

# Relating Sublinear Space Computability Among Graph Connectivity and Related Problems

Tatsuya Imai<sup>1</sup> and Osamu Watanabe<sup>2</sup>(✉)

<sup>1</sup> Heroz, Inc., Tokyo, Japan

<sup>2</sup> Department of Mathematical and Computer Science,  
Tokyo Institute of Technology, Tokyo 152-8552, Japan  
`watanabe@is.titech.ac.jp`

**Abstract.** We investigate sublinear-space computability relation among the directed graph vertex connectivity problem and its related problems, where by “sublinear-space computability” we mean in this paper  $O(n^{1-\varepsilon})$ -space and polynomial-time computability w.r.t. the number  $n$  of vertices. We demonstrate algorithmic techniques to relate the sublinear-space computability of directed graph connectivity and undirected graph length bounded connectivity.

## 1 Introduction and Preliminaries

Space complexity is one of the important complexity measures. In general algorithms with small complexity are important, but recently, due to the increase of data size, we face demands for sublinear-space algorithms in various applications, that is, demands for algorithms using much smaller working memory than input data size. Sublinear-space computability is also important from a theoretical view point for understanding the nature of computation. For example, the famous  $L = NL$  question is about the  $O(\log n)$ -space computability of the following directed graph connectivity problem. (Although we formulate in this paper connectivity problems as a problem of asking the connectivity of a given pair of vertices, the space complexity is the same even if we consider the connectivity for *all* pairs of vertices.)

stConn (Directed Graph Connectivity)

**input:** Directed graph  $G = (V, E)$  and vertices  $s, t \in V$ .

**task:** Determine whether there exists a path from  $s$  to  $t$ .

**size parameter:** The number of vertices, denoted by  $n$ .

In order to understand the  $O(\log n)$ -space (in)computability of stConn, different versions of this connectivity problem have been investigated, and various important results have been obtained. For example, the breakthrough result of Reingold [7] shows that the connectivity is  $O(\log n)$ -space decidable for undirected graphs. On the other hand, not so much has been studied for a bit more relaxed  $o(n)$ -space computability. In this paper, we consider one of such  $o(n)$ -space bounds, that is,  $O(n^{1-\varepsilon})$ -space bound defined by “saving” parameter  $\varepsilon > 0$ .

The stConn problem may not be solvable in  $O(\log n)$ -space, but it may still be solvable in  $o(n)$ -space and *and* polynomial-time. In fact, Barnes et al. [3] gave an  $O(n/2^{\sqrt{\log n}})$ -space and polynomial-time algorithm. But we aim for a stronger  $o(n)$ -space bound, that is,  $O(n^{1-\varepsilon})$ -space computability as Widgerson asked in [8]. Here we also require<sup>1</sup> the polynomial-time computability, which is crucial from both theoretical and practical view points. In fact, we have an  $O((\log n)^2)$ -space (and  $O(n^{\log n})$ -time) algorithm for stConn from Savitch's theorem. We do not want to go beyond the polynomial-time bound for reducing working memory. Thus, in this paper we consider both polynomial-time *and*  $O(n^{1-\varepsilon})$ -space computability, which we will call *sublinear-space computability* throughout this paper.

Recently, sublinear-space computability has been shown for some graph classes [1, 2, 4, 6]. For example, for the directed *planar* graph connectivity problem, we have an  $O(\sqrt{n})$ -word-space and polynomial-time algorithm [2]. Unfortunately, however, an essential gap seems to exist to extend it to the general case. In this paper we would like to identify a requirement/restriction that makes the problem difficult. Certainly, directedness is a key for the hardness because the connectivity is decidable in  $O(\log n)$ -space for undirected graphs. As an alternative to directedness, we consider “bounded length” requirement; that is, we consider the problem that asks, for a given  $b$ , whether there is a path from  $s$  to  $t$  consisting of at most  $b$  edges, in other words,  $s$  is connected to  $t$  by a path of “length” at most  $b$ . Let us use  $\text{UstConn}_{\text{lb}}$  to denote this version of *undirected* graph connectivity problem. It has been known that stConn is  $O(\log n)$ -reducible to  $\text{UstConn}_{\text{lb}}$ ; that is, the difficulty of solving stConn in  $O(\log n)$ -space can be transformed to  $\text{UstConn}_{\text{lb}}$ , or more specifically, we have  $\text{stConn} \notin \text{L} \Rightarrow \text{UstConn}_{\text{lb}} \notin \text{L}$ . We ask in this paper whether this type of relation holds for their sublinear-space computability.

As a main result, we show a way to relate the sublinear-space computability of  $\text{UstConn}_{\text{lb}}$  to that of stConn with almost same saving. This can be regarded as an sublinear-space approximately preserving reduction. We also explain the idea of a similar sublinear-space approximately preserving reduction. Therefore, we can conclude that directedness and length bound are computationally equivalent requirements also in the sublinear-space computability context. While we leave it open to extend this technique to other NL problems, we show similar relation holds for another graph connectivity problem and its length bounded version, which we hope to give us a hint to obtain a more general technique relating sublinear-space computability.

*Preliminaries.* We use standard notions and notation in graph theory and computational complexity theory. In this paper we consider both directed and undirected graphs, but we may assume that a graph is directed unless it is specified as undirected. A directed edge is denoted by an order pair of vertices, e.g.,  $(u, v)$ ,

<sup>1</sup> Any  $O(\log n)$ -space algorithm is (modified to) a polynomial-time algorithm; thus, it is not necessary to require the polynomial-time computability when discussing the log-space computability.

whereas an undirected edge is denoted by a set of vertices, e.g.,  $\{u, v\}$ . In the directed case, by a “path” we mean the sequence of directed edges having one direction from its source vertex to destination vertex. For any path, its *length* is the number of edges on the path. For any vertices  $u$  and  $v$ , a *shortest path* from  $u$  to  $v$  is a path from  $u$  to  $v$  with the smallest length, and by  $\text{len}(u, v)$  we denote the length of the shortest path from  $u$  to  $v$ .

We basically follow the standard machine based framework for discussing time and space complexity. We consider that input data is given separately in a read-only memory area, and space complexity is the amount of working memory used for computation. Precisely speaking, we should measure the number of bits; but in our context we may ignore a  $O(\log n)$  factor and use the number of working variables to measure space complexity. Throughout this paper we use  $n$  denote the number of vertices of a given graph, which is regarded as the main size parameter for all problems considered in this paper. We do not use the number of edges as a size parameter. This is because (i) the number of edges does not seem to be so relevant for discussing polynomial-time computability and space complexity, and (ii) we indeed have a polynomial-time and  $O(n)$ -size algorithms for various connectivity problems.

## 2 Length Bounded Undirected Graph Connectivity

As explained in Introduction, motivated by the  $O(\log n)$ -space computability of the undirected connectivity problem, we consider its length bounded version. That is, the following problem. (In this paper we use  $[k]$  to denote  $\{0, 1, \dots, k\}$  instead of  $\{1, \dots, k\}$ .)

UstConn<sub>lb</sub>

**input:** Undirected graph  $G = (V, E)$ ,  $s, t \in V$ , and integer  $b \in [n - 1]$ .

**task:** Determine whether there exists a path between  $s$  and  $t$  of length  $\leq b$ .

Nothing is known for the sublinear-space computability of this problem. Here we assume the following sublinear-space computability of this problem. That is, we assume that  $\varepsilon$  saving holds for this problem. We discuss whether a similar saving can be implied from this assumption for stConn.

**Assumption 1.** *There is an algorithm `Algo_UstConnlb` that solves UstConn<sub>lb</sub> in polynomial-time and  $O(n^{1-\varepsilon})$ -space.*

It has been well known that UstConn<sub>lb</sub> is also NL-complete problem. In particular, there is a standard log-space many-one reduction from stConn to UstConn<sub>lb</sub>, and any  $O(\log n)$ -space algorithm solving UstConn<sub>lb</sub> can be used to give an  $O(\log n)$ -space algorithm for stConn. Let us recall this reduction first.

Consider any instance  $(G, s, t)$  of stConn, where  $G = (V, E)$  is a directed graph and  $s$  and  $t$  are vertices in  $V$ . The reduction creates a “layered” undirected graph  $nG = (nV, nE)$  that is defined by

$$\begin{aligned}
nV &= \{ (i, v) \mid i \in [n-1] \text{ and } v \in V \}, \text{ and} \\
nE &= \{ \{ (i, u), (i+1, v) \} \mid i \in [n-2] \text{ and } (u, v) \in E \} \\
&\quad \cup \{ \{ (i, u), (i+1, u) \} \mid i \in [n-2] \text{ and } u \in V \}.
\end{aligned}$$

Then it is easy to see the following property holds. Thus, a mapping  $(G, s, t)$  to  $(nG, (0, s), (n-1, t), n-1)$  is a reduction from  $\text{stConn}$  to  $\text{UstConn}_{\text{lb}}$ .

*Claim.* For any  $s, t \in V$ , there is a path from  $s$  to  $t$  in  $G$  if and only if there is a path from  $(0, s)$  to  $(n-1, t)$  in  $nG$  of length  $n-1$ .

Let **Algo.red** denote the algorithm solving  $\text{stConn}$  by using this reduction method and **Algo.UstConn<sub>lb</sub>**. Note that the instance given to **Algo.UstConn<sub>lb</sub>** has  $n^2$  vertices. Thus, we have the following complexity bounds.

**Lemma 1.** ***Algo.red** solves  $\text{stConn}$  in polynomial-time and  $O((n^2)^{1-\varepsilon})$ -space.*

Note that  $O(n^{2(1-\varepsilon)}) = O(n^{1-(2\varepsilon-1)})$ ; this space bound is still sublinear if  $\varepsilon > 0.5$ . But clearly, the saving got reduced considerably due to the increase of the graph size, i.e., the number of vertices, by the reduction.

We introduce two algorithmic ideas to suppress this graph size increase of the standard reduction. For this we consider the length bounded connectivity also in a directed graph, and we introduce the notion of “bounded length” below. Consider any directed graph  $G = (V, E)$ , and let  $\text{leng}(u, v)$  denote the length from  $u$  to  $v$  on this graph. For any  $u, v \in V$  and for any  $b \in [n-1]$ , we define  $\text{leng\_bl}(u, v, b)$  by

$$\text{leng\_bl}(u, v, b) = \begin{cases} \text{leng}(u, v), & \text{if } \text{leng}(u, v) \leq b, \text{ and} \\ \perp, & \text{otherwise.} \end{cases}$$

We call this function *bounded length*. When necessary, we write  $\text{leng\_bl}(G; u, v, b)$  for explicitly expressing the length is considered on  $G$ . Note that deciding whether  $\text{leng\_bl}(u, v, b) \neq \perp$  is exactly the length bounded connectivity on the directed graph  $G$ . We can generalize the reduction based algorithm **Algo.red** to determine whether  $\text{leng\_bl}(u, v, b) \neq \perp$  in polynomial-time and  $O((bn)^{1-\varepsilon})$ -space. Let us still use **Algo.red** to denote this algorithm, and let  $t_{\text{red}}(bn)$  denote a polynomial time bound for **Algo.red** to determine  $\text{leng\_bl}(u, v, b) \neq \perp$ .

Clearly, the graph size increase can be suppressed if  $\text{stConn}$  can be solved by using only  $\text{leng\_bl}(\cdot, \cdot, b)$  with small  $b$ . One simple idea is to compute bounded length recursively. For example, consider the following graph  $G(b) = (V, E(b))$ , where

$$E(b) = \{ (u, v) \mid \text{leng\_bl}(u, v, b) \neq \perp \}.$$

Apply **Algo.red** on  $G(b)$  to determine  $\text{leng\_bl}(G(b); u, v, b) \neq \perp$ . Whenever **Algo.red** needs to see whether an edge  $(u, v)$  exists in  $G(b)$ , we run **Algo.red** on  $G$  to determine  $\text{leng\_bl}(G; u, v, b) \neq \perp$ . Clearly, we have  $\text{leng\_bl}(G(b); u, v, b) \neq \perp$  if and only if  $\text{leng\_bl}(G; u, v, b^2) \neq \perp$ , and it is easy to see that this depth two recursion for deciding  $\text{leng\_bl}(G(b); u, v, b) \neq \perp$  can be done in  $O((t_{\text{red}}(bn))^2)$ -time and  $O(2(bn)^{1-\varepsilon})$ -space.

We can extend this idea and use **Algo\_red** on  $G^r(b) = (V, E^r(b))$ , where  $E^r(b)$  is defined inductively by  $E^r(b) = \{(u, v) \mid \text{leng\_bl}(G^{r-1}(b): u, v, b) \neq \perp\}$ . Let **Algo\_red<sup>r</sup>** denote the algorithm that determines  $\text{leng\_bl}(G^{r-1}(b): u, v, b) \neq \perp$  by using **Algo\_red** recursively up to depth  $r$ . That is, **Algo\_red<sup>r</sup>** determines  $\text{leng\_bl}(G^{r-1}(b): u, v, b) \neq \perp \iff \text{leng\_bl}(G: u, v, b^r) \neq \perp$ . Thus, for solving an stConn instance  $(G, s, t)$ , it is enough to compute  $\text{leng\_bl}(G^{r-1}: s, t, b)$  with  $b = n^{1/r}$  by **Algo\_red<sup>r</sup>**. This gives the following bounds.

**Lemma 2.** *For any  $r \geq 1$ , **Algo\_red<sup>r</sup>** solves stConn in  $O((t_{\text{red}}(n^{1+1/r})^r)$ -time and  $O(rn^{(1+1/r)(1-\varepsilon)})$ -space.*

**Remark.** *Though the parameter  $r$  need not be a constant for the algorithm, it must be a constant in order to bound the running time by polynomial. Then we can simplify the above space bound by*

$$O(n^{(1+1/r)(1-\varepsilon)}) = O(n^{1-((1+1/r)\varepsilon-1/r)}).$$

Hence, the saving of **Algo\_red<sup>r</sup>** is  $(1 + 1/r)\varepsilon - 1/r$ .

Unfortunately, the above saving is still not so good. In order to have a non-trivial saving we need to choose  $r > 1/\varepsilon$ , which makes the time bound very high (while it is still polynomial). We can overcome this problem by using the idea in [3]. Barnes et al. [3] gave a weak sublinear-space algorithm. Their algorithm is a combination of two algorithms B1 and B2; algorithm B2, which is used by B1 as a subroutine, is in fact computes the bounded length, i.e.,  $\text{leng\_bl}(\cdot, \cdot, L)$  for relatively small  $L$ . Here we use the B1 part of their algorithm.

We first show that  $\text{leng\_bl}(G: u, v, b^r)$  is indeed sublinear-space computable by using decision algorithm **Algo\_red<sup>i</sup>** and its modification. The idea is simple. Consider any  $u, v \in V$ . If  $\text{leng\_bl}(u, v, b^r) = \perp$ , then we are done. (Here and below by  $\text{leng\_bl}(\cdot, \cdot, \cdot)$  we mean  $\text{leng\_bl}(G: \cdot, \cdot, \cdot)$ .) Otherwise, it suffices to test whether  $\text{leng\_bl}(u, v, d) \neq \perp$  holds for all  $d \in [b^r - 1]$ ; we have  $\text{leng\_bl}(u, v, b^r) = d$  with the minimum  $d$  such that  $\text{leng\_bl}(u, v, d) \neq \perp$  holds. Note here that  $d = a_0 + a_1b + a_2b^2 + \dots + a_{r-1}b^{r-1}$  for some  $a_0, \dots, a_{r-1} \in [b - 1]$ . Then we can test whether  $\text{leng\_bl}(u, v, d) \neq \perp$  holds by using a layered graph  $(rV, rE')$  similar to  $(rV, rE)$ . Here  $rE'$  is defined as follows: for each  $i \geq 0$ ,  $rE'$  has an edge  $\{(i, u), (i + 1, v)\}$  if and only if  $\text{leng\_bl}(u, v, a_i b^i) \neq \perp$ , which can be tested by using a slightly modified **Algo\_red<sup>i+1</sup>**. Hence,  $\text{leng\_bl}(u, v, d) \neq \perp$  can be tested in time  $O(t_{\text{red}}(rn)t_{\text{red}}(bn)^r) = O(t_{\text{red}}(bn)^{r+1})$  and in space  $O((rn)^{1-\varepsilon} + (bn)^{1-\varepsilon}) = O((bn)^{1-\varepsilon})$  (since we may assume that  $r < b$ ). Therefore,  $\text{leng\_bl}(u, v, b^r)$  is computable as follows.

**Lemma 3.** *We have an algorithm **Algo\_bl** that computes  $\text{leng\_bl}(G: u, v, b^r)$  in  $O(b^r t_{\text{red}}(bn)^{r+1})$ -time and  $O((bn)^{1-\varepsilon})$ -space.*

Next we introduce a key tool, namely, a small “separator.” Consider any instance for stConn, i.e.,  $G = (V, E)$  and  $s, t \in V$ , and fix them in the following

explanation. Let  $L$  be an algorithm parameter that is determined later. For any  $j \in [L - 1]$ , let  $V_j$  be a subset of  $V$  defined by

$$V_j = \{ v \mid \text{leng}(s, v) \bmod L = j \}.$$

Family  $\{V_j\}_{j \in [L-1]}$  has several important properties. First, it is a partition of the set of vertices of  $V$  reachable from  $s$ ; and hence, there should be some  $j_0$  such that  $|V_{j_0}| \leq n/L$ . We use such  $V_{j_0}$  to record the reachability from  $s$ , thereby reducing the space complexity for memorization. Another important property, though it is trivial, is that each  $V_j$  is a separator of all shortest paths of length  $\geq j$ . More specifically, for any  $j \in [L - 1]$ , if  $\text{leng}(s, v) \geq j$ , then there should be some vertex  $u$  in  $V_j$  that is on one of the shortest paths from  $s$  to  $v$  and for which  $\text{leng}(u, v) < L$  holds; thus, once we have  $V_{j_0}$ , we only need to compute, for each  $u \in V_{j_0}$ , its bounded length to  $t$ , i.e.,  $\text{leng\_bl}(u, t, L)$ , to determine the connectivity from  $s$  to  $t$ .

These properties justify the following algorithm outline for deciding connectivity from  $s$  to  $t$  in  $G$ : (1) Compute  $V_j$  for each  $j \in [L - 1]$  from  $j = 0$  to  $L - 1$ . If the algorithm finds that  $|V_j| > n/L$ , then it stops the computation of  $V_j$  and moves to the computation of  $V_{j+1}$ . On the other hand, move to the next step as soon as  $V_j$  with appropriate size can be computed. (2) Compute  $\text{leng\_bl}(u, t, L)$  for each  $u \in V_{j_0}$ . Output “yes” if there is some  $u \in V_{j_0}$  such that  $\text{leng\_bl}(u, t, L) \neq \perp$ . Otherwise, output “no.” This is essentially the **B1** part of the algorithm of Barnes et al., and we name the algorithm that solves **stConn** following this outline as **Algo\_Betal**; note that we assume that the algorithm **Algo\_bl** of Lemma 3 is used here for computing the bounded length.

Now it remains to implement the above step (1). Below we give a procedure for computing  $V_j$  for a given  $j \in [L - 1]$ ; the actual computation of (1) is to use this procedure to find  $j_0$  for which the procedure successfully computes  $V_{j_0}$ . For this procedure, we can again use the bounded length. In [3] **B1** is stated as a breadth first algorithm, but here for the sake of later explanation, we state it as a “closest vertex first” algorithm, which can be regarded as a variation of the Dijkstra’s algorithm. In the following procedure, we use a variable  $D$  to denote the set of vertices in  $V_j$  whose length from  $s$  has been determined, and for any  $v \in D$ , we use  $d[v]$  to record this length. For any  $u \in D$  and  $v \in V \setminus D$ , we define a function  $\text{cost\_via}(u, v)$  by

$$\text{cost\_via}(u, v) = d[u] + \text{leng\_bl}(u, v, L).$$

That is,  $\text{cost\_via}(u, v)$  is the length of a path from  $s$  to  $v$  that has  $u$  in  $V_j$ . It is easy to see that  $u$  is the vertex in  $V_j$  on the path closest to  $v$ .

The correctness of the procedure will be explained in the next section for a more general procedure. Here we analyze the time and space complexity bounds of **Algo\_Betal**. Note that the most time consuming part is the computation of the bounded length  $\text{leng\_bl}(\cdot, \cdot, L)$ , and it is easy to see that the bounded length is computed at most  $O(Ln^3)$  times; thus, by using **Algo\_bl** of Lemma 3 for computing the bounded length, we can bound the total running time by  $O(Ln^3 L(t_{\text{red}}(L^{1/r}n))^{r+1})$ , which is roughly  $\text{poly}(n)^r$ . On the other hand, we can

```

procedure for computing  $V_j$ 
   $d[s] \leftarrow 0$ ;  $D \leftarrow \{s\}$ ;
  while  $v_{\min} \neq \perp$  do {
     $v_{\min} \leftarrow \perp$ ;  $\text{cost}_{\min} \leftarrow +\infty$ ;
    for each  $v \in V \setminus D$  do {
       $u_{\text{closest}} \leftarrow \text{argmin}\{\text{cost\_via}(u, v) \mid u \in D\}$ ;
       $\text{cost} \leftarrow \text{cost\_via}(u_{\text{closest}}, v)$ ;
      if  $\text{leng\_bl}(u_{\text{closest}}, v, L) = L$ 
         $\wedge \text{cost} < \text{cost}_{\min}$  then {
           $v_{\min} \leftarrow v$ ;  $\text{cost}_{\min} \leftarrow \text{cost}$ ;
        }
      }
    }
    if  $v_{\min} \neq \perp$  then {
       $D \leftarrow D \cup \{v_{\min}\}$ ;  $d[v] \leftarrow \text{cost}_{\min}$ ;
      if  $|D| > n/L$  then report failure and stop;
    }
  }
  return  $D$  as  $V_j$ ;

```

**Fig. 1.** Procedure for computing  $V_j$  in **Algo.Betal**

bound the space complexity of **Algo.Betal** by  $O(n/L + (L^{1/r}n)^{1-\varepsilon})$ . This space bound is (approximately) minimized by choosing  $L$  to satisfy  $n/L = (L^{1/r}n)^{1-\varepsilon}$ , or equivalently,  $n^\varepsilon = L^{1+(1-\varepsilon)/r}$ . With this choice of  $L$ , we can bound the space complexity by  $O(n^\alpha)$ , where

$$\alpha = 1 - \frac{\varepsilon}{1 + (1 - \varepsilon)/r} = 1 - \varepsilon \left( 1 - \frac{1 - \varepsilon}{r + (1 - \varepsilon)} \right) < 1 - \varepsilon \left( 1 - \frac{1}{r + 1} \right).$$

**Theorem 1.** *Using the sublinear-space algorithm assumed by Assumption 1, algorithm **Algo.Betal** solves stConn in  $O((\text{poly}(n))^r)$ -time and  $O(n^{1-(1-1/(r+1))\varepsilon})$ -space.*

We may regard this result as a reduction from stConn to UstConn<sub>lb</sub> that preserve approximate sublinear-space computability, where by “approximate” we mean that one can get a saving as arbitrarily close to the original saving. Here let us call intuitively our construction of algorithm **Algo.Betal** as a *sublinear-space approximately preserving reduction* without giving any formal definition. Naturally we may ask whether this sublinear-space approximately preserving reduction exists also from UstConn<sub>lb</sub> to stConn. Here again we can use a similar idea to show such a reduction.

Consider any instance  $(G, s, t, b)$  of UstConn<sub>lb</sub>, where  $G = (V, E)$  is an undirected graph,  $s, t \in V$ , and  $b \in [n - 1]$ . We consider a layered directed graph  $nG = (nV, nE)$  defined by

$$\begin{aligned}
nV &= \{ (i, v) \mid i \in [b] \text{ and } v \in V \}, \text{ and} \\
nE &= \{ ((i, u), (i + 1, v)) \mid i \in [b - 1] \text{ and } \{u, v\} \in E \} \\
&\quad \cup \{ \{ (i, u), (i + 1, u) \} \mid i \in [b - 1] \text{ and } u \in V \}.
\end{aligned}$$

Then again it is easy to see that  $\text{leng\_bl}(s, t, b) \neq \perp$  if and only if there is a directed path from  $(0, s)$  to  $(b, t)$  in  $nG$ . Therefore, using an argument similar to the above, we can also define a sublinear-space approximately preserving reduction from  $\text{UstConn}_{\text{lb}}$  to  $\text{stConn}$ . The detail construction as well as giving a formal definition to the notion of “sublinear-space approximately preserving reduction” is left to the interest reader.

### 3 Another Example: Two Vertex Distance Problem

Although we have close sublinear-space computability relation between  $\text{stConn}$  and  $\text{UstConn}_{\text{lb}}$ , it is not so clear whether similar relation holds with the other NL-problems. While we have not been able to develop a general result, we can show some example result that would give us a hint for applying our technique to other problems.

We consider here the problem of computing the “distance” between two vertices in a directed and weighted graph. A weighted graph is a graph whose edge is given a cost. In this explanation we assume that each cost is a positive integer; furthermore, in order to avoid introducing another size parameter, we also assume that each cost can be expressed by  $\text{poly}(\log n)$  bits so that cost computation can be trivially done in  $\text{poly}(\log n)$ -space. For specifying a cost at each edge, we use a *cost function*, a mapping from an edge to its cost. For example, a weighted directed/undirected graph is given by  $G = (V, E, c)$ , where  $c$  is a mapping from  $E$  to its cost. Consider any pair of vertices  $u$  and  $v$  of some weighted graph. For any path from  $u$  to  $v$ , its *weight* is the sum of the cost of edges on the path. A *lightest path* from  $u$  to  $v$  is a path from  $u$  to  $v$  with the smallest weight, and the *distance* from  $u$  to  $v$  (denoted by  $\text{dist}(s, t)$ ) is the weight of the lightest path from  $u$  to  $v$ . We will keep using “length” to mean the number of edges and “shortest path” to mean a path (connecting a specified pair of vertices) with the smallest number of edges. In summary we consider the following problem.

stDist (Two Vertex Distance Problem)

**input:** Directed and weighed graph  $G = (V, E, c)$ ,  $s, t \in V$ , and  $d \geq 0$ .

**task:** Determine whether  $\text{dist}(s, t) \leq d$ .

**Remark.** For simplicity we assume in this paper that weights are integers expressed in  $\text{poly}(\log n)$  bits.

Clearly this problem is in NL. But it is not clear that a similar sublinear-space (approximately) preserving reduction from this problem to, e.g.,  $\text{stConn}$ . Yet, we can still consider similar relation to its undirected and length bounded version. More specifically, consider the following problem.



UstDist<sub>lb</sub>

**input:** Undirected and weighed graph  $G = (V, E, c)$ ,  $s, t \in V$ ,  $d \geq 0$ ,  
and  $b \in [n - 1]$ .

**task:** Determine whether  $\text{dist\_bl}(s, t, b) \leq d$ .

Here  $\text{dist\_bl}(s, t, b)$  is the length bounded distance from  $s$  to  $t$ , that is, the weight of the lightest path from  $s$  to  $t$  of length  $\leq b$ ; we assume that  $\text{dist\_bl}(s, t, b) = \perp$  if there is no path of length  $\leq b$  from  $s$  to  $t$ . We use  $\text{dist\_bl}(s, t, b)$  also for directed graphs. Note that even if there is a path from  $s$  to  $t$  of length  $\leq b$  (i.e.,  $\text{dist\_bl}(s, t, b) \neq \perp$ ), we may not have  $\text{dist\_bl}(s, t, b) = \text{dist}(s, t)$ . (Cf. We have  $\text{leng\_bl}(s, t, b) = \text{leng}(s, t)$  if  $\text{leng\_bl}(s, t, b) \neq \perp$ .)

We again base an assumption that  $\text{UstDist}_{lb}$  has a polynomial-time and  $O(n^{1-\epsilon})$ -space algorithm, which we denote as  $\text{Algo\_UstDist}_{lb}$ . Using this algorithm, we show a sublinear-space algorithm for  $\text{stDist}$  with an approximately same saving. The problems are somewhat complicated compared with  $\text{stConn}$  and  $\text{UstConn}_{lb}$ , which is mainly due to the above mentioned difference between  $\text{leng\_bl}$  and  $\text{dist\_bl}$ . Thus, for deriving the sublinear-space computability of  $\text{stDist}$  (based on  $\text{Algo\_UstDist}_{lb}$ ) there are some points where the previous argument need to be modified appropriately. We explain below such points.

Let us consider one instance for  $\text{stDist}$ ; that is, a directed and weighted graph  $G = (V, E, c)$  and  $s, t \in V$ . As before, our first step is to develop an algorithm based on a log-space many-one reduction from  $\text{stDist}$  to  $\text{UstDist}_{lb}$ . More specifically, for any  $u, v \in V$  and any bound  $b \geq 0$ , we define a layered and weighted undirected graph  $bG = (bV, bE, c^+)$  as before, with which we can decide whether  $\text{dist\_bl}(G: u, v, b) \leq d$  for a given  $d$  by using  $\text{Algo\_UstDist}_{lb}$ . Then we can use a binary search to determine the value of  $\text{dist\_bl}(G: u, v, b)$ . This algorithm is polynomial-time and  $O((bn)^{1-\epsilon})$ -space. The second step is also similar to the previous argument. We use a recursive way to compute  $\text{dist\_bl}(u, v, b^r)$  in  $O(r(bn)^{1-\epsilon})$ -space, while the computation time grows to  $\text{poly}(bn)^r$ . Here again we first define a decision algorithm and uses it to compute the value of  $\text{dist\_bl}(u, v, b^r)$  by a binary search. Now an interesting point is the last step where we use the idea of the algorithm of Barnes et al.

We introduce some new notions. Consider any two vertices  $u, v \in V$ . Note that a lightest path from  $u$  to  $v$  may not be unique. A shortest path among such lightest paths is called a *best path*. Note that there may be still more than one best paths, but we do not need to distinguish them in the following discussion. We introduce a function  $\text{bpleng}$  that gives the length of a best path; that is,  $\text{bpleng}(u, v)$  is the length of a best path from  $u$  to  $v$ . We also consider its length bounded version. For any integer  $b \geq 0$ , consider lightest paths from  $u$  to  $v$  of length  $\leq b$ ; then a length  $b$  bounded best path is a shortest one among such lightest paths, and  $\text{bpleng\_bl}(u, v, b)$  is the length of this length bounded best path. In other words,  $\text{bpleng\_bl}(u, v, b)$  is the length of a shortest path with weight  $\text{dist\_bl}(u, v, b)$ . In the previous argument, we explained a way to compute  $\text{leng}(u, v, b)$ , i.e., the length of a shortest path from  $u$  to  $v$  within length bound  $b$ . Here we can use a similar technique; we only need to modify the algorithm so that it computes the length of a shortest path within length bound  $b$  with weight

$d$ , for  $d = \text{dist\_bl}(u, v, b)$  computed beforehand. In this way, we can compute  $\text{bpleng\_bl}(u, v, b^r)$  in  $\text{poly}(bn)^r$ -time and  $O(r(bn)^{1-\varepsilon})$ -space.

With two functions  $\text{dist\_bl}$  and  $\text{bpleng\_bl}$ , we now explain how to generalize the idea of Barnes et al. The key point is to use the length of a best path to partition  $V$ . For a given parameter  $L$ , we consider a family  $\{V_j\}_{j \in [L-1]}$ , where for each  $j \in [L-1]$ ,  $V_j$  is defined by

$$V_j = \{v \mid \text{bpleng}(s, v) \bmod L = j\}.$$

Again  $\{V_j\}_{j \in [L-1]}$  is a partition of all vertices of  $V$  connected from  $s$ . In particular, each  $V_j$  is a separator of all best paths of length  $\geq j$ . Note again that there must be some  $V_j$  such that  $|V_j| \leq n/L$ . We use one of such  $V_j$ 's, say,  $V_{j_0}$ , to record necessary information to search the lightest (in fact, best) path from  $s$  to all vertices in  $V$ . The information we need to keep for each  $v \in V_{j_0}$  is the weight (which is in fact distance) and the length of a best path from  $s$  to  $v$ . It is easy to see an outline similar to the one explained for **Algo.Betal** works. Here we only explain the procedure for computing  $V_j$  for a given  $j \in [L-1]$ .

```

procedure for computing  $V_j$ 
 $d[s], l[s] \leftarrow 0; D \leftarrow \{s\};$ 
while  $v\_min \neq \perp$  do {
   $v\_min \leftarrow \perp; \text{dist\_min} \leftarrow +\infty;$ 
  for each  $v \in V \setminus D$  do {
     $u\_closest \leftarrow \text{argmin}\{\text{cost\_via}(u, v) \mid u \in D\};$ 
     $(\text{dist}, \text{leng}) \leftarrow \text{cost\_via}(u\_closest, v);$ 
    if  $\text{leng} = L$ 
      and  $\text{dist} < \text{dist\_min}$  then {
         $v\_min \leftarrow v; \text{dist\_min} \leftarrow \text{dist}; \text{leng\_min} \leftarrow \text{leng};$ 
      }
  }
  if  $v\_min \neq \perp$  then {
     $D \leftarrow D \cup \{v\_min\}; d[v] \leftarrow \text{dist\_min}; l[v] \leftarrow \text{leng\_min};$ 
    if  $|D| > n/L$  then report failure and stop;
  }
}
return  $D$  as  $V_j$ ;

```

**Fig. 2.** Procedure for computing  $V_j$  for the stDist problem

The outline of the procedure is the same as before. We use a variable  $D$  to denote the set of vertices in  $V_j$  whose best path from  $s$  has been determined, and for any  $v \in D$ , we use  $d[v]$  and  $l[v]$  to record the weight and length of its best path. For any  $u \in D$  and  $v \in V \setminus D$ , we define a function  $\text{cost\_via}(u, v)$  by

$$\text{cost\_via}(u, v) = (d[u] + \text{dist\_bl}(u, v, L), l[u] + \text{bpleng\_bl}(u, v, L)).$$

That is, “cost” is now a pair of the weight and length of the lightest path from  $s$  to  $v$  going through  $u$  (in  $D$ ). Then for comparing a pair of such costs, we use the lexicographic order; that is, compare weights first (and if they are equal) compare lengths next. For computing  $u\_closest$  in the procedure, we use this comparison.

The key point for showing the correctness of this procedure is stated in the following lemma. Once the lemma is proved, the correctness of the procedure and the whole algorithm follows easily, which we omit in this paper. The lemma can be proved by an induction on  $k$ , which is also not so difficult and omitted here.

**Lemma 4.** *Let  $v_1, v_2, \dots$  be the enumeration of elements of  $V_j$  under the order given by our cost comparison. Then for any  $k \geq 1$ ,  $v_k$  is the  $k$ th vertex that is selected as  $v\_min$  and put into  $D$ . Furthermore, values  $d[v\_min]$  and  $l[v\_min]$  at the point  $v\_min = v_k$  is put into  $D$  are respectively  $dist(s, v_k)$  and  $bpleng(s, v_k)$ .*

Since the other part and the analysis for  $L$  is almost the same, we omit the rest of the argument and state only the result.

**Theorem 2.** *Suppose that there is an algorithm that solves  $UstDist_{lb}$  in polynomial-time and  $O(n^{1-\varepsilon})$ -space. Then for any integer  $r > 0$ , we have an algorithm that solves  $stDist$  in  $O((poly(n))^r)$ -time and  $O(n^{1-(1-1/(r+1))\varepsilon})$ -space.*

Clearly, the sublinear preserving reduction designed for this theorem can be used as a part of a real algorithm. In fact, as explained in [5], we can modify the algorithm given in [3] (i.e., the one for  $B2$  explained in the previous section) to design the one computing both  $dist\_bl(\cdot, \cdot, b^r)$  and  $bpleng\_bl(\cdot, \cdot, b^r)$  in polynomial-time and  $O(n/b^r + r(b + n/k))$ -space (ignoring a  $\log n$  factor) for relatively small (but not necessarily constant)  $r$ , where  $k$  is another algorithm parameter. By using this algorithm and choosing parameters  $b$ ,  $k$ , and  $r$  appropriately as explained in [3] we can derive the following weakly sublinear-space algorithm corresponding to the one for  $stConn$  of Barnes et al.

**Corollary 3.** [5] *There exists a polynomial-time and  $O(n/2^{\sqrt{\log n}})$ -space algorithm for  $stDist$ .*

## 4 Concluding Remarks

We showed a way to relate the sublinear-space computability of  $UstConn_{lb}$  to that of  $stConn$  with almost same saving, which can be regarded as an sublinear-space approximately preserving reduction. We extend this reduction technique for relating the sublinear-space computability of  $stDist$  and  $UstDist_{lb}$ . We then naturally ask whether similar relation holds for any other NL problems; we may even need to develop a framework for discussing directedness and length bound restriction in general. Also it would be interesting if we can give a similar reduction from, say,  $stDist$  to  $stConn$ .

In the context of our sublinear-space computability, we do not have to restrict ourselves to problems in NL. For example, it would be interesting if we can extend our technique for relating problems in  $LogCFL$ , etc.

**Acknowledgements.** The authors would like to thank Dr. Kotaro Nakagawa for his helpful comments on earlier version of this paper. This work is supported in part by the ELC project (MEXT KAKENHI Grant No. 24106008).

## References

1. Asano, T., Doerr, B.: Memory-constrained algorithms for shortest path problem. In: Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCg 2011) (2011)
2. Asano, T., Kirkpatrick, D., Nakagawa, K., Watanabe, O.:  $\tilde{O}(\sqrt{n})$ -space and polynomial-time algorithm for planar directed graph reachability. In: Csuhaj-Varjú, E., Dietzfelbinger, M., Ésik, Z. (eds.) MFCS 2014, Part II. LNCS, vol. 8635, pp. 45–56. Springer, Heidelberg (2014)
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 Structure in Complexity Theory Conference, pp. 27–33. IEEE Computer Society Press (1992)
4. Chakraborty, D., Pavan, A., Tewari, R., Vinodchandran, V., Yang, L.: New time-space upper bounds for directed reachability in high-genus and  $H$ -minor-free graphs. In: ECCG TR14-035 (2014)
5. Imai, T.: Polynomial time memory constrained shortest path algorithms for directed graphs (in Japanese). In Proceedings of 12th Forum for Informatics (FIT 2013), IEICE Japan, RA-002 (2013)
6. Imai, T., Nakagawa, K., Pavan, A., Vinodchandran, N.V., Watanabe, O.: An  $O(n^{\frac{1}{2}+\epsilon})$ -space and polynomial-time algorithm for directed planar reachability. In: Proceedings of the 28th Conference on Computational Complexity (CCC 2013), pp. 277–286. IEEE (2013)
7. Reingold, O.: Undirected connectivity in log-space. J. ACM **55**(4), 1–24 (2008)
8. Wigderson, A.: The complexity of graph connectivity. In: Havel, I.M., Koubek, V. (eds.) MFCS 1992. LNCS, vol. 629, pp. 112–132. Springer, Heidelberg (1992)

SOFSEM 2016: Theory and Practice of Computer  
Science

42nd International Conference on Current Trends in  
Theory and Practice of Computer Science, Harrachov,  
Czech Republic, January 23-28, 2016, Proceedings

Freivalds, R.M.; Engels, G.; Catania, B. (Eds.)

2016, XV, 630 p. 150 illus. in color., Softcover

ISBN: 978-3-662-49191-1