A Tight Analysis of the Katriel-Bodlaender Algorithm for Online Topological Ordering

Hsiao-Fei Liu[1] and Kun-Mao Chao[1,2,*]
[1]Department of Computer Science and Information Engineering
[2]Graduate Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan 106

August 24, 2007

Abstract

Katriel and Bodlaender [7] modify the algorithm proposed by Alpern et al. [2] for maintaining the topological order of the n nodes of a directed acyclic graph while inserting m edges and prove that their algorithm runs in $O(\min\{m^{3/2}\log n, m^{3/2} + n^2 \log n\})$ time and has an $\Omega(m^{3/2})$ lower bound. In this paper, we give a tight analysis of their algorithm by showing that it runs in time $\Theta(m^{3/2} + mn^{1/2}\log n)$[1].

General Terms: Algorithms
Additional Key Words and Phrases: Topological Order, Online Algorithms, Tight Analysis

1 Introduction

A topological order *ord* of a directed acyclic graph (DAG) $G = (V, E)$ is a linear order of all its vertices such that if G contains an edge (u, v), then $ord(u) < ord(v)$. In this paper we study an online variant of the topological ordering problem in which the edges of the DAG are given one at a time and we have to update the order *ord* each time an edge is added.

When dealing with DAGs, the topological order of vertices often provides very crucial information for further algorithm development. Thus online topological ordering is of interests because it is very likely to be required when one has to develop online algorithms on DAGs. For example, the online topological ordering has appeared in the following contexts.

- Incremental Evaluation of Computational Circuits [2].

*corresponding author, kmchao@csie.ntu.edu.tw

[1]In this paper, we assume $m = \Omega(n)$. In fact, our analysis can be easily extended to prove that the algorithm runs in time $\Theta(\min\{m^{3/2} + mn^{1/2}\log n, m^{3/2}\log m + n\})$ without the assumption $m = \Omega(n)$.

- Incremental Compilation [8, 11], where dependencies between modules are maintained to reduce the amount of recompilation performed when an update occurs.
- Local Search [10]. Local search is one of the main approaches to combinatorial optimization and often requires sophisticated incremental algorithms.
- Online Computation of Strongly Connected Components [12].
- Online Cycle Detection [7, 12, 13]. Currently the best online cycle detection algorithm for sparse directed graphs is built upon the Katriel-Bodlaender algorithm and has the same complexity as the Katriel-Bodlaender algorithm. Thus our analysis improves the upper bound of the online cycle detection problem to $O(m^{3/2} + mn^{1/2} \log n)$.
- Source Code Analysis [12, 13], where the aim is to determine the target set for all pointer variables in a program, without executing it.

Alpern et al. [2] give an algorithm which takes $O(||\delta|| \log ||\delta||)$ time for each edge insertion, where $||\delta||$ measures the number of edges and nodes of a minimal subgraph that needs to be updated. (For a formal definition of $||\delta||$, please see [2, 14, 15].) Pearce and Kelly [14] propose a different algorithm which needs slightly more time to process an edge insertion in the worst case than the algorithm given by Alpern et al. [2], but show experimentally their algorithm perform well on sparse graphs.

Marchetti-Spaccamela et al. [9] give an algorithm which takes $O(mn)$ time for inserting m edges. Katriel [6] shows that the analysis is tight. Recently, Katriel and Bodlaender [7] modify the algorithm proposed by Alpern et al. [2], which is referred to as the Katriel-Bodlaender algorithm in this paper. They prove that their algorithm has both an $O(\min\{m^{3/2} \log n, m^{3/2} + n^2 \log n\})$ upper bound and an $\Omega(m^{3/2})$ lower bound on runtime for m edge insertions. This is the best amortized result for sparse graphs so far. They also analyze the complexity of their algorithm on structured graphs. They show that it runs in time $O(mk \log^2 n)$ where k is the treewidth of the underlying undirected graph and can be implemented to run in $O(n \log n)$ time on trees. On the other hand, Ajwani et al. [1] proposed an $O(n^{2.75})$-time algorithm, independent of the number of edges inserted. This is the best amortized result for dense graphs so far.

In this paper, we prove that the Katriel-Bodlaender algorithm takes $\Theta(m^{3/2}+mn^{1/2} \log n)$ time for inserting m edges. By combining this with Ajwani et al.'s result [1], we get an upper bound of $O(\min\{m^{3/2} + mn^{1/2} \log n, n^{2.75}\})$ for online topological ordering. It is an improvement over the previous best upper bound of $O(\min\{m^{3/2}\log n, m^{3/2} + n^2\log n, n^{2.75}\})$.

The rest of this paper is organized as follows. In Section 2, we describe how the Katriel-Bodlaender algorithm works, define notation and introduce some theorems proved in [7]. Section 3 proves that the Katriel-Bodlaender algorithm runs in $O(m^{3/2}+mn^{1/2}\log n)$ time, and Section 4 shows it needs $\Omega(m^{3/2} + mn^{1/2}\log n)$ time. Since the upper bound matches the lower bound, our analysis is tight. Section 5 summarizes our results and discusses future work.

2 The Katriel-Bodlaender Algorithm

The pseudo code of the Katriel-Bodlaender algorithm is given in Figure 1.[2] The algorithm works as follows. The topological order of nodes is maintained by an *order data structure* ORD, which can maintain a total order and support the following operations in constant time [4, 3]:

- $InsertAfter(x, y)$ ($InsertBefore(x, y)$): Inserts x immediately after (before) y in the total order.
- $Delete(x)$: Removes x from the total order.
- $>_{ord}(x, y)$: Returns true if and only if x follows y in the total order.
- $Next(x)$ ($Prev(x)$): Returns the element that appears immediately after (before) x in the total order.

Initially the nodes are inserted into ORD in an arbitrary order. Each time a new edge $(Source, Target)$ arrives, $AddEdge(Source, Target)$ is called to insert the edge $(Source, Target)$ into the graph and update the total order maintained by ORD to a valid topological order for the modified graph.

It remains to describe how $AddEdge(Source, Target)$ operates. In each iteration of the first while loop, there is one node s which is a candidate for insertion into stack ToS (the node with maximal rank in the current topological order which reaches a node in ToS but is not in ToS) and one node t which is a candidate for insertion into stack $FromT$ (the node with minimal rank in the current topological order which can be reached from a node in $FromT$ but is not in $FromT$). The algorithm always adds at least one of them into the

[2]For the sake of exposition, we slightly modify the way Katriel and Bodlaender present their algorithm. The only nontrivial modifications are the conditions in lines 8 and 14. However, by Lemma 2.3 in [7], one can verify that the conditions are equivalent to the ones in [7].

relevant set. The way in which it decides which candidate(s) to add aims to balanced the number of edges outgoing from nodes in $FromT$ and the number of edges entering into nodes in ToS. That is, the algorithm always chooses a candidate so that the increase of $\max\{\sum_{v \in ToS} Indegree[v], \sum_{v \in FromeT} Outdegree[v]\}$ will be fewer after adding the candidate into its relevant set. If a tie occurs, then both s and t will be added into their relevant sets. If s is added to its relevant set ToS, all nodes which can reach s by one edge will be inserted into $ToSNeighbors$ and then s will be reset to the max element in $ToSNeighbors$. $ToSNeighbors$ is a priority queue maintaining all nodes which can reach nodes in ToS by one edges but is not in ToS. $ToSNeighbors$ is implemented by Fibonacci heaps [5] which can support insertions and extractions in $O(1)$ and $O(\log n)$ amortized time respectively. $ToSNeighbors$ determine the ranks of its elements according to the total order maintained by ORD. Similarly, if t is added to $FromT$, then all nodes which are reachable from t by one edge will be inserted into $FromTNeighbors$ and then t will be reset to the min element in $FromTNeighbors$. $FromTNeighbors$ is a priority queue maintaining all nodes which can be reached from nodes in $FromT$ by one edge but is not in $FromT$. $FromTNeighbors$ is also implemented by Fibonacci heaps and determine the ranks of its elements according to the total order maintained by ORD. The first while loop stops when $t >_{ord} s$ or any one of $ToSNeighbors$ and $FromTNeighbors$ is empty. If $ToSNeighbors$ ($FromTNeighbors$) is empty when the first while loop stops then s (t) will be reset to $ORD.Prev(Target)$ ($ORD.Next(Source)$) before we update ORD. The update of ORD is carried out by fulfilling the following two tasks. First, delete all nodes in ToS from ORD and then insert them, in the same relative order among themselves, immediately after s. Secondly, delete all nodes in $FromT$ from ORD and then insert them, in the same relative order among themselves, immediately before t. After the update of ORD, the edge $(Source, Target)$ is inserted into the graph.

In the following, we shall define some notation. Let n and m be the number of nodes and edges in the DAG $G = (V, E)$ respectively. Let $G_i = (V, E_i)$ be the graph after the i^{th} edge insertion. Let $Indegree_i[v]$ ($Outdegree_i[v]$) be the indegree (outdegree) of v in G_i. Let $FromT_i$ (ToS_i) denote the set of nodes in the stack $FromT$ (ToS) at the end of the first while loop upon the insertion of the i^{th} edge. Let s_i (t_i) denote the value of the variable s (t) at the end of the first while loop upon the insertion of the i^{th} edge. Let $T_i = \sum_{v \in FromT_i} Outdegree_{i-1}[v]$ and $S_i = \sum_{v \in ToS_i} Indegree_{i-1}[v]$. Let x_i denote $\max\{T_i, S_i\}$ and y_i denote $\max\{|ToS_i|, |FromT_i|\}$. Let $>_{ord_i}$ be the total order maintained by ORD after the i^{th} edge insertion. The following three theorems are proved in [7].

Theorem 1: The Katriel-Bodlaender algorithm needs $O(m^{3/2} + \sum_{1 \le i \le m} y_i \log n)$ time to insert m edges into an initially empty n-node graph.

Theorem 2: The Katriel-Bodlaender algorithm needs $\Omega(m^{3/2})$ time to insert m edges into an initially empty n-node graph.

Theorem 3: $Indegree_{i-1}[s_i] + S_i \ge x_i$ and $Outegree_{i-1}[t_i] + T_i \ge x_i$, for all i in $[1, m]$.

3 The $O(m^{3/2} + mn^{1/2} \log n)$ Upper Bound

In this section, we shall prove that the algorithm runs in time $O(m^{3/2} + mn^{1/2} \log n)$. By Theorem 1, we know the algorithm runs in time $O(m^{3/2} + \sum_{1 \le i \le m} y_i \log n)$, so we only have to show that $\sum_{1 \le i \le m} y_i$ is $O(mn^{1/2})$. For simplicity, we assume that $x_i \ge y_i$ for all i in $[1, m]$, although it should be $x_i \ge y_i - 1$ for all i in $[1, m]$.

An edge $e = (u, v)$ is called to be in front of (behind) a node w in G_i if and only if there is a path from v (w) to w (u) in G_i. A pair $(e, w) \in E \times V$ is called to be ordered in G_i if and only if e is either in front of or behind w in G_i. In the following proofs, we adopt one of the potential functions defined in [7]: The number of ordered pairs in $E \times V$. Let Φ_i denote the set $\{(e, w) \in E \times V |\ (e, w)$ is ordered in $G_i\}$, ϕ_i denote $|\Phi_i|$ and $\triangle\phi_i$ denote $\phi_i - \phi_{i-1}$.

Lemma 4: For all edges e incoming into ToS_i in G_{i-1} and for all nodes w in $FromT_i$, e is not in front of w in G_{i-1}.

Proof: Let $e = (u, v)$. Suppose for the contradiction that there is a path from v to w in G_{i-1}. It implies that $w >_{ord_{i-1}} v$. There are three cases to consider. Case 1: The iteration in which variable s was assigned v is before the iteration in which variable t was assigned w in the i^{th} call of $AddEdge$. Since the nodes were assigned to variable s in decreasing order, we had $t >_{ord_{i-1}} s$ after variable t was assigned w and then left the loop. It contradicts to the assumption that w is in $FromT_i$. Case 2: The iteration in which variable t was assigned w is before the iteration in which variable s was assigned v in the i^{th} call of $AddEdge$. Since the nodes were assigned to variable t in increasing

```
Function AddEdge(Source, Target)
ToS ← []; FromT ← [];
ToSNeighbors ← []; FromTNeighbors ← [];
ToSIndegree ← 0; FromTOutdegree ← 0;
s ← Source; t ← Target;
while s >_ord t and s ≠ nil and t ≠ nil do
    m_s ← ToSIndegree; ℓ_s ← Indegree[s];
    m_t ← FromTOutdegree; ℓ_t ← Outdegree[t];
    if m_s + ℓ_s ≤ m_t + ℓ_t then
        ToS.Push(s);
        foreach (w, s) ∈ E do ToSNeighbors.Insert(w);
        ToSIndegree ← ToSIndegree + Indegree[s];
        s ← ToSNeighbors.ExtractMax;
    end if
    if m_s + ℓ_s ≥ m_t + ℓ_t then
        FromT.Push(t);
        foreach (t, w) ∈ E do FromTNeighbors.Insert(w);
        FromTOutdegree ← FromTOutdegree + Outdegree[t];
        t ← FromTNeighbors.ExtractMin;
    end if
end while
if s = nil then s ← ORD.Prev(Target);
if t = nil then t ← ORD.Next(Source);
while ToS.NotEmpty do
    s′ ← ToS.Pop;
    ORD.Delete(s′); ORD.InsertAfter(s′, s); s ← s′;
end while
while FromT.NotEmpty do
    t′ ← FromT.Pop;
    ORD.Delete(t′); ORD.InsertBefore(t′, t); t ← t′;
end while
E ← E ∪ (Source, Target); Outdegeree[Source]++; Indegree[Target]++;
```

Figure 1: The algorithm proposed by Katriel and Bodlaender [7].

order, we had $t >_{ord_{i-1}} s$ after the variable s was assigned v and then left the loop. It contradicts to the assumption that v is in ToS_i. Case 3: Variable s and variable t was assigned v and w respectively at the same iteration in the i^{th} call of *AddEdge*. Since $w >_{ord_{i-1}} v$, we had $t >_{ord_{i-1}} s$ after variable t was assigned w and then left the loop. It contradicts to the assumption that w is in $FromT_i$ and v is in ToS_i. □

Lemma 4 states that all the S_i edges incoming into ToS_i are not in front of $FromT_i$ in G_{i-1}. Because all these S_i edges became in front of $FromT_i$ after the i^{th} edge insertion, we know $\triangle\phi_i \geq S_i \times |FromT_i|$. To pave the way for proving Lemma 8, we have to show $y_i^2 \leq \triangle\phi_i$ when $y_i = |FromT_i|$. If S_i was always larger than or equal to y_i when $y_i = |FromT_i|$, then we could jump to prove Lemma 8 directly. Since it is not the case, we need more lemmas. There are two cases to consider: First, $w <_{ord_{i-1}} s_i$ for all w in $FromT_i$, i.e., s_i is after $FromT_i$ in the total order $<_{ord_{i-1}}$. Second, some nodes in $FromT_i$ are after s_i in the total order $<_{ord_{i-1}}$. The following lemma deals with the first case.

Lemma 5: If $w <_{ord_{i-1}} s_i$ for all w in $FromT_i$ and $y_i = |FromT_i|$, then $y_i^2 \leq \triangle\phi_i$.

Proof: Since $w <_{ord_{i-1}} s_i$ for all w in $FromT_i$, we can deduce that all edges incident to s_i in G_{i-1} are not in front of w in G_{i-1} for all w in $FromT_i$. By combining this result with Lemma 4, we know there are at least $Indegree_{i-1}[s_i] + S_i$ edges not in front of w in G_{i-1} for all w in $FromT_i$. Because all these $Indegree_{i-1}[s_i] + S_i$ edges are in front of w in G_i for all w in $FromT_i$ and $y_i = |FromT_i|$, we can deduce that $(Indegree_{i-1}[s_i] + S_i) \times y_i \leq \triangle\phi_i$. By Theorem 3 and the assumption $y_i \leq x_i$, we have $y_i^2 \leq x_i \times y_i \leq (Indegree_{i-1}[s_i] + S_i) \times y_i \leq \triangle\phi_i$. □

The following lemma is used in the proof of Lemma 7 which deals with the second case.

Lemma 6: If there exists w in $FromT_i$ such that $w >_{ord_{i-1}} s_i$, then, in the i^{th} call of *AddEdge*, the iteration in which variable t was assigned t_i is not after the iteration in which variable s was assigned s_i.

Proof: Suppose for the contradiction that the iteration in which variable s was assigned s_i is before the iteration in which variable t was assigned t_i. Let $\hat{t}$ be the last element pushed into $FromT_i$. Consider the iteration in which variable t was assigned t_i. At the beginning, the value of variable s was s_i and the value of variable t was $\hat{t}$. Since $\hat{t} >_{ord_{i-1}} s_i$, we failed in the test condition and left the loop. Thus, $\hat{t}$ was not pushed into $FromT_i$, a contradiction. □

Lemma 7: If there exists w in $FromT_i$ such that $w >_{ord_{i-1}} s_i$ and $y_i = |FromT_i|$, then $y_i^2 \leq \triangle\phi_i$.

Proof: Consider the iteration in which variable t was assigned value t_i in the i^{th} call of *AddEdge*. The value of $m_t + \ell_t$ was equal to T_i. By Lemma 6, we know the value of variable s was not s_i when line 6 was executed, so $m_s + \ell_s \leq S_i$. Since variable t was selected to be assigned a new value, we know $m_t + \ell_t \leq m_s + \ell_s$. By combining the results above, we get $T_i \leq S_i$. It implies that $S_i = x_i$. By Lemma 4, we know there are at least $S_i = x_i$ edges not in front of w in G_{i-1} for all w in $FromT_i$. Because all these x_i edges are in front of w in G_i for all w in $FromT_i$ and $y_i = |FromT_i|$, we can deduce that $x_i y_i \leq \triangle\phi_i$. By the assumption $x_i \geq y_i$, we have $y_i^2 \leq \triangle\phi_i$. □

Lemma 8: $\sum_{y_i=|FromT_i|} y_i^2 \leq mn$.

Proof: By Lemma 5 and Lemma 7, we know $y_i^2 \leq \triangle\phi_i$ if $y_i = |FromT_i|$. Since $\phi_0 = 0$, $\phi_m \leq mn$, $\triangle\phi_i \geq 0$, and $y_i^2 \leq \triangle\phi_i$ if $y_i = |FromT_i|$, we can deduce that $\sum_{y_i=|FromT_i|} y_i^2 \leq \sum_{1\leq i\leq m} \triangle\phi_i \leq mn$. □

The following lemma can be proved by an argument similar to the one for proving Lemma 8.

Lemma 9: $\sum_{y_i=|ToS_i|} y_i^2 \leq mn$.

Theorem 10: $\sum_{1\leq i\leq m} y_i$ is $O(mn^{1/2})$.

Proof: By Lemma 8 and Lemma 9, we know $\sum_{1\leq i\leq m} y_i^2 \leq 2mn$. Since $\sum_{y_i<n^{1/2}} y_i \leq mn^{1/2}$, we only have to show $\sum_{y_i\geq n^{1/2}} y_i$ is $O(mn^{1/2})$. Since $n^{1/2}\sum_{y_i\geq n^{1/2}} y_i \leq \sum_{y_i\geq n^{1/2}} y_i^2 \leq \sum_{1\leq i\leq m} y_i^2 \leq 2mn$, we have $\sum_{y_i\geq n^{1/2}} y_i \leq 2mn^{1/2} = O(mn^{1/2})$. □

Theorem 11: The Katriel-Bodlaender algorithm needs $O(m^{3/2} + mn^{1/2}\log n)$ time to insert m edges into an initially empty n-node graph.

Proof: Theorem 1 states that the Katriel-Bodlaender algorithm needs $O(m^{3/2} + \sum_{1\leq i\leq m} y_i \log n)$ time to insert m edges into an initially empty n-node graph. Theorem 10 states that $\sum_{1\leq i\leq m} y_i$ is $O(mn^{1/2})$. By combining these two results, we know that the Katriel-Bodlaender algorithm needs $O(m^{3/2} + mn^{1/2}\log n)$ time to insert m edges into an initially empty n-node graph. □

4 The $\Omega(m^{3/2} + mn^{1/2} \log n)$ Lower Bound

In this section, we shall prove that the algorithm runs in time $\Omega(m^{3/2} + mn^{1/2} \log n)$.

Theorem 12: The Katriel-Bodlaender algorithm needs $\Omega(m^{3/2} + mn^{1/2} \log n)$ time to insert m edges into an initially empty n-node graph.

Proof: It is equivalent to show that the algorithm needs $\Omega(\max\{m^{3/2} + mn^{1/2} \log n\})$ time to insert m edges into an initially empty n-node graph. Theorem 2 states that the algorithm needs $\Omega(m^{3/2})$ time to insert m edges into an initially empty n-node graph. Since $m^{3/2} \geq mn^{1/2} \log n$ if and only if $m \geq n \log^2 n$, we only have to show that the algorithm needs $\Omega(mn^{1/2} \log n)$ time if $m \leq n \log^2 n$. Without loss of generality we assume that $m \geq n$. In the following, we describe an input which takes the algorithm $\Omega(mn^{1/2} \log n)$ time to process if $n \leq m \leq n \log^2 n$. For simplicity, we assume that both n and m are exact powers of 16.

Let $\{v_0, v_2, \ldots, v_{n-1}\}$ be the nodes of the DAG sorted by the order maintained by ORD before edge insertions. Let $u_i = v_{\frac{n}{4}+i}$ for $i = 0, \ldots, \frac{3n}{4} - 1$. Define P_i to be $(v_{\frac{(i-1)n^{1/2}}{4}}, v_{\frac{(i-1)n^{1/2}}{4}+1}, \ldots, v_{\frac{in^{1/2}}{4}-1})$, for $i = 1, \ldots, n^{1/2}$. Define Q_i to be $(u_{(i-1)n^{1/4}}, u_{(i-1)n^{1/4}+1}, \ldots, u_{in^{1/4}-1})$, for $i = 1, \ldots, \frac{m}{2n^{1/2}}$. Let $Source_i = u_{in^{1/4}-1}$, i.e., the last node of Q_i, for $i = 1, \ldots, \frac{m}{2n^{1/2}}$. Let $Target_i = v_{\frac{(i-1)n^{1/2}}{4}}$, i.e., the first node of P_i, for $i = 1, \ldots, n^{1/2}$.

The input is composed of four parts.

Part 1. Construct $n^{1/2}$ identical subgraphs as in Figure 2(a) by inserting the edge $(v_{\frac{(i-1)n^{1/2}}{4}}, v_{\frac{(i-1)n^{1/2}}{4}+j})$ for all $i = 1, \ldots, n^{1/2}$ and $j = 1, \ldots, \frac{n^{1/2}}{4} - 1$. There are $n/4 - n^{1/2} < n/4 \leq m/4$ edge insertions in this part.

Part 2. Construct $\frac{m}{2n^{1/2}}$ identical subgraphs as in Figure 2(b) by inserting the edge $(u_{(i-1)n^{1/4}+j}, u_k)$ for all $i \in [1, \frac{m}{2n^{1/2}}]$, $j \in [0, n^{1/4}-2]$ and $k \in ((i-1)n^{1/4}+j, in^{1/4})$. There are $\frac{m}{2n^{1/2}} \times \frac{n^{1/2}-n^{1/4}}{2} < m/4$ edge insertions in this part.

Part 3. This part is composed of $n^{1/2}$ rounds and there are $\frac{m}{2n^{1/2}}$ edge insertions in each round. In the i^{th} round, the following edges are inserted in their listed order: $\{(Source_1, Target_i)$, $(Source_2, Target_i)$,. . .,$(Source_{\frac{m}{2n^{1/2}}}, Target_i)\}$. There are $m/2$ edge insertions in this part. Figure 2(c) illustrates how the total order maintained by ORD will change when Part 3 arrives.

Part 4. Insert edges without causing cycles until there are m edges in the DAG.

Upon the insertion of edge $(Source_i, Target_j)$ in Part 3 for all i and j, all nodes in P_j will be inserted into $FromTNeighbors$ at the same iteration and then extracted. Since there are $n^{1/2}/4$ nodes in P_j for all j, each edge insertion in Part 3 takes the algorithm $\Omega(n^{1/2} \log n)$ time to process. Because there are $m/2$ edges in Part 3, the total complexity is $\Omega(mn^{1/2} \log n)$. □

Theorem 13: The Katriel-Bodlaender algorithm needs $\Theta(m^{3/2} + mn^{1/2} \log n)$ time to insert m edges into an initially empty n-node graph..

Proof: It follows directly from Theorem 11 and Theorem 12. □

5 Concluding Remarks

We give a tight analysis of the Katriel-Bodlaender algorithm by proving that it runs in $\Theta(m^{3/2} + mn^{1/2} \log n)$ time. By combining this with the result in [1], we get an upper bound of $O(\min\{m^{3/2} + mn^{1/2} \log n, n^{2.75}\})$ for online topological ordering. It is an improvement upon the previous best upper bound of $O(\min\{m^{3/2} \log n, m^{3/2} + n^2 \log n, n^{2.75}\})$. The only non-trivial lower bound is due to Ramalingam and Reps [15], who show that any algorithm need $\Omega(n \log n)$ time while inserting $n - 1$ edges in the worst case if all labels are maintained explicitly. Bridging the large gap between the lower bound and the upper bound remains an open problem.

Acknowledgments. We thank Irit Katriel for reading our manuscript carefully and providing numerous valuable comments. We thank Chia-Jung Chang for verifying our proof.

References

[1] Deepak Ajwani, Tobias Friedrich, and Ulrich Meyer. An $O(n^{2.75})$ Algorithm for Online Topological Ordering. In *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT)*, 53-64, 2006.

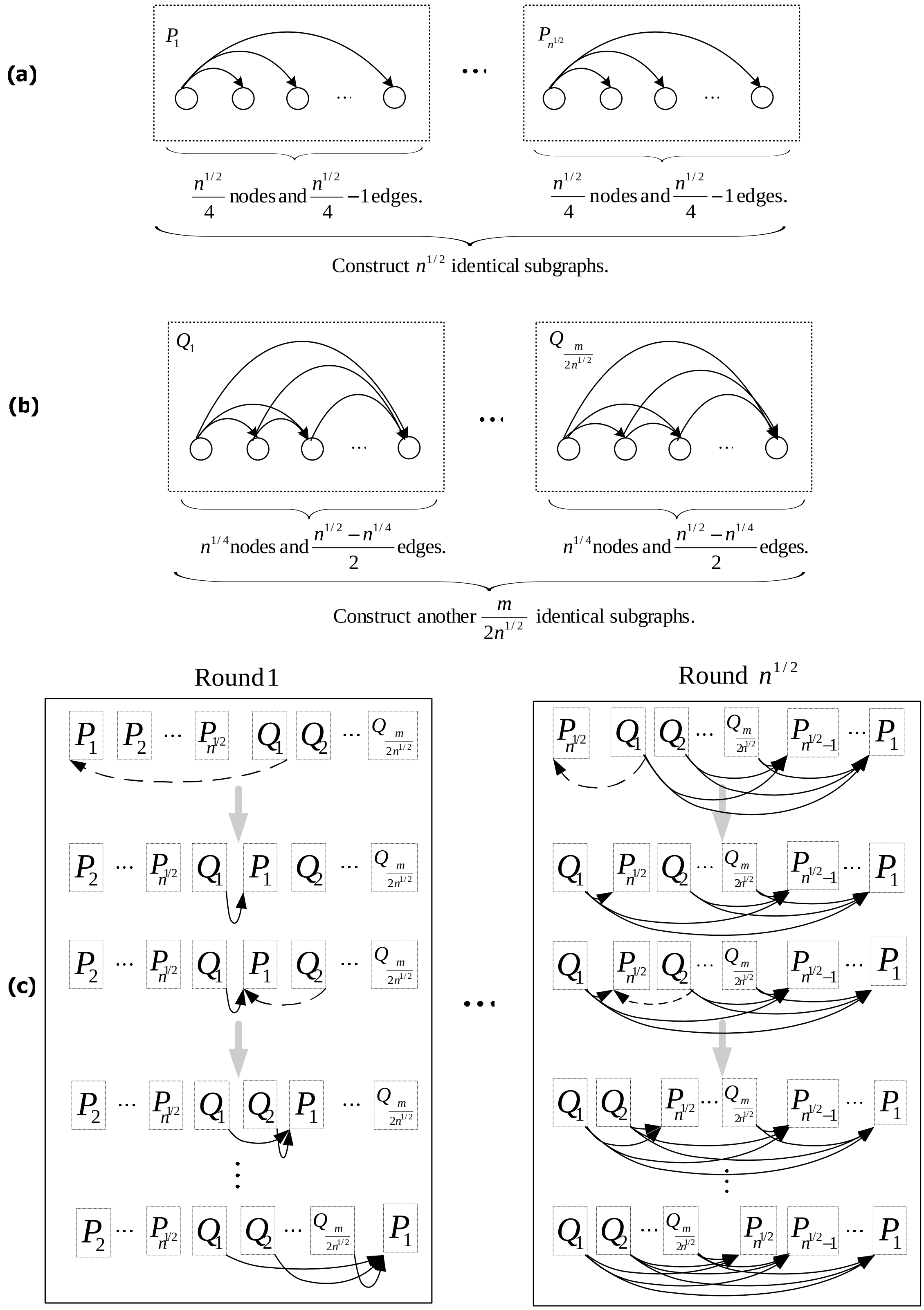

Figure 2: An input which requires $\Omega(mn^{1/2} \log n)$ time if $n \leq m \leq n \log^2 n$.

[2] Bowen Alpern, Roger Hoover, Barry K. Rosen, Peter F. Sweeney, and F. Kenneth Zadeck. Incremental Evaluation of Computational Circuits. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), 32-42, 1990.

[3] Michael A. Bender, Richard Cole, Erik D. Demaine, Martin Farach-Colton, and Jack Zito. Two Simplified Algorithms for Maintaining Order in a List. In *Proceedings of the 10th Annual European Symposium on Algorithms* (*ESA*), 152-164, 2002.

[4] Paul F. Dietz and Daniel D. Sleator. Two Algorithms for Maintaining Order in a List. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing* (*STOC*), 365-372, 1987.

[5] Michael L. Fredman, Robert E. Tarjan. Fibonacci Heaps and their Uses in Improved Network Optimization Algorithms. *Journal of the ACM*, 34(3): 596-615, 1987.

[6] Irit Katriel. On Algorithms for Online Topological Ordering and Sorting. Research Report MPI-I-2004-1-003, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 2004.

[7] Irit Katriel and Hans L. Bodlaender. Online Topological Ordering. *ACM Transactions on Algorithms*, 2(3): 364-379, 2006.

[8] Alberto Marchetti-Spaccamela, Umberto Nanni, and Hans Rohnert. On-Line Graph Algorithms for Incremental Compilation. In *Proceeding of International Workshop on Graph-Theoretic Concepts in Computer Science* (*WG*), 70-86, 1993.

[9] Alberto Marchetti-Spaccamela, Umberto Nanni, and Hans Rohnert. Maintaining a Topological Order Under Edge Insertions. *Information Processing Letters* 59(1): 53-58, 1996.

[10] Laurent Michel and Pascal Van Hentenryck. A constraint-based Architecture for Local Search. In *Proceedings of the 17th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (*OOPSLA*), 83-100, 2002.

[11] Stephen M. Omohundro, Chu-Cheow Lim, and Jeff Bilmes. The Sather Language Compiler/Debugger Implementation. *Technical Report TR-92-017, International Computer Science Institute, Berkley*, 1992.

[12] David J. Peace. Some Directed Graph Algorithms and Their Application to Pointer Analysis. Ph.D. Thesis, Imperial College of Science, Technology and Medicine, University of London, 2005.

[13] David J. Pearce, Paul H. J. Kelly, and Chris Hankin. Online Cycle Detection and Difference Propagation: Applications to Pointer Analysis. *Software Quality Journal*, 12(4): 309–335, 2004.

[14] David J. Pearce and Paul H. J. Kelly. A Dynamic Algorithm for Topologically Sorting Directed Acyclic Graphs. *ACM Journal of Experimental Algorithms*, 11(1.7): 1-24, 2007.

[15] Ganesan Ramalingam and Thomas W. Reps. On Competitive On-Line Algorithms for the Dynamic Priority-Ordering Problem. *Information Processing Letters*, 51(3): 155-161, 1994.