Foundations of Software Technology and Theoretical Computer Science (Bangalore) 2008.
Editors: R. Hariharan, M. Mukund, V. Vinay; pp 256-267

STCON in Directed Unique-Path Graphs

Sampath Kannan[*], **Sanjeev Khanna**[†], **Sudeepa Roy**[‡]
University of Pennsylvania
Philadelphia, PA, USA
{kannan,sanjeev,sudeepa}@cis.upenn.edu

ABSTRACT. We study the problem of space-efficient polynomial-time algorithms for *directed st-connectivity* (STCON). Given a directed graph G, and a pair of vertices s, t, the STCON problem is to decide if there exists a path from s to t in G. For general graphs, the best polynomial-time algorithm for STCON uses space that is only slightly sublinear. However, for special classes of directed graphs, polynomial-time poly-logarithmic-space algorithms are known for STCON. In this paper, we continue this thread of research and study a class of graphs called *unique-path graphs with respect to source* s, where there is at most one simple path from s to any vertex in the graph. For these graphs, we give a polynomial-time algorithm that uses $\tilde{O}(n^{\varepsilon})$ space for any constant $\varepsilon \in (0,1]$. We also give a polynomial-time, $\tilde{O}(n^{\varepsilon})$-space algorithm to *recognize* unique-path graphs. Unique-path graphs are related to configuration graphs of unambiguous log-space computations, but they can have some directed cycles. Our results may be viewed along the continuum of sublinear-space polynomial-time algorithms for STCON in different classes of directed graphs - from slightly sublinear-space algorithms for general graphs to $O(\log n)$ space algorithms for trees.

1 Introduction

We study the *directed st-connectivity (STCON)* problem, where given a directed graph G, a source vertex s and a terminal vertex t, we are interested in finding whether there is a path from s to t in G. The STCON problem can be solved in polynomial time using standard search algorithms (for eg. Depth First Search (DFS) or Breadth First Search (BFS)). These algorithms run in $O(m+n)$ time and use $O(n \log n)$ space on a graph with n vertices and m edges. Improving the space complexity of STCON is a well-studied and fundamental problem. The best known deterministic upper bound is given by Savitch's theorem [18], which solves STCON in $O(\log^2 n)$ space. On the other hand, STCON is known to be NL-complete [13]; i.e. giving an $O(\log n)$ space algorithm will imply $L = NL$. A comprehensive survey on the complexity of STCON can be found in [19].

An interesting related question is the time-space trade-off involved in solving STCON [8]. Savitch's theorem uses $O(\log^2 n)$ space, but takes super-polynomial ($n^{O(\log n)}$) time. DFS or BFS takes linear time but its standard implementation requires $O(n \log n)$ space. The only algorithm known till date that breaks the linear space barrier but takes polynomial time is due to Barnes *et al* [6]. This algorithm uses $n/2^{\Theta(\sqrt{\log n})}$ space to solve STCON in any directed graph.

[*]Supported by NSF Award CT-ISG 0716172.
[†]Supported in part by a Guggenheim Fellowship, an IBM Faculty Award, and by NSF Award CCF-0635084.
[‡]Supported by NSF Award IIS-0803524.

On the other hand, the space complexity of the undirected counterpart of STCON, namely USTCON, has been recently resolved by Reingold [15], who showed that USTCON $\in L$. USTCON can also be solved in randomized $O(\log n)$ space and $O(mn)$ time using random walks [1].

STCON has been studied in more restricted models of computation than the Turing machine model. For example in the *JAG (Jumping Automaton for Graphs)* model proposed by Cook and Rackoff, it has been shown that STCON has a lower bound of $\Omega(\log^2 n / \log \log n)$ on space complexity [10]. The same lower bound has also been shown by Berman and Simon [7] for the Randomized JAG model. Poon further defined a stronger *Node Named JAG (NNJAG)* model [14] and showed a time-space lower bound of $T = 2^{\Omega(\frac{\log^2(n \log n/S)}{\log \log n})} \times \sqrt{nS/\log n}$ on both the JAG and the NNJAG model [12] (where T denotes the time and S denotes the space), underscoring the difficulty in designing polynomial-time sublinear-space algorithms for STCON.

One important complexity class in the study of STCON and reachability problems is *Unambiguous Log-Space (UL or USPACE($\log n$))* [5]. This is a subclass of NL and characterizes the class of problems accepted by logarithmic-space-bounded non-deterministic Turing Machines with at most one accepting computation path for each input. Though this appears to be a strong restriction on the computation power of non-deterministic log-space machines, UL/poly has been shown to be identical to NL/poly in [17]. Further subclasses of UL with stronger requirements in terms of uniqueness of the computation path between two configurations have been defined [9]. The complexity class *Reach Unambiguous Log-Space (RUSPACE($\log n$))* requires any two configurations reachable from the start configuration to have a unique computation path in between them and in *Strong Unambiguous Log-Space (StUSPACE($\log n$))* any two configurations should have at most one path between them. Clearly StUSPACE($\log n$) $\subseteq$ RUSPACE($\log n$) $\subseteq$ UL. In RUSPACE($\log n$) the set of vertices reachable from the *source* vertex forms a tree whereas in StUSPACE($\log n$) the set of vertices reachable from *any* vertex forms a tree. Configuration graphs for StUSPACE($\log n$) have also been described as *Mangroves* in [4].

Though both the space complexity and the time-space tradeoff for STCON have not been resolved till date for general directed graphs, these problems have been studied extensively on interesting subclasses of directed graphs. Given an oracle to access the set of incoming edges and outgoing edges for a node in the graph, the STCON problem can be easily solved in polynomial time using $O(\log n)$ space on a tree. Allender *et al* have given a polynomial-time $O(\log^2 n / \log \log n)$ space algorithm to solve the STCON problem on StUSPACE($\log n$) that also works for RUSPACE($\log n$) [4]. On planar DAGs with single source STCON has been shown to be solvable using $O(\log n)$ space [3]. But none of these graph families allow the presence of cycles. A recent survey of Allender [2] highlights the results on the complexity of reachability in UL and its subclasses and on other special subclasses of directed graphs. Also Reingold's technique has been generalized in [16] to solve STCON in $O(\log n)$ space for *regular directed graphs*, where there is a value d such that the in-degrees and out-degrees of all vertices are d.

In this paper we define a subclass of directed graphs that we call *unique-path graphs with respect to source vertex s*. These graphs are defined by the existence of at most one simple path from s to any vertex in the graph. We will show later that this class of graphs is characterized by the absence of forward or cross edges with respect to any DFS-tree rooted at a vertex reachable from the source vertex. But some back edges may be present, i.e., we allow the presence of some cycles in these graphs.

Configuration graphs for UL and its subclasses are closely related to unique-path graphs. Unique-path graphs strictly contain trees and configuration graphs for StUSPACE($\log n$) and RUSPACE($\log n$), since we allow some cycles. But, a configuration graph for UL is not necessarily a unique-path graph, and vice-versa. Figure 1 shows examples of unique-path graphs and configuration graphs of different subclasses of UL.

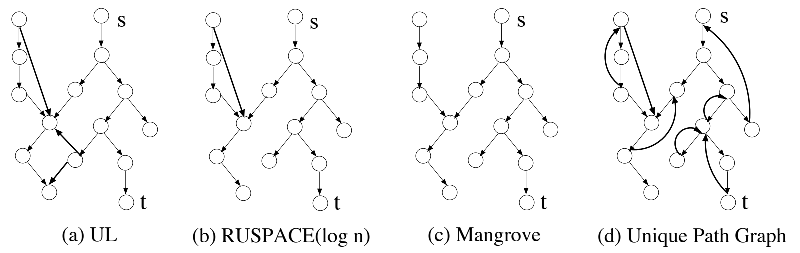

Figure 1: Examples of unique-path graphs and configuration graphs of UL and its subclasses

Our Results. As noted above, upper bounds on the space complexity of polynomial-time algorithms for STCON ranges from $O(\log n)$ (on trees), $O(\log^2 n/\log\log n)$ (on mangroves) to $n/2^{\Theta(\sqrt{\log n})}$ (on general directed graphs). In this paper, we show that for any $\varepsilon \in (0,1]$, the STCON problem can be solved in unique-path graphs in $n^{O(\frac{1}{\varepsilon})}$ time using $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$ space [§]; this gives a polynomial time algorithm that uses $\tilde{O}(n^\varepsilon)$ space for any constant ε. We also show that we can *recognize* unique-path graphs in $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$ space and $n^{O(\frac{1}{\varepsilon})}$ time.

Our algorithm uses a sublinear-space implementation of DFS in unique-path graphs. The standard implementation of DFS uses linear space for two purposes: (i) to maintain a stack for backtracking from a vertex v after exploring all vertices reachable from it, and (ii) to keep track of all vertices already visited and avoid rediscovering them. We show that, in unique-path graphs, these purposes can be served by maintaining a sublinear-space data structure which we call *landmark vertices*. We first give an $\tilde{O}(\sqrt{n})$-space polynomial-time algorithm for STCON in unique-path graphs. Extending our techniques further, we obtain an algorithm which improves the space requirement to $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$.

Organization. The rest of the paper is organized as follows. In Section 2 we define a unique-path graph and discuss some useful properties of unique-path graphs. In Section 3 we give an $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$-space $n^{O(\frac{1}{\varepsilon})}$-time algorithm for any $\varepsilon \in (0,1]$ to solve STCON in unique-path graphs. In Section 4 we show how to decide if an input directed graph is a unique-path graph in $\tilde{O}(\frac{n^\varepsilon}{\varepsilon})$ space and $n^{O(\frac{1}{\varepsilon})}$ time. Section 5 contains conclusions and some directions for future work.

[§] $\tilde{O}(f(n))$ denotes $O(f(n)\log^k n)$, for some constant k.

2 Preliminaries

Given a directed graph G, we will use $V(G)$ and $E(G)$ to denote the set of vertices and edges, respectively, in G. We assume that there are no self loops or parallel edges in the graph. A path where no intermediate vertices is repeated is called a *simple path*; a *simple cycle* is defined similarly. Two simple paths p_1, p_2 are called *distinct* if they differ in at least one edge. Next we define a *unique-path graph*.

DEFINITION 1. *A directed graph G is a* unique-path graph *with respect to a source vertex s if there is at most one simple path from s to any vertex $v \in V(G)$.*

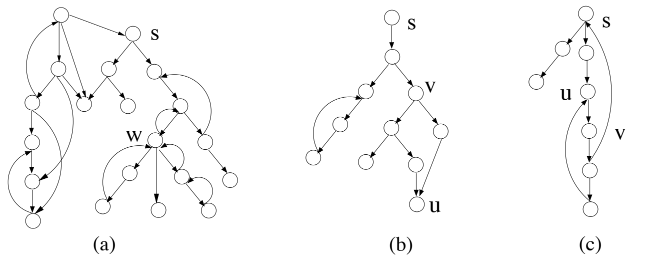

Figure 2: Examples of unique-path graphs ((a)) and non-unique-path graphs ((b) and (c))

It is easy to see that a directed graph G is a unique-path graph with source vertex s iff there is at most one simple path between any two distinct vertices $u, v \in V(G)$, where both u and v are reachable from s. If a directed graph is not a unique-path graph, we call it a *non-unique-path graph*. Examples of unique-path and non-unique-path graphs are shown in Figure 2. The graph in Figure 2(a) is a unique-path graph with source vertex s, but the graphs in Figure 2(b) and (c) are non-unique-path graphs - in both cases there are two distinct simple paths from s to u. The definition of unique-path graphs does not put any restriction on the vertices which are not reachable from the source vertex s; they can have arbitrary number of simple paths between them. Even there can be multiple simple paths from a vertex u to a vertex v where v is reachable from s but u is not. Also note that while there is at most one simple path between any pair of *distinct* vertices reachable from s, a vertex reachable from s (such as w in Figure 2(a)) can lie on many different simple cycles.

For any vertex x in a graph G, we denote by $N^-(x)$ (resp. $N^+(x)$) the set of vertices that have an out-going edge to (in-coming edge from) x in G. We assume that the input graph G is represented in an adjacency-list format, where for each vertex $x \in V(G)$, $N^+(x)$ and $N^-(x)$ are specified as lists. Given $u, v \in N^+(x)$ (or $N^-(x)$) u is called a *successor* of v if u immediately follows v in the list. We assume access to the incoming and the outgoing edges of a node v and therefore the neighbors of v via queries to an *oracle* that answers as follows: given vertices v and w, the oracle can answer if $w \in N^+(v)$ (or in $N^-(v)$), i.e., we can check if (v, w) (or (w, v)) $\in E(G)$. Also we can query the oracle to return the successor (if any) of w in $N^+(v)$ or in $N^-(v)$.

2.1 Properties of Unique-Path Graphs

The algorithm we present to solve STCON in unique-path graphs relies on depth first search (DFS) from the source vertex s. We therefore begin by making a few observations about DFS in unique-path graphs. DFS from a vertex $v \in V(G)$ generates a tree called the *DFS-tree* with v as the *root* of the tree. The edges used in the tree are called *tree edges*. Apart from tree edges, DFS on general directed graphs yield three other types of edges: *back edges, forward edges and cross edges* (see, for instance [11]). The *parent* of a vertex v is the vertex $u \in N^-(v)$ such that (u, v) is a tree edge and will be denoted by $\pi(v)$. Lemma 2 states a necessary and sufficient condition for a directed graph G to be a unique-path graph with respect to a vertex s and Lemma 3 describes the structure of back edges in a unique-path graph. The proofs are easy and are omitted due to space constraint.

LEMMA 2. *A directed graph G is a unique-path graph with respect to $s \in V(G)$ iff DFS invoked from any vertex reachable from s does not produce any forward or cross edges.*

For a back edge (u, v) in a DFS-tree, let $\text{SPAN}(u, v)$ denote the set of vertices on the path in the DFS-tree from v to u including v and u.

LEMMA 3. *Let G be a unique-path graph. Let $(u, v), (x, y) \in E(G)$ be back edges in the DFS-tree with w as the root, where w is reachable from s. Then $|\text{SPAN}(u, v) \cap \text{SPAN}(x, y)| \leq 1$.*

3 Algorithm for STCON in Unique-path Graphs

We assume that G is a directed unique-path graph with respect to source s in this section. We will solve STCON in unique-path graphs by implementing DFS from s in polynomial time using $O(n^\varepsilon)$ space. A typical implementation of DFS relies on remembering the set of vertices that have already been visited (to avoid *rediscovering* previously visited vertices), and remembering the current exploration path for *backtracking* from a vertex v after all vertices reachable from v have been visited (using a stack). Both these tasks can be accomplished using linear space. We show that, for unique-path graphs, these steps can be implemented in polynomial time using sublinear space by maintaining some sparse auxiliary information.

Our final aim is to design an $O(n^\varepsilon)$ space polynomial-time algorithm to implement STCON in unique-path graphs for a constant $\varepsilon \in (0, 1]$ [¶]. In Section 3.1 we give an $O(\sqrt{n})$ space polynomial-time algorithm to present our techniques. In Section 3.2 we will use our techniques recursively to get an $O(n^\varepsilon)$-space polynomial-time algorithm. We will assume that the oracle takes one unit of time to answer any query, though we get a polynomial-time algorithm as long as the time taken by the oracle is bounded by a polynomial. We will refer to the last vertex discovered by the DFS with an unfinished DFS call as the *current vertex*, and the path using the tree edges from s to the current vertex as the *active path*.

3.1 An $O(\sqrt{n})$-Space Algorithm

We prove the following theorem in this section.

[¶]From now on, "$O(f(n))$ space" will refer to the space needed to store $O(f(n))$ words; the bit complexity will be $O(f(n) \log n) = \tilde{O}(f(n))$.

Algorithm 1 An $O(\sqrt{n})$-space, polynomial-time algorithm for STCON on a unique-path graph G with source vertex s and terminal vertex t

```
1: CURRENT:
2: – Suppose the control of the DFS is at the current vertex $x$ (initially $x = s$).
3: – Either DFS has backtracked to $x$ from some vertex $v \in N^+(x)$, or $x$ is a newly discovered vertex.
4: if DFS has backtracked to $x$ from $v$ then
5:     – Ask the oracle to return the successor of $v$ in $N^+(x)$.
6: else {$x$ is a newly discovered vertex}
7:     – Ask the oracle to return the first vertex in $N^+(x)$.
8: end if
9: NEXT:
10: if the oracle returns that there are no more vertices in $N^+(x)$ then {either the DFS has backtracked from the last child of $x$ in $N^+(x)$ or $N^+(x)$ is empty)}
11:     if $x$ is same as the source vertex $s$ then {the search from $s$ is complete}
12:         – Exit with the answer 'there are no paths from $s$ to $t$ in $G$'.
13:     end if
14:     – Perform the backtrack step for $x$ to reach $u = \pi(x)$.
15:     – Set $x = u$, pass the control to (new) $x$ and jump to Step CURRENT.
16: else
17:     – The oracle returns $y$ as the next vertex in $N^+(x)$.
18:     – Perform the discovery step for the edge $(x, y)$.
19:     if $(x, y)$ is a back edge then {$y$ has been visited before,}
20:         – Ask the oracle to return the successor of $y$ in $N^+(x)$ and jump to Step NEXT.
21:     else {$y$ is a newly discovered vertex}
22:         if $y$ is same as the terminal vertex $t$ then
23:             – Exit with the answer 'there is a path from $s$ to $t$ in $G$'.
24:         end if
25:         – Set $x = y$, pass the control to (new) $x$ and jump to Step CURRENT.
26:     end if
27: end if
```

THEOREM 4. *STCON is solvable in $O(mn + m^2\sqrt{n})$ time with $O(\sqrt{n})$ space in unique-path graphs.*

Overview of the Algorithm Algorithm 1 describes how STCON in unique-path graphs can be implemented in $O(\sqrt{n})$ space and polynomial time. It relies on a sublinear-space implementation of two key subroutines. The first subroutine is to *backtrack from a vertex x*, i.e., to return the control to the parent $\pi(x)$ once the DFS finishes at x. The second subroutine is the *discovery step for an edge (x, y)*, which is called from a current vertex x to determine if an edge (x, y) being considered by the DFS is a back edge. Now in order to complete the description of the algorithm, it suffices to describe how we implement the backtrack and the discovery steps. Note that Algorithm 1 and later the procedures for the backtrack and the discovery steps always start the search on G from a vertex reachable from s. Thus we only need the unique-path property of the vertices reachable from the source vertex s and do not have any restriction on the rest. First we introduce the notion of L-bounded DFS, which is used as a subroutine in the procedures for the backtrack and the discovery steps.

L-bounded DFS

DEFINITION 5. *A DFS search is called L-*bounded *if it backtracks whenever the length of the active path exceeds L.*

For a unique-path graph, if we store the entire active path, the backtrack and the discovery steps can be easily implemented. The next lemma follows from the above observation; we omit the proof due to space constraint.

LEMMA 6. *Given a unique-path graph G with respect to source s, a vertex $v \in V(G)$ reachable from s and and an integer L, an L-bounded DFS from v can be implemented in $O(L)$ space and $O(n + mL)$ time. Moreover, it visits every vertex within distance L from v, and does not visit any vertex at distance greater than L from v.*

We note that an $O(\sqrt{n})$ space, polynomial-time algorithm for STCON in unique-path graphs can be obtained from the above lemma with the approach of [6]. But there is no obvious way of improving the space complexity beyond $O(\sqrt{n})$ using this approach. We present here another approach for solving STCON in unique-path graphs in $O(\sqrt{n})$ space. This will be the starting point to obtain an algorithm that reduces the space requirement to $O(n^{\varepsilon})$.

Next we describe the implementation of the backtrack and discovery steps. The main idea in implementing these steps is maintaining *landmark vertices* which are a few evenly spaced vertices on the active path from s to the current vertex x. The landmark vertices will be denoted by z_i, $i = 0, 1, \cdots$, where the landmark vertex z_i is at distance $i\lfloor\sqrt{n}\rfloor$ from s along the current active path ($s = z_0$); i is called the *index* of the landmark vertex z_i. We will consider the current vertex x as an additional landmark vertex z_p, where $z_0, z_1, \cdots, z_{p-1}$ is the the set of landmark vertices maintained along the active path to x. Since it is easy to maintain the length of the active path from s to the current vertex x in Algorithm 1, the landmark vertices can be maintained by a simple modification of the algorithm. The space needed to maintain the landmark vertices is $O(\sqrt{n})$, because the number of landmark vertices is $O(\sqrt{n})$. As in standard DFS, Algorithm 1 performs $O(n)$ backtrack and $O(m)$ discovery steps; thus to implement the whole algorithm in $O(\sqrt{n})$ space and polynomial time it suffices to show that the backtrack and the discovery steps can be implemented in $O(\sqrt{n})$ space and polynomial time.

Backtrack Step

Let x be the current vertex and let $v_1, v_2, ..., v_q$ be the vertices in $N^-(x)$. Suppose $v_i = \pi(x)$ in the DFS-tree. Since G is a unique-path graph with respect to source s, the unique simple path from s to x is through the edge (v_i, x). Recall that if $z_0, \cdots, z_p$ are the landmark vertices then current vertex $x = z_p$ and z_{p-1} is the previous landmark vertex.

Procedure 2 Procedure to implement the backtrack step from the current vertex x

```
1: – Let $v_1, v_2, ..., v_q$ be the vertices in $N^-(x)$.
2: for each $v_j \in N^-(x)$ do
3:     – Perform a $\sqrt{n}$-bounded DFS from $z_{p-1}$ in the graph $G - (v_j, x)$.
4:     if $z_p = x$ is not reached then
5:         – Return $v_j$ as $\pi(x)$.
6:     end if
7: end for
```

LEMMA 7. *In the graph $G - (v_i, x)$ the current vertex $z_p = x$ is not discovered by a $\sqrt{n}$-bounded DFS from z_{p-1} iff v_i is the parent of x in the original DFS-tree.*

PROOF. (if) Assume $v_i = \pi(x)$ and there is a path from z_{p-1} to x in the graph $G - (v_i, x)$. Then there are two distinct paths from z_{p-1} to x, one uses the tree edge (v_i, x) and the other does not. Thus there are two distinct paths from s to x - this contradicts that G is a unique-path graph. (only if) Let $v_j \in N^-(x)$ and $v_j \neq \pi(x)$. As the landmark vertices are placed $\sqrt{n}$ distance apart along the active path, by Lemma 6, x will be discovered by a $\sqrt{n}$-bounded DFS from the last landmark vertex z_{p-1} in the graph $G - (v_j, x)$. ∎

The number of $\sqrt{n}$-bounded DFS to implement the backtrack step from x is at most $|N^-(x)|$. From Lemma 6, each $\sqrt{n}$-bounded DFS takes time $O(n + m\sqrt{n})$. Hence all the backtrack steps performed in G can be implemented in $O(mn + m^2\sqrt{n})$ time and $O(\sqrt{n})$ space.

Discovery Step

The goal of the discovery step at a current vertex x is to check if a vertex $y \in N^+(x)$ has already been visited by the DFS. By Lemma 3, a DFS from s in the unique-path graph G cannot produce any forward or cross edges; hence this is equivalent to checking if the edge (x, y) is a back edge.

Procedure 3 gives the implementation of the discovery step. If y is one of the landmark vertices then clearly (x, y) is a back edge, i.e., Case 1 in Procedure 3 returns the correct output. Otherwise let $Z = Z(y)$ be the set of landmark vertices reachable from y by a $\sqrt{n}$-bounded DFS. Consider any back edge (x, y) such that y lies between the landmark vertices z_{j-1} and z_j ($j \geq 1$); then at least $z_j \in Z$. Hence if Z is empty, we know that (x, y) is not a back edge. Let $z_k \in Z$ be the landmark vertex with the highest index k in Z. Note that if (x, y) is a back edge, then $k \geq 1$. So the outputs of Case 2 and Case 3 are correct. But a $\sqrt{n}$-bounded DFS from y can discover more than one landmark vertex, since there can be successive back edges. The relation between z_j and z_k is described by the following lemma when (x, y) is a back edge.

LEMMA 8. *(a) If $j < p - 1$ or $j = p$, then z_j is the landmark vertex with the highest index j in Z. (b) If $j = p - 1$, then z_{p-1} or z_p is the landmark vertex with the highest index j in Z.*

PROOF. As distance of z_j from y is $\leq \sqrt{n}$, by Lemma 6, z_j will be discovered by a $\sqrt{n}$-bounded DFS from y, i.e., $z_j \in Z$. (a) If $j = p$, then z_p has the highest index p in Z since z_p is the last landmark vertex. If $j < p - 1$, the distance between y and z_ℓ is $> \sqrt{n}$ for any $\ell > j$.

Procedure 3 Procedure to implement the discovery step for the edge (x, y)

```
1: – Let $z_0 = s, z_1, \cdots, z_p = x$ be the current set of landmark vertices.
2: if $y \in \{z_0, \cdots, z_{p-1}\}$ then {(Case 1): $y$ is one of the landmark vertices}
3:    – Return '$(x, y)$ is a back edge' ($y \neq z_p$, since there are no self loops).
4: end if
5: – Perform a $\sqrt{n}$-bounded DFS from $y$ and let $Z$ be the set of landmark vertices reached
   by this DFS.
6: if $Z$ is empty then {(Case 2): no landmark vertex is reached}
7:    – Return '$y$ has not been visited'.
8: else
9:    – Let $z_j \in Z$ be the landmark vertex with the highest index $j$.
10:   if $j = 0$ then {(Case 3): $Z = \{z_0(= s)\}$}
11:      –Return '$y$ has not been visited'.
12:   else if $(j < p)$ then {(Case 4)}
13:      – Perform a second $\sqrt{n}$-bounded DFS from $z_{j-1}$, and terminate the DFS as soon as
         one of $z_j$ or $y$ is discovered.
14:      if $y$ is discovered then
15:         – Return '$(x, y)$ is a back edge'.
16:      else
17:         – Return '$y$ has not been visited'.
18:      end if
19:   else {(Case 5): $z_p(= x)$ is the landmark vertex $z_j$ with highest index $j$}
20:      – Perform a $2\sqrt{n}$-bounded DFS from $z_{p-2}$ and terminate the DFS as soon as one of
         $x$ or $y$ is discovered. (if $p = j = 1$, perform a $\sqrt{n}$-bounded DFS from $z_0$).
21:      if $y$ is discovered then
22:         – Return '$(x, y)$ is a back edge'.
23:      else
24:         – Return '$y$ has not been visited'.
25:      end if
26:   end if
27: end if
```

Hence by Lemma 6 the $\sqrt{n}$-bounded DFS from y cannot discover z_ℓ. (b) If $j = p - 1$, then $z_{p-1} \in Z$, but depending on the distance of the current vertex $z_p = x$ from z_{p-1}, z_p may or may not belong to Z. ∎

The following lemma proves the correctness of Case 4; the correctness of Case 5 can be proved similarly.

LEMMA 9. *An edge (x, y) is a back edge iff we terminate with the discovery of vertex y by a $\sqrt{n}$-bounded DFS from z_{j-1} (i.e. y is discovered before z_j).*

PROOF. (only if) Suppose (x, y) is a back edge. Thus y is an ancestor of x. A $\sqrt{n}$-bounded DFS from y discovers exactly one landmark vertex that is a descendant of y since landmark vertices are spaced $\sqrt{n}$ apart. Since the highest-indexed landmark vertex discovered from

y is z_j, z_j is a descendant of y and hence y is on the unique path from z_{j-1} to z_j. Therefore y will be discovered before z_j by the $\sqrt{n}$-bounded DFS from z_{j-1}. (if) Suppose y is a newly discovered vertex. Then no landmark vertex is a descendant of y and the unique path from z_{j-1} to y is through z_j. So the $\sqrt{n}$-bounded DFS from z_{j-1} cannot discover y before z_{j-1}. ∎

Thus the discovery step involves at most two $\sqrt{n}$-bounded (or $2\sqrt{n}$-bounded) DFS. From Lemma 6, each discovery step takes time $O(n + m\sqrt{n})$. So all the discovery steps in the graph G can be performed in time $O(mn + m^2\sqrt{n})$ time and $O(\sqrt{n})$ space. This completes the proof of Theorem 4.

3.2 Improving to $O(n^\varepsilon)$ Space

Applying the ideas above recursively, we can improve the space bound to n^ε for any $\varepsilon \in (0, 1]$ while still achieving a polynomial time bound. We will prove the following theorem in this section.

THEOREM 10. *For any $\varepsilon \in (0, 1]$, STCON in unique-path graphs is solvable with $O(\frac{n^\varepsilon}{\varepsilon})$ space in $n^{O(\frac{1}{\varepsilon})}$ time.*

We first modify the definition of the landmark vertices. Now the landmark vertices will be spaced $n^{1-\varepsilon}$ distance apart on the current search path, so that they can be stored using $O(n^\varepsilon)$ space. Note that the immediate problem in increasing the spacing of the landmark vertices is that, in both the backtrack and discovery steps, landmark vertices may not be reachable by a n^ε-bounded DFS. So we need to apply the ideas of the previous section recursively.

We define the procedure D-REACH(u, U, H, d), where H is a subgraph of the unique-path graph G with source s[‖], $u \in V(H)$ and u is reachable from s, $U \subseteq V(H), 1 \le d \le |V(H)| - 1$. This procedure decides if there exists a vertex $v \in U$ within distance d from u in H. If such a $v \in U$ exists, then the procedure returns the first such vertex v and terminates; otherwise it outputs that no such vertex in U exists. A variant called D-REACH-ALL(u, U, H, d) determines *all* vertices in U reachable in distance d from u. This variant has the same time and space complexity as D-REACH. The STCON problem is same as D-REACH$(s, \{t\}, G, n - 1)$.

LEMMA 11. *If $|U| \le n^\varepsilon$, the procedure* D-REACH(u, U, H, n^ε)*, can be implemented in $O(n + mn^\varepsilon)$ time using $O(n^\varepsilon)$ space.*

PROOF. The set U, $|U| \le n^\varepsilon$, can be stored in $O(n^\varepsilon)$ space. Since u is reachable from s, D-REACH(u, U, H, n^ε) can be implemented like an L-bounded DFS by storing the entire active path; thus the backtrack step takes $O(1)$ time for each vertex. For each edge (x, y), the discovery step is performed by checking (i) if y belongs to the active path (of $\le n^\varepsilon$ length) and (ii) if y is a new vertex, then whether it belongs to U, where $|U| \le n^\varepsilon$ (and in that case the procedure returns with output y). Clearly the discovery step can be performed in $O(n^\varepsilon)$ time. Hence $O(n + mn^\varepsilon)$ time suffices to implement D-REACH(u, U, H, n^ε). ∎

[‖]G_1 is a *subgraph* of G_2 if $V(G_1) \subseteq V(G_2)$ and $E(G_1) \subseteq E(G_2)$.

Suppose we are at the current vertex x and $z_0 = s, z_1, \cdots, z_p = x$ is the set of landmark vertices stored at the top-most level of the recursion. In the backtrack step, similar to Procedure 2, for each $u \in N^-(x)$ we need to check if the landmark vertex z_{p-1} can reach the current vertex $z_p = x$ in the graph $G - (u, x)$. As the distance of x from z_{p-1} can be at most $n^{1-\varepsilon}$, for each $u \in N^-(x)$, we recursively call D-REACH$(z_{p-1}, \{x\}, G - (u, x), n^{1-\varepsilon})$. For the entire graph G, we have to make at most m such calls. Similarly for each discovery step, we have to call a $n^{1-\varepsilon}$-bounded D-REACH-ALL procedures first, and then we may have to call either a $n^{1-\varepsilon}$-bounded or a $2n^{1-\varepsilon}$-bounded D-REACH procedure (depending on the cases in Procedure 3). Using the same notations as in Section 3.1, these recursive procedures are: (i) D-REACH-ALL$(y, Z_{cur}, G, n^{1-\varepsilon})$ (where Z_{cur} is the current set of landmark vertices at the top-most level), (ii) if z_j is the highest indexed landmark vertex found, a second call is made either to D-REACH$(z_{j-1}, \{z_j, y\}, G, n^{1-\varepsilon})$ (in Case 4) or to D-REACH$(z_{p-2}, \{z_p, y\}, G, 2n^{1-\varepsilon})$ (in Case 5). Hence, there are at most $2m$ calls to $n^{1-\varepsilon}$-bounded D-REACH procedures (invoked by the backtrack and discovery steps) and at most m calls to $n^{1-\varepsilon}$-bounded or $2n^{1-\varepsilon}$-bounded D-REACH procedures (invoked by the discovery step).

Let $T(m, n, d)$ denote the running time of D-REACH(u, U, H, d) ($|U| \leq n^{\varepsilon}$), when H has at most n nodes and m edges. Note that, in each call to the D-REACH and D-REACH-ALL procedures used by our algorithm, $|U| \leq n^{\varepsilon}$ (since U is a subset of the landmark vertices). Hence we have the following recursion. $T(m, n, n) \leq 2mT(m, n, n^{1-\varepsilon}) + m \max(T(m, n, n^{1-\varepsilon}), T(m, n, 2n^{1-\varepsilon})) + O(m + n)$, i.e. $T(m, n, n) \leq 3mT(m, n, 2n^{1-\varepsilon}) + O(m + n)$. As the base case we have, $T(m, n, n^{\varepsilon}) = O(n + mn^{\varepsilon}) = (m + n)^{O(1)}$. The first two parameters in the recurrence relation are not changed at any step and they do not play active role in the solution of the recurrence. The solution to this recurrence is $(m + n)^{O(1+\frac{1}{\varepsilon})} = n^{O(\frac{1}{\varepsilon})}$, which is a polynomial when ε is a constant.

Next we analyze the increased space requirement due to these recursive calls. Note that, at any point of time we have to remember the landmark vertices at all levels of the recursion. But we can reuse the space allocated to landmark vertices in successive DFS calls at the same recursion level. The recursion depth is at most $\frac{1}{\varepsilon}$. Hence we have to remember at most $O(\frac{n^{\varepsilon}}{\varepsilon})$ vertices. So the overall space complexity of this recursive algorithm is $O(\frac{n^{\varepsilon}}{\varepsilon})$. This proves Theorem 10.

4 Recognition of Unique-Path Graphs

We prove the following theorem in this section. But due to space constraint, we omit the proof of the theorem.

THEOREM 12. *Given a directed graph G and a vertex $s \in V(G)$, there is an $O(\frac{n^{\varepsilon}}{\varepsilon})$-space, $n^{O(\frac{1}{\varepsilon})}$-time algorithm to decide whether G is a unique-path graph with respect to source s.*

5 Conclusions

An interesting open question is whether there are polynomial-time, polylog-space algorithms for STCON in unique-path graphs. It would also be interesting to see if our ideas can be extended to obtain an $O(n^{\varepsilon})$-space polynomial-time algorithm for STCON in a more general family of graphs.

References

[1] R. ALELIUNAS, R. M. KARP, R. J. LIPTON, L. LOVASZ, AND C. RACKOFF. Random walks, universal traversal sequences, and the complexity of maze problems, *FOCS*, (1979) 218–223.

[2] E. ALLENDER. Reachability Problems: An Update, *CiE*, (2007) 25–27.

[3] E. ALLENDER, D. A. M. BARRINGTON, T. CHAKRABORTY, S. DATTA, AND S. ROY. Grid Graph Reachability Problems, *CCC*, (2006) 299–313.

[4] E. ALLENDER AND K.-J. LANGE. RUSPACE($\log n$) $\subseteq$ DSPACE ($\log^2 n / \log\log n$), *Theory Comput. Syst.*, 31(5), (1998) 539–550.

[5] C. ÁLVAREZ AND B. JENNER. A very hard log-space counting class, *Theor. Comput. Sci.*, 107(1), (1993) 3–30.

[6] G. BARNES, J. F. BUSS, W. L. RUZZO, AND B. SCHIEBER. A Sublinear Space, Polynomial Time Algorithm for Directed s-t Connectivity, *SIAM J. Comput.*, 27(5), (1998) 1273–1282.

[7] P. BERMAN AND J. SIMON. Lower Bounds on Graph Threading by Probabilistic Machines (Preliminary Version), *FOCS*, (1983) 304–311.

[8] A. BORODIN. Time Space Tradeoffs (Getting Closer to the Barrier?), *ISAAC*, (1993) 209–220.

[9] G. BUNTROCK, B. JENNER, K.-J. LANGE, AND P. ROSSMANITH. Unambiguity and Fewness for Logarithmic Space, *FCT*, (1991) 168–179.

[10] S. A. COOK AND C. RACKOFF. Space Lower Bounds for Maze Threadability on Restricted Machines, *SIAM J. Comput.*, 9(3), (1980) 636–652.

[11] T. H. CORMEN, C. E. LEISERSON, R. L. RIVEST, AND C. STEIN. Introduction to Algorithms, Second Edition, *The MIT Press*, (2001).

[12] J. EDMONDS AND C. K. POON. A nearly optimal time-space lower bound for directed *st*-connectivity on the NNJAG model, *STOC*, (1995) 147–156.

[13] C. M. PAPADIMITRIOU. *Computational complexity*, Addison-Wesley, (1994).

[14] C. K. POON. Space Bounds for Graph Connectivity Problems on Node-named JAGs and Node-ordered JAGs, *FOCS*, (1993) 218–227.

[15] O. REINGOLD. Undirected ST-connectivity in log-space, *STOC*, (2005) 376–385.

[16] O. REINGOLD, L. TREVISAN, AND S. P. VADHAN. Pseudorandom walks on regular digraphs and the RL vs. L problem, *STOC*, (2006) 457–466.

[17] K. REINHARDT AND E. ALLENDER. Making Nondeterminism Unambiguous, *FOCS*, (1997) 244–253.

[18] W. J. SAVITCH. Relationships between nondeterministic and deterministic tape Relationships Between Nondeterministic and Deterministic Tape Complexities, *J. Comput. Syst. Sci.*, 4(2), (1970) 177–192.

[19] A. WIGDERSON. The Complexity of Graph Connectivity, *MFCS*, (1992) 112–132.