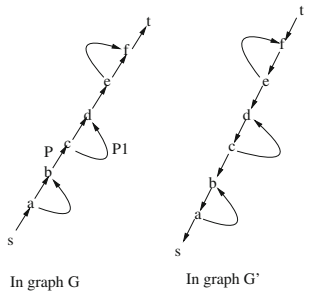

Fig. 5. When G contains an even length $s-t$ path

P and P_1 both are of same parity (either both odd or both even), but this is not the case.

Now we can check the presence of odd length cycle in the graph G' in polynomial time and $O(n^{\frac{1}{2}+\epsilon})$ space (by Lemma 1). □

Acknowledgement. The first author would like to thank Surender Baswana for some helpful discussions and comments. We also thank the anonymous reviewers for their helpful comments and suggestions.

References

1. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. Syst. Sci. 4, 177–192 (1970)
2. Wigderson, A.: The complexity of graph connectivity. In: Mathematical Foundations of Computer Science, pp. 112–132 (1992)
3. Barnes, G., Buss, J.F., Ruzzo, W.L., Schieber, B.: A sublinear space, polynomial time algorithm for directed s-t connectivity. In: Proceedings of the Seventh Annual Structure in Complexity Theory Conference, pp. 27–33 (1992)
4. Imai, T., Nakagawa, K., Pavan, A., Vinodchandran, N.V., Watanabe, O.: An $O(n^{1/2+\epsilon})$-Space and Polynomial-Time Algorithm for Directed Planar Reachability. In: 2013 IEEE Conference on Computational Complexity (CCC), pp. 277–286 (2013)
5. Chakraborty, D., Pavan, A., Tewari, R., Vinodchandran, N.V., Yang, L.: New time-space upperbounds for directed reachability in high-genus and h-minor-free graphs. Electronic Colloquium on Computational Complexity (ECCC) 21, 35 (2014)
6. Kulkarni, R.: On the power of isolation in planar graphs. TOCT 3(1), 2 (2011)
7. LaPaugh, A.S., Papadimitriou, C.H.: The even-path problem for graphs and digraphs. Networks 14(4), 507–513 (1984)
8. Nedev, Z.P.: Finding an Even Simple Path in a Directed Planar Graph. SIAM J. Comput. 29, 685–695 (1999)
9. Pach, J., Tóth, G.: Graphs drawn with few crossings per edge. Combinatorica 17(3), 427–439 (1997)