



Simultaneous Space-Time Complexities of Variants of Reachability

Supervisor – Dr. Vimal Raj Sharma

Vaibhav Agarwal (B21CS077)

Alok Kumar (B21CS006)

Contents


- ❖ An $O(n^{1/2+\epsilon})$ -Space and Polynomial-Time Algorithm for Directed Planar Reachability
- ❖ A Sublinear Space, Polynomial Time Algorithm for Directed s - t Connectivity
- ❖ Simultaneous Time-Space Upper Bounds for Red-Blue Path Problem in Planar DAGs

An $O(n^{1/2+\epsilon})$ -Space and Polynomial-Time Algorithm for Directed Planar Reachability

- The graph reachability problem is to decide, for a given graph G and its two vertices s and t , whether there is a path from s to t in G .
- This paper shows that the reachability problem over directed planar graphs can be solved simultaneously in polynomial time and approximately $O(\sqrt{n})$ space.

This paper proves the following theorem by giving a polynomial time and $O(n^{1/2+\epsilon})$ space algorithm for directed planar reachability.

- **Theorem 1.** For any constant $0 < \epsilon < 1/2$, there is an algorithm 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 .



Separator Definition. For any undirected graph G and for any constant ρ , $0 < \rho < 1$, a subset of vertices S is called a ρ -separator if (i) removal of S disconnects G into two subgraphs A and B , and (ii) the number of vertices of any component is at most ρn . The size of a separator is the number of vertices in the separator.

Separator Algorithm. There is an algorithm that takes an undirected planar graph G with n vertices as input and outputs a $(8/9)$ - separator of G . This algorithm runs in polynomial time and uses $\tilde{O}(\sqrt{n})$ space (i.e. $O(\sqrt{n}(\log n)^{O(1)})$ space).

First we define an algorithm SepFamily that uses separator algorithm iteratively to compute a separator family that splits the graph into sublinear size components.

SepFamily. For any $\epsilon > 0$, there is an algorithm SepFamily that takes a planar graph as input and outputs an $n^{1-\epsilon}$ - separator family of size $O(n^{1/2+\epsilon/2})$ in polynomial time and $\tilde{O}(n^{1/2+\epsilon/2})$ space.

Our reachability algorithm uses the above SepFamily algorithm.

❖ *Algorithm for Directed Planar Reachability*

1. **PlanarReach**($\hat{G}, \hat{s}, \hat{t}, n$)
(let \hat{n} be the number of vertices of \hat{G})
2. **If** $\hat{n} \leq n^{1/2}$
3. **then** **BFS**($\hat{G}, \hat{s}, \hat{t}$)
4. **Else** (let $\hat{r} = \hat{n}^{1-\epsilon}$)
5. Run **SepFamily** to compute \hat{r} -separator family \bar{S}
6. Run **ImplicitBFS**($(\bar{S} \cup \{\hat{s}, \hat{t}\}, \bar{E}), \hat{s}, \hat{t}$)
 // **ImplicitBFS** executes in the same way as **BFS**
 // except for the case “ $(a, b) \in \bar{E}$?” is queried,
 // i.e., it is asked whether $G(V' \cup \bar{S})$ has an
 // edge (a, b) ? In this case, the query is answered
 // by the following process 6.1 ~ 6.5.
- 6.1. **For** every $x \in V$
 // V_x = the set of vertices of $\underline{G}(V \setminus \bar{S})$'s
 // connected component containing x .
- 6.2. **If** **PlanarReach**($G(V_x \cup \bar{S}), a, b, n$) is True
- 6.3. **then** Return True for the query
- 6.4. **End_For**
- 6.5. Return False for the query

A Sublinear Space, Polynomial Time Algorithm for Directed s-t Connectivity

- The directed s-t connectivity (**STCON**) is the problem of detecting whether there is a path from a distinguished vertex s to a distinguished vertex t in a directed graph.
- The main result of this paper is a new deterministic algorithm for directed s-t connectivity that achieves polynomial time and sublinear space simultaneously. The algorithm can use as little as $n/2^{\Theta(\sqrt{\log n})}$ space while running in polynomial time.
- The algorithm to solve STCON is constructed from 2 algorithms. These 2 algorithms can be combined efficiently to yield the desired result.

Modified Breadth-First Search algorithm: It performs modified bfs of the graph.

Short Paths algorithm: It finds paths of fixed length between 2 vertices in the graph.

Modified breadth-first search algorithm

- The algorithm constructs the partial BFS tree instead of the complete one to limit the memory space.
- Our modified tree contains at most n/λ vertices. We can do this by only storing the vertices in every λ^{th} level of the tree.
- The algorithm constructs this partial tree one level at a time. It begins with level 0, which consists of s only, and generates levels $j, j + \lambda, j + 2\lambda, \dots, j + \lambda \cdot \lfloor n/\lambda \rfloor$.
- Given a set, S , of vertices, we have to find all vertices within distance λ of S and within distance $\lambda - 1$ of S . This can be used to generate the levels of the partial tree.
- To get V_{i+1} (vertices in the $(i+1)^{\text{th}}$ level), we add to V_i (vertices in the i^{th} level) all vertices that are within distance λ but not $\lambda - 1$ of V_i .

```

Algorithm Bfs (integer:  $\lambda$ );
    {remember every  $\lambda$ th level of the breadth-first search tree}
    for  $j = 0$  to  $\lambda - 1$  do begin
        {first level to remember, apart from level 0}
         $S = \{s\}$ .
        for all vertices,  $v$  do begin
            {Find vertices on the first level}
            if  $v$  within distance  $j$  of  $s$  and
                $v$  not within distance  $j - 1$  of  $s$  then
                if  $|S| > n/\lambda$  then try next  $j$ .    {Don't store more than  $n/\lambda$  vertices, + vertex  $s$ }
                else add  $v$  to  $S$ .
            end;
        for  $i = 1$  to  $\lfloor n/\lambda \rfloor$  do begin
             $S' = \emptyset$ .
            for all vertices,  $v$  do begin
                {Find vertices on the next level. *}
                if  $v$  within distance  $\lambda$  of some vertex in  $S$  and
                    $v$  not within distance  $\lambda - 1$  of any vertex in  $S$  then
                    if  $|S| + |S'| > n/\lambda$  then try next  $j$ .
                    else add  $v$  to  $S'$ .
                end;
             $S = S \cup S'$ .
            end;
        if  $t$  within distance  $\lambda$  of a vertex in  $S$  then return (CONNECTED);
        else return (NOT CONNECTED);
    end;
end Bfs.
    
```

Short Paths algorithm

- Suppose we divide the vertices into k sets, according to their vertex number mod k . Then, every path of length L ($L = f(n)$) can be mapped to an $(L+1)$ -digit number in base k , where digit i has value j if and only if the i th vertex in the path is in set j . Conversely, each such number defines a set of possible paths of length L .
- Given this mapping, the algorithm is straightforward: generate all possible $(L+1)$ -digit k -ary numbers, and check for each number whether there is a path in the graph that matches it.
- For a given k -ary number, the algorithm uses approximately $2n/k$ space to test for the existence of a matching path in the graph.

So, through this algorithm, we can find a path from s to t of length exactly L by testing the $(L+1)^{\text{th}}$ -digit number.

→ Now, the above 2 algorithms (**modified BFS** and **short paths**) can be combined in an efficient way to give our required algorithm for STCON that uses sublinear space and polynomial time.

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

In this paper, we consider the RedBluePath problem, which states that given a directed 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 such that the first edge is red and the last edge is blue.

The RedBluePath problem is NL-complete even when restricted to planar DAGs.

A natural relaxation of RedBluePath problem is EvenPath problem.

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.

For DAGs, EvenPath problem is NL-complete but for planar DAGs, EvenPath problem is in UL.

The reachability problem in directed graphs reduces to RedBluePath problem in planar DAGs.

The main contribution of this paper is to exhibit a polynomial time and $O(n^{1/2+\epsilon})$ space algorithm (for any $\epsilon > 0$) for RedBluePath problem and EvenPath problem in planar DAG.

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

Proof:

```

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

```

```


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

```

```

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( $\overline{G'}, G', v, t', temp + 1(mod\ 2), final$ )
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
11   /*  $V_a$  be the set of vertices of  $G'$  */
12   /*  $V_a =$  the set of vertices of  $\hat{H}$ 's connected component
      containing  $a$ , where  $H = G[V \setminus \overline{V'}]$  */
13   if RedBluePathDetect( $G[V_a \cup \overline{V'}], u, v, n, init, temp$ ) is true then
14     Return true for the query;
15   end
16 end
/* End of the query procedure */

```



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

This paper establishes a relation between EvenPath problem in planar DAGs and the problem of finding odd length cycle in a directed planar graph.

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

Lemma 1 can be proved with the help of following proposition.

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.

This paper proves Theorem 2 by using lemma 1 above.



Thank you