数理解析研究所講究録
第 695 巻 1989 年 169-177

# Topological Sorting の NLOG 完全性について

– The Complexity of Topological Sorting Algorithms –

正代 隆義
Takayoshi Shoudai
Department of Mathematics, Kyushu University

We consider the following problem: Given a directed acyclic graph $G$ and vertices $s$ and $t$, is $s$ ordered before $t$ in the topological order generated by a given topological sorting algorithm? For known algorithms, we show that these problems are log-space complete for NLOG. It also contains the lexicographically first topological sorting problem. The algorithms use the result that NLOG is closed under complementation.

## 1. Introduction

The topological sorting problem is, given a directed acyclic graph $G = (V, E)$, to find a total ordering of its vertices such that if $(v, w)$ is an edge then $v$ is ordered before $w$. It has important applications for analyzing programs and arranging the words in the glossary [6]. Moreover, it is used in designing many efficient sequential algorithms, for example, the maximum flow problem [11].

Some techniques for computing topological orders have been developed. The algorithm by Knuth [6] that computes the lexicographically first topological order runs in time $O(|E|)$. Tarjan [11] also devised an $O(|E|)$ time algorithm by employing the depth-first search method. Dekel, Nassimi and Sahni [4] showed a parallel algorithm using the parallel matrix multiplication technique. Ruzzo also devised a simple $NL^*$-algorithm as is stated in [3]. Hence this problem is in $NC^2$. However, any completeness result does not seem to be known as to the exact complexity of the topological sorting problem.

In this paper, we consider the decision problems for the orders generated by topological sorting algorithms. We show that the problems for the above four algorithms are all $NLOG$-complete. But it is not known whether all possible topological sorting problems are $NLOG$-complete. Immermann [5] and Szelepcsényi [10] showed that $NLOG$ is closed under complementation. By using this result, we can give $NLOG$-alogrithms computing the lexicographically first toplogical sort and other problems. It should be noted that the "lexicographically first" property often makes some problems $P$-complete [7,8,9]. But the topological sorting with this property remains in $NLOG$.

## 2. Topological sort

We classify the known topological sorting algorithms into the following four types. Let $G = (V, E)$ be a directed acyclic graph, where $V = \{1, ..., n\}$ and $|E| = m$.

(K) *Knuth* [6]: For each vertex $v \in V$, we assume an adjacency list $adj[v]$ of vertices linked by an edge from $v$ and an integer $count[v]$ for the number of predecessors of $v$. The vertices whose *count* has been reduced to zero but which have not yet been ordered are stored in a heap [11]. Two operations *insert* and *deletemin* are possible on a heap. The algorithm is as follows. Initially the heap $h$ is empty.

```
begin
  for $v := 1$ to $n$ do
    if $count[v] = 0$ then $insert(v, h)$;
  for $i := 1$ to $n$ do
    begin
      $v := deletemin(h)$; $order[i] := v$
      for each $u \in adj[v]$ do
        begin
          $count[u] := count[u] - 1$;
          if $count[u] = 0$ then $insert(u, h)$;
        end
      end
    end.
```

Since *deletemin* and *insert* take $O(\log n)$ [11] and each operation executes $n$ times, this algorithm has an $O(n \log n + m)$ time bound. This algorithm generates the lexicographically first topological order on $G$. We call this order *lft-order* for short.

(**T**) *Tarjan* [11]: We assume an adjacency list which is ordered lexicographcally. The depth-first serch is executed to make a forest. Then we order the vertices in decreasing order as they are unstacked. This algorithm runs in $O(n+m)$ time.

(**D**) *Dekel, Nassimi and Sahni* [4]: We assume an adjacency matrix. By using matrix multiplication, we can compute the length of the longest path from any vertex with no predecessors to each vertex. Then vertices are sorted in nondecreasing order of their lengths. This is an $NC^2$-algorithm.

(**R**) *Ruzzo* (Cook [3]): We compute the transitive closure of an adjacency matrix. This gives the number of predecessors of each vertex by summing the columns. Then we sort vertices by these numbers.

Generally the problem TS($A$) is defined as follows: Given a directed acyclic graph $G = (V, E)$ and two distinguished vertices $s$ and $t$, determine whether $s$ ordered before $t$ in the order generated by a given topological sorting algorithm $A$. We define the lexicographically first topological sorting problem (LFTS) as TS(**K**).

## 3. Topological sortings are NLOG-hard

We will show that the problem LFTS is $NLOG$-hard. The topological sorting problems for the other algorithms mentioned in Section 2 can also be shown $NLOG$-hard in a similar way.

We identify the vertices with its numbers. For $u, v \in V$, $u \rightarrow^* v$ denote the existence of a path from $u$ to $v$, and $u \rightarrow^+ v$ denote the fact $u \neq v$ and $u \rightarrow^* v$. In addition, $u \prec v$ denote the fact that $u$ is ordered before $v$ in the *lft-order*. For the *lft-order* of any directed acyclic graph, the following lemma holds.

**Lemma 3.1.** *For any $u, v \in V$, $u \prec v$ if and only if either $u \rightarrow^+ v$ or there exists a vertex $x$ such that $u \prec x$, $x \rightarrow^* v$ and $x > u$.*

**Proof.** If $u \prec v$ and $u \not\rightarrow^+ v$, let $x$ be the rightmost vertex in the *order* $\prec$ such that $u \prec x$ and $x \rightarrow^* v$. Its vertex may be $v$. If $x < u$, it contradicts the fact that $\prec$ is the lexicographically first order. Thus $x > u$. Conversely, it is easy to see $u \prec v$ if the condition holds. □

**Corollary 3.2.** *Suppose $u$ is the largest vertex in $V$. Then $u \prec v$ if and only if $u \rightarrow^+ v$.*

**Proof.** Straightforward by Lemma 3.1. □

**Theorem 3.3.** *TS(A) is NLOG-hard for $A =$ **K**, **T**, **D**, **R**.*

**Proof.** The monotone graph accessibility problem (MGAP) is described as follows: Given a directed acyclic graph $G$ and vertices $s$ and $t$, then determine whether $t$ is reachable from $s$. It is known that MGAP is *NLOG*-complete. We first give a reduction from MGAP to LFTS (TS(**K**)). For an instance $(G, s, t)$ of MGAP, we renumber the vertices so that $s$ has the largest number of $V$. If there exists a path from $s$ to $t$, $s$ must be ordered before $t$ in any topological order. Conversely if $s$ is ordered before $t$ in the *lft-order*, Corollary 3.2 implies that there exists a path from $s$ to $t$. It is clear that this reduction is computable in log-space.

Reductions from MGAP to the other problems are similar. For TS(**T**), we renumber vertices so that $s$ has the smallest number. Then $s$ is the root of the first tree in the depth-first forest. If $t$ is reachable from $s$, this tree contains $t$. Otherwise, $t$ is in one of the other trees. For TS(**D**) and TS(**R**), we attach a new directed $n$-chain by identifying the sink of the $n$-chain with the vertex $s$ where $n = |V|$. Then $s$ has the larger key than that of any vertex which is not reachable from $s$. □

## 4. NLOG-algorithms for the TS problems

In this section, we will present an $NLOG$-algorithm for LFTS (TS(**K**)) and describe $NLOG$-algorithms for TS($A$) ($A$ = **T**, **D**, **R**). The class $NLOG$ is closed under complementation [5,10]. Hence $NLOG$ contains the complement of MGAP (co-MGAP). We use the MGAP and its complement to see reachabilities between vertices.

For $u \in V$, $R(u)$ denote the set $\{v \in V | v \rightarrow^* u\}$. We consider a sequence $u_1, u_2, \cdots, u_p$ of vertices defined as follows:

$$\begin{aligned} u_1 &:= \max R(s) \cap R(t). \\ u_i &:= \max\{v \in R(s) \cap R(t) | u_{i-1} \rightarrow^+ v\} \ (i = 2, 3, \cdots). \end{aligned}$$

We assume that the set function max returns zero if the argument is empty. Let $u_p$ be the last vertex whose value is positive.

**Lemma 4.1.** *For any $u \in R(s) \cap R(t)$, $u_i \prec u$ if and only if $u_i \rightarrow^+ u$ ($i = 1, 2, \cdots, p$).*

**Proof.** The proof is by induction on $i$. Suppose $u_1 \prec u$ for $u \in R(s) \cap R(t)$. If $u_1 \not\rightarrow^+ u$, there exists a vertex $x$ in $R(s) \cap R(t)$ which is larger than $u_1$ by Lemma 3.1. This contradicts the definition of $u_1$. Thus $u_1 \rightarrow^+ u$. Suppose $u_i \prec u$. If $u_i \not\rightarrow^+ u$, there exists a vertex $x$ such that $u_i \prec x$, $x \rightarrow^* u$ and $x > u_i$. So $x \in R(s) \cap R(t)$. Therefore $u_{i-1} \prec u_i$ and $u_i \prec x$ imply $u_{i-1} \prec x$. By the induction hypothesis, $u_{i-1} \rightarrow^+ x$. This contradicts the definition of $u_i$. □

**Corollary 4.2.** *If $u_p \prec w$, then $w \notin R(s) \cap R(t)$.*

**Proof.** Straightforward by Lemma 4.1. □

**Lemma 4.3.** *We assume that there exists $i$ ($1 \leq i \leq p$) such that $w' < u_i$ for any $w' \in R(s) - R(t)$. Then for any $w \in R(s) - R(t)$, $u_i \prec w$ if and only if $u_i \rightarrow^+ w$.*

**Proof.** If $u_i \prec w$ and $u_i \not\rightarrow^+ w$, then there exists $x$ which satisfies $u_i \prec x$ and $x > u_i$. Thus, by the assumption, $x \in R(s) \cap R(t)$. This contradicts the definition of $u_i$ since $u_i \rightarrow^+ x$ by Corollary 4.2. □

**Lemma 4.4.** *For $w \in R(s) - R(t)$, if either $w > u_1$ or there exists $i$ $(2 \leq i \leq p)$ such that $u_{i-1} \prec w$ and $w > u_i$, then $u_p \prec w$.*

**Proof.** Suppose $w \prec u_p$. Since $w \not\to^+ u_p$, there exists a vertex $x$ which is in $R(s) \cap R(t)$. This fact and $u_{i-1} \prec w$ (if $i \geq 2$) imply $u_i \prec x$. This contradicts the definition of $u_i$. □

From now on, we assume that there does not exist a path between $s$ and $t$. Let $v_s = \max\{v \in R(s) - R(t) | u_p \prec v\}$ and $v_t = \max\{v \in R(t) - R(s) | u_p \prec v\}$. (If $R(s) \cap R(t) = \phi$, let $v_s = \max R(s)$ and $v_t = \max R(t)$.)

**Lemma 4.5.** *$v_s < v_t$ if and only if $s \prec t$.*

**Proof.** Suppose $v_t \prec s$. By Lemma 3.1, there exists a vertex $x$ such that $v_t \prec x$, $x \to^* s$ and $x > v_t$. Since $u_p \prec v_t$, $u_p \prec x$. Thus $x \in R(s) - R(t)$ by Corollary 4.2 and $x \to^* s$. It contradicts the fact $v_s < v_t$. So $s \prec v_t$, thus, $s \prec t$. Conversely, if $s \prec t$, then $v_s \prec t$. By Lemma 3.1, there exists a vertex which is larger than $v_s$. Thus $v_s < v_t$. □

When we can verify that there exists $i$ $(2 \leq i \leq p)$ such that $u_{i-1} > w$ for any $w \in R(s) - R(t)$, if $u_{i-1} \to^+ w$ and $u_i < w$, then $u_p \prec w$ by Lemma 4.3 and Lemma 4.4. Therefore, under this assumption, $v_s$ equals to $\max\{v \in R(s) - R(t) | u_{i-1} \to^+ v\}$ if this value is larger than $u_i$.

**Theorem 4.4.** *LFTS is NLOG-complete.*

**Proof.** The following algorithm can decide in NLOG whether $s \prec t$. We can decide the reachability and unreachability by using the MGAP and co-MGAP.

```
begin
    if $s \to^+ t$ then accept
    else if $t \to^+ s$ then reject;
    $v_s := \max R(s) - R(t)$; $v_t := \max R(t) - R(s)$; $u := \max R(s) \cap R(t)$;
    while true do
    begin
        if $u < v_s$ or $u < v_t$ then
            if $v_s < v_t$ then accept else reject;
        $v_s := \max\{v \in R(s) - R(t) | u \to^+ v\}$;
        $v_t := \max\{v \in R(t) - R(s) | u \to^+ v\}$;
        $u := \max\{v \in R(s) \cap R(t) | u \to^+ v\}$;
    end
end.
```

The variable $u$ is stored the value $u_i$ ($i = 1, 2, \cdots$). It is easy to see that this algorithm correctly decide the order between $s$ and $t$ by Lemma 4.3, Lemma 4.4 and Lemma 4.5. The reachability between vertices is computed in $NLOG$ by enumerating all vertices and employing the MGAP and co-MGAP. □

TS(**R**) is also $NLOG$-complete since the number of predecessors can be computed in nondeterministic log-space by using Reach.

For computing the length of the longest path, we use the following problem: Given a directed acyclic graph $G$, vertices $s$ and $t$ and a nonnegative integer $l$, decide whether there exists a path whose length is longer then $l$. This problem is solved in $NLOG$. By employing it and its complement, we can construct the function for computing the length of the longest path. This implies an $NLOG$-algorithm for TS(**D**).

TS(**T**) uses the depth-first search method, but this seaching is not known to be in $NLOG$ [1]. However, we can solve TS(**T**) in $NLOG$ for a directed acyclic graph. For any vertex $v$, the root of the tree which contains $v$ is the smallest vertex such that $v$ is reachable from it. This root can be found in $NLOG$ by using Reach. We compute the roots for $s$ and $t$, respectively. Then the root of $s$ is larger than that of $t$ if and only if $s$ is ordered before $t$. If they are identical, we find the new root for

$s$ (resp. $t$) such that it is one of the children of the current root and the subtree at the new root contains $s$ (resp. $t$). Then we compare them. These roots also can be found in $NLOG$. We iterate this process until they are not identical.

## 5. Concluding Remarks

Cook [3] illustrates a taxonomy of problems in parallel computations. The class $NLOG$ locates between $NC^1$ and $NC^2$. The sorting problem is solvable by circuits of depth $O(\log n)$ with polynomial gates [2]. On the other hand, we have shown that the known topological sorting problems are all $NLOG$-complete. Since it seems that $NLOG$ differs from $NC^1$, there may exist no $NC^1$-algorithms computing one of the four types topological sorting problems. But it is an open question whether there is an $NC^1$ topological sorting algorithm.

**Acknowledgement.** The author would like to thank Prof. S. Miyano for many helpful discussions and much thoughtful criticism.

## 参考文献

[1] A. Aggarwal and R.J. Anderson, A random NC algorithm for depth first search, Proc. 19th ACM Symp. Theory of Comput. (1987) 325-334.

[2] M. Ajtai, J. Komlós and E. Szemerédi, An $O(n \log n)$ sorting network, Proc. 15th ACM Symp. Theory of Comput. (1983) 1-9.

[3] S.A. Cook, A taxonomy of problems with fast parallel algorithms, *Inform. Contr.* **64** (1985) 2-22.

[4] E. Dekel, D. Nassimi and S. Sahni, Parallel matrix and graph algorithms, *SIAM J. Comput.* **10** (4) (1981) 657-675.

[5] N. Immermann, Nondeterministic space is closed under complement, Technical Report, Department of Computer Sicence, Yale University, July (1987).

[6] D. Knuth, "The Art of Computer Programming," Vol.1, Addison-Wesley, Reading, Mass. (1968).

[7] S. Miyano, The lexicographically first maximal subgraph problems: P-completeness and NC algorithms, Proc. 13th ICALP (Lecture Notes in Comuter Science **267**) (1987) 425-434.

[8] S. Miyano. A parallelizable lexicographicaly first maximal edge-induced subgraph problem, *Inform. Process. Lett.* **27** (1988) 75-78.

[9] J.H. Reif, Depth-first search is inherently sequential, *Inform. Process. Lett.* **20** (1985) 229-234.

[10] R. Szelepcsényi, The method of forcing for nondeterministic automata, *Bulletin of the EATCS* **33** (1987) 96-100.

[11] R.E. Tarjan, "Data Structures and Network Algorithms," CBMS-NSF Regional Conference Series in Applied Mathematics **44**, SIAM, Philadelphia, Pennsylvania (1983).