DIRECTED ACYCLIC GRAPHS AND DISJOINT CHAINS

Yangjun Chen

Dept. Applied Computer Science, University of Winnipeg, Canada

Abstract: The problem of decomposing a DAG (directed acyclic graph) into a set of disjoint chains has many applications in data engineering. One of them is the compression of transitive closures to support reachability queries, by which we will be able to decompose G into a minimal set of disjoint chains in O(n^2 + bn \sqrt{|B|}) time, where n is the number of the nodes of G, and b is G’s width, defined to be the size of a largest node subset U of G such that for every pair of nodes u, v \in U, there does not exist a path from u to v or from v to u. However, in some cases, it fails to do so. In this paper, we analyze this algorithm and show the problem. More importantly, a new algorithm is discussed, which can always find a minimal set of disjoint chains in the same time complexity as Chen’s.

1 INTRODUCTION

Given a DAG G(V, E) (directed acyclic graph) with |V| = n and |E| = e, we want to decompose it into a minimal set of disjoint chains such that any node in G appears on some chain, and on each chain, if node v appears above node u, there is a path from v to u in G.

This problem is important to compressing a transitive closure (Wang et al., 2006; Warshall, 1962) to support reachability queries, by which we will check whether a given node v in G is reachable from another node u through a path in G. Recently, an interesting algorithm is proposed by Chen et al. (Y. Chen and Y. Chen, 2008) which claims to be able to decompose G into a minimal set of disjoint chains in O(n^2 + bn \sqrt{|B|}) time, where n is the number of the nodes of G, and b is G’s width, defined to be the size of a largest node subset U of G such that for every pair of nodes u, v \in U, there does not exist a path from u to v or from v to u. However, in some cases, it fails to do so. In this paper, we analyze this algorithm and show the problem.

More importantly, a new algorithm is discussed, which can always find a minimal set of disjoint chains in the same time complexity as Chen’s.

Keywords: Graphs, DAGs, Chains, Paths, Transitive Closure, Reachability Queries.
2 ALGORITHM DESCRIPTION

In this section, we give our new algorithm, which is inspired by Chen’s algorithm. However, to remove the problem in Chen’s algorithm, we devise two new procedures for generating chains and resolving virtual nodes, respectively.

First, for the chain generation, we distinguish between two kinds of virtual nodes and handle them in different ways so that the reachability between nodes can be transferred bottom-up by using such virtual nodes.

Second, for the virtual node resolution, a new data structure, the so-called combined alternating graph, is constructed so that the number of virtual nodes resolved at each level is maximized.

In the following, we first discuss how a DAG can be decomposed into disjoint chains which may contain virtual nodes in 2.1. Then, in 2.2, we show how the virtual nodes can be resolved.

2.1 DAG Stratification and Chain Generation

As with Chen’s algorithm, our algorithm works in three phases: DAG stratification, chain generation, and virtual node resolution.

In the first phase, a DAG $G(V, E)$ is stratified into several levels $V_0$, ..., $V_h$, such that $V = V_0 \cup \cdots \cup V_h$ and each node in $V_i$ has its children appearing only in $V_{i-1}$, ..., $V_1$ ($i = 2, \ldots, h$), where $h$ is the height of $G$, i.e., the length of the longest path in $G$. For each node $v$ in $V_i$, its level is said to be $i$, denoted $l(v) = i$. In addition, $C(v)$ ($j < i$) represents a set of links with each pointing to one of $v$’s children, which appears in $V_j$. Therefore, for each $v$ in $V_i$, there exist $1, \ldots, i$ ($i < j < i, i = 1, \ldots, h$) such that the set of its children equals $C_i(v) \cup \cdots \cup C_h(v)$. Assume that $V_i = \{v_1, v_2, \ldots, v_k\}$. We use $C_j(v)$ ($j < i$) to represent $C_i(v) \cup \cdots \cup C_j(v)$.

This phase is almost the same as Chen’s. But for each node $v$ at a level, we also use $B_i^{(v)}$ to represent a set of links with each pointing to one of $v$’s parents, which appears in $V_j$.

In the second phase, a series of (undirected) bipartite graphs (Asrtian et al., 1998; Hopcroft et al., 1973) will be constructed. In this process, some virtual nodes may be introduced into the levels $V_i$ ($i = 1, \ldots, h - 2$). Especially, we distinguish between two kinds of virtual nodes. One is the virtual nodes created for actual nodes; and the other is the virtual nodes generated for virtual nodes. They will be handled differently.
In the following, we begin our discussion with a summarization of some important concepts related to bipartite graphs, which are needed to define virtual nodes.

**Definition 1. (concepts related to matching, Asrian et al., 1998)** Let \( G(V, E) \) be a bipartite graph. Let \( M \) be a maximum matching of \( G \). A node \( v \) is said to be covered by \( M \), if some edge of \( M \) is incident to \( v \). We will also call an uncovered node free. A path or cycle is alternating, relative to \( M \), if its edges are alternately in \( EM \) and \( M \). A path is an augmenting path if it is an alternating path with free origin and terminus.

In addition, it is well known that using the Hopcroft-Karp algorithm (Hopcroft et al., 1973) a maximum matching of \( G \) can be found in \( O(|E| \sqrt{|V|}) \) time.

Also, the following symbols are used for ease of explanation:
\[
V_i = V_i \cup \{ \text{virtual nodes introduced into } V_i \}.
\]
\[
C_i = \{ v \} \cup \{ \text{all the new edges from the nodes in } V_i \text{ to the virtual nodes introduced into } V_i \}.
\]
\[
G(V_i, V_i'; C_i) \text{- the bipartite graph containing } V_i \text{ and } V_i'.
\]

**Definition 2. (virtual nodes for actual nodes)** Let \( G(V, E) \) be a DAG, divided into \( V_0, \ldots, V_{h+1} \) (i.e., \( V = V_0 \cup \ldots \cup V_{h+1} \)). Let \( M_i \) be a maximum matching of the bipartite graph \( G(V_i, V_i'; C_i) \) and \( v \) be a free actual node (in \( V_{i+1} \)) relative to \( M_i \) (\( i = 1, \ldots, h - 1 \)).

Add a virtual node \( v' \) into \( V_i \). In addition, for each node \( u \in V_{i+1} \), a new edge \( u \rightarrow v' \) will be created if one of the following two conditions is satisfied:
1. \( u \rightarrow v \in E \); or
2. There exists an edge \((v_1, v_2)\) covered by \( M_i \) such that \( v_1 \) and \( v \) are connected through an alternating path relative to \( M_i \); and \( u \in B_{i+1}(v_1) \) or \( u \in B_{i+1}(v_2) \).

\( v' \) is called the source of \( v \), denoted \( s(v') \).

A virtual edge from \( v' \) to \( v \) is also generated to indicate the relationship between \( v' \) and \( v \). Besides, a new edge \( u \rightarrow v' \) will be marked with 'directly connectable' if one of the following conditions are satisfied:
1. \( u \rightarrow v \in E \); or
2. There is an alternating path of length 1, which connects \( v_3 \) and \( v_4 \). That is, \( v_3 \rightarrow v_4 \rightarrow v \).

We mark these edges with 'directly connectable' because it is possible for us to directly connect \( u \) and \( v \) to remove \( v' \).

The following example helps for illustration.

**Example 1.** Consider the graph shown in Fig. 3(a).

It can be divided into three levels as shown in Fig. 3(b). The bipartite graph made up of \( V_1 \) and \( V_0 \), \( G(V_1, V_0; C_1) \), is shown in Fig. 3(c) and a possible maximum matching \( M_i \) of it is shown in Fig. 3(d).

**Figure 3:** A bipartite graph and a maximum matching.

Relative to \( M_i \), we have two free nodes \( i \) and \( a \). For them, two virtual nodes \( i' \) and \( a' \) will be constructed. Then, \( V_i' = \{ b, e, h, i', a' \} \). In addition, four new edges \((d, i'), (d, a'), (g, i'), \) and \((g, a')\) will be constructed. But all of them will not be marked with 'directly connectable'.

The motivation of constructing such a virtual node (e.g., \( i' \)) is that it is possible to connect \( f \) to \( d \) or \( g \) to form part of a chain if we transfer the edges on an alternating path: \( b \rightarrow c \rightarrow e \rightarrow f \) (see Fig. 3(e), where a solid edge represents an edge belonging to \( M_i \) while a dashed edge to \( C_i \{M_i \} \), or \( h \rightarrow f \rightarrow b \rightarrow c \rightarrow e \rightarrow f \). Then, we can connect \( d \) or \( g \) to \( f \), as well as \( b \) or \( h \) to \( i \) without increasing the number of chains, as illustrated in Fig. 3(f). This can be achieved by the virtual node resolution process (see 2.2).

For the graph shown in Fig. 4(a), which is the second bipartite graph established for the graph shown in Fig. 3.4(a), a possible maximum matching \( M_2 \) is shown in Fig. 4(b). So \( M_1 \cup M_2 \) is a set of chains as shown in Fig. 4(c).

**Definition 3. (virtual nodes for virtual nodes)** Let \( M_i \) be a maximum matching of the bipartite graph \( G(V_i, V_i'; C_i) \) and \( v' \) be a free virtual node (in \( V_{i+1} \)) relative to \( M_i \) (\( i = 1, \ldots, h - 1 \)). Add a virtual node \( v'' \) into \( V_i \). Set \( s(v'') \) to be \( w = s(v') \). Let \( k(w) = j \). For each node \( u \in V_{i+1} \), a new edge \( u \rightarrow v' \) will be created if there exists an edge \((v_1, v_2)\) covered by \( M_{i+1} \) such that \( v_1 \) and \( v \) are connected through an alternating path relative to \( M_{i+1} \) and \( u \in B_{i+1}(v_1) \) or \( u \in B_{i+1}(v_2) \).

Again, a virtual edge from \( v'' \) to \( v' \) will be generated to facilitate the virtual node resolution process.
Example 2. Consider the graph shown in Fig. 5(a).

This graph can be divided into four levels as shown in Fig. 5(b). The first bipartite graph consisting of \( V_1 \) and \( V_0 \), \( G(V_1, V_0; C_1) \), is shown in Fig. 5(c) and a possible maximum matching \( M_1 \) of it is shown in Fig. 5(d). Relative to \( M_1 \), we have a free node \( f \). For it, a virtual nodes \( f' \) will be constructed. Then, \( V_1' = \{b, f', d, h\} \) (see Fig. 5(e)). Assume that the maximum matching found for \( G(V_2; V_1'; C_2) \) is as shown in Fig. 5(f). A virtual node \( f'' \) for \( f' \) will be established. So \( V_2' = \{f'', e, g\} \). Especially, we are able to connect node \( f'' \) and node \( p \) for the following reason:

i) \( s(f') = s(f) = f; \)

ii) \( (b, e) \in M_1; \)

iii) \( f \) is connected to \( b \) through an alternating path: \( f \rightarrow b; \) and

iv) \( p \in B_3(c). \)

The corresponding bipartite graph \( G(V_3, V_2'; C_3) \) is shown in Fig. 5(g). The unique maximum matching of \( G(V_3, V_2'; C_3) \) is shown in Fig. 5(h). By unifying \( M_1, M_2, \) and \( M_5 \), we get a set of disjoint chains as shown in Fig. 6(a).

2.2 Virtual Node Resolution

In the third phase, we will remove all the virtual nodes. This will be done top-down level by level; and at each level any virtual node, which does not have a parent along a chain, will be simply eliminated. In addition, we call a virtual node \( v' \) a transit virtual node if one of the following two conditions is satisfied.

1. Let \( u, v', w \) be three consecutive nodes on a chain.
   \( u \rightarrow v' \) is a marked edge (i.e., a directly connectable edge); or

2. \( w \) is a virtual node.

In both cases, we connect \( u \) and \( w \) and then remove \( v' \). It is because in case (1), both \( u \) and \( w \) are actual nodes and we have \( u \rightarrow w \in E \) or there exists a actual node \( x \) such that \( u \rightarrow x \in E \) and \( x \rightarrow w \in E \). In case (2), \( w \) is a virtual node, working as a ‘transfer’ of reachability.

For example, since node \( f' \) in Fig. 6(a) is a virtual node, node \( f'' \) is a transit virtual node. It can be directly removed, leading to a set of chains as shown in Fig. 6(b). But node \( f' \) cannot be removed in this way since it is not a transit virtual node.

In the following, we discuss how to resolve a non-transit virtual node, for which more effort is needed. First, we define a new concept.

Definition 4. (alternating graph) Let \( M_i \) be a maximum matching of \( G(V_i, V_{i+1}'; C_i) \). The alternating graph \( \tilde{G}_i \) with respect to \( M_i \) is a directed graph with the following sets of nodes and edges:

\[
\tilde{V}(\tilde{G}_i) = V_i \cup V_{i+1}', \quad \text{and} \quad \tilde{E}(\tilde{G}_i) = \{u \rightarrow v \mid u \in V_i, v \in V_{i+1}', (u, v) \in M_i \} \cup \{v \rightarrow u \mid u \in V_{i+1}', v \in V_i, (u, v) \in C_i \setminus M_i \}.
\]

Example 3. Consider the graph shown in Fig. 3(a) once again. Relative to \( M_i \) of \( G(V_i, V_{i+1}'; C_i) \) shown in Fig. 3(d), nodes 3 and \( \alpha \) are two free nodes. The alternating graph with respect to \( M_i \) is shown in Fig. 7(a). It is redrawn in Fig. 7(b) for a clear explanation.

In order to resolve the non-transit virtual nodes in \( V_i' \), we will combine \( \tilde{G}_{i,3} \) and \( \tilde{G}_i \) by connecting...
some nodes \( v' \) in \( \tilde{G}_{i+1} \) to some nodes \( u \) in \( \tilde{G}_i \) if the following conditions are satisfied.

1. \( v' \) is a non-transit virtual node appearing in \( V_i' \).
   (Note that \( V(\tilde{G}_{i+1}) = V_{i+1} \cup V'_i \))

2. There exist a node \( x \) in \( V_{i+1} \) and a node \( y \) in \( V_i \) such that \( (x, v') \in M_{i+1}, x \to y \in C_{i+1} \), and \( (y, u) \in M_i \).

We denote this combined graph by \( \tilde{G}_{i+1} \oplus \tilde{G}_i \).

Figure 7: An alternating graph.

For illustration, consider \( G(V_2, V'_1; C_2) \) shown in Fig. 4(a). Assume that the found maximum matching \( M_2 \) is as shown in Fig. 4(b). Then, the alternating graph \( \tilde{G}_2 \) (with respect to \( M_2 \)) is a graph shown in Fig. 8(a). \( \tilde{G}_2 \oplus \tilde{G}_1 \) is shown in Fig. 8(b). Note that \( i' \) and \( a' \) are two non-transit virtual nodes.

Figure 8: Illustration for combined graph.

For instance, by transferring the edges on the path from \( a' \to e \) in \( \tilde{G}_2 \) in Fig. 9(a), we will connect \( h \) to \( a \) (in \( \tilde{G}_1 \)), and \( i' \) will be removed. By transferring the edges on the path from \( i' \to a \) in Fig. 9(b), we will get a set of disjoint chains shown in Fig. 10(a) with all the virtual nodes being removed. The number of chains is still 5.

In order to resolve as many non-transit virtual nodes (appearing in \( V_i' \)) as possible, we need to find a maximum set of node-disjoint paths (i.e., no two of these paths share any nodes), each starting at a non-transit virtual node (in \( \tilde{G}_{i+1} \)) and ending at a free node in \( \tilde{G}_i \), or ending at a free node in \( \tilde{G}_i \). For example, to resolve \( a' \) and \( i' \), we need first to find two paths in the above combined graph, as shown in Fig. 9(a).

Figure 9: Illustration for node-disjoint paths.

(1) Let \( v_1 \to v_2 \to \ldots \to v_b \) be a found path. Transfer the edges on the path.

(2) If \( v_k \) is a node in \( \tilde{G}_{i+1} \), we simply remove the corresponding virtual node \( v_k \).

(3) If \( v_k \) is a node in \( \tilde{G}_i \), connect the parent of \( v_k \) along the corresponding chain to \( v_2 \). Remove \( v_1 \).

As mentioned above, we connect \( a' \) to \( f \) since it is possible for us to transfer the edges on an alternating path (relative to \( M_1 \)) starting from node \( f \) (relative to \( M_1 \)) and terminating at free node \( i \) or \( a \) (in \( V_{i+1} \)), which will make \( i \) or \( a \) covered without increasing the number of chains.

The same analysis applies to node \( i' \) (in \( \tilde{G}_2 \)), which is also connected to node \( f \) (in \( \tilde{G}_1 \)).
It remains to show how to find a maximal set of node-disjoint paths in \( \tilde{G}_{i+1} \oplus \tilde{G}_{i} \).

For this purpose, we define a maximum flow problem over \( \tilde{G}_{i+1} \oplus \tilde{G}_{i} \) (with multiple sources and sinks) as follows.

1) Each non-transit virtual node in \( \tilde{G}_{i+1} \) is designated as a source. Each free node (in \( \tilde{G}_{i} \)) relative to \( M_{i+1} \), or free node (in \( \tilde{G}_{i} \)) relative to \( M_i \) is designated as a sink.

2) Each edge \( u \to v \) is associated with a capacity \( c(u, v) = 1 \). (If \( (u, v) \) is not an edge in \( \tilde{G}_{i+1} \oplus \tilde{G}_{i} \), \( c(u, v) = 0 \).)

3) time \( (Even, 1979; Karzanov, 1974; Cotman et al., 2001) \). However, a network as constructed above is a 0-1 network. In addition, for each node \( v \), we have either \( d_{in}(v) \leq 1 \) or \( d_{out}(v) \leq 1 \), where \( d_{in}(v) \) and \( d_{out}(v) \) represent the indegree and outdegree of \( v \) in \( \tilde{G}_{i+1} \oplus \tilde{G}_{i} \), respectively. It is because each path in \( \tilde{G}_{i+1} \oplus \tilde{G}_{i} \) is an alternating path relative to \( M_{i+1} \) or relative to \( M_i \). So each node excerpt sources and sinks is an end node of an edge covered by \( M_{i+1} \) or by \( M_i \). As shown in (Even, 1979; Theorem 6.3 on page 120), it needs only \( O(\sqrt{n} e) \) time to find a maximum flow in this kind of networks. Especially, a maximum flow exactly corresponds to a maximal set of disjoint paths. See the proof of Lemma 6.4 in (Even, 1979, page 120.)

According to the above discussion, we give the following algorithm for resolving virtual nodes. We assume that each virtual node has a parent along a chain. Otherwise, it can be simply eliminated.

**Algorithm virtual-resolution(S)**

input: \( S \) - a chain set obtained by executing the chain generation process.

output: a set of chains containing no virtual nodes.

begin
1. for \( i = h - 2 \) downto 1 do
2. {for any transit virtual node \( v' \) in \( V_{i}' \) do
3. |
4. | let \( u, v', w \) be three consecutive nodes on a chain;
5. | connect \( u \) and \( w \);}
6. |
7. construct \( \tilde{G}_{i+1} \oplus \tilde{G}_{i} \); (*Begin to handle non-transit virtual nodes.*)
8. find a maximal set of node disjoint paths: \( P_1, \ldots, P_l \);
9. for \( j = 1 \) to \( l \) do
10. {let \( P_j = v_1 \to v_2 \to \ldots \to v_{k_j} \);
11. if \( v_k \) is a free node relative to \( M_i \) then
12. {transfer the edges on \( P_j \); remove \( v_k \);} 
13. else (* \( v_k \) is a free node relative to \( M_{i+1} *)
14. {let \( \nu \) be a node such that \( (\nu, v_k) \in M_i \);
15. transfer the edges on \( P_j \); remove \( v_k \);
16. connect \( \nu \) to \( v_k \);
17. }
18. removed any unsolved virtual node;
19. }
end

In the main for-loop of the above algorithm, we first handle transit virtual nodes (lines 2 - 6). Then, we construct \( \tilde{G}_{i+1} \oplus \tilde{G}_{i} \) to resolve all the non-transit virtual nodes (see line 7). For this purpose, we search for a maximal set of node disjoint paths (see line 8). We also distinguish between two kinds of node disjoint paths: paths ending at a free node relative to \( M_i \) and paths ending at a free node relative to \( M_{i+1} \). For the first kind of paths, we simply transfer the edges on a path and then remove the corresponding virtual node (see line 12). For the second kind of paths, we need to do something more to connect the parent of the corresponding virtual node (along the chain) to the second node of the path (see line 16). In line 18, we remove all those virtual nodes, which cannot be resolved. Each of such virtual nodes leads to splitting of a chain into two chains.

Note that removing a transit virtual node will not increase the number of chains. Also, resolving a non-transit virtual node using a node disjoint path does not lead to a chain splitting. So the number of increased chains during the virtual node resolution process is minimum since the number of node disjoint paths is maximum.

### 3 TIME COMPLEXITY

Now we analyze the computational complexities of our algorithm. The cost of the whole process can be divided into three parts:

- cost\(_1\): the time for stratifying a DAG.
- cost\(_2\): the time for generating disjoint chains, which may contain virtual nodes.
- cost\(_3\): the time for resolving virtual nodes.

As shown in (Chen and Chen, 2008), cost\(_1\) is bounded by \( O(n + e) \).

cost\(_2\) mainly contains two parts. One part: cost\(_{21}\) is the time for finding a maximum matching of every \( G(V', V_{i+1}'; C) (i = 1, \ldots, h - 1; V_0' = V_0) \). The other part: cost\(_{22}\) is the time for checking whether, for each actual free node appearing in \( V_{i+1}' \), there exists an
edge \((v_i, v_j)\) covered by \(M_i\) such that \(v_i\) and \(v_j\) are connected through an alternating path relative to \(M_i\). The time for finding a maximum matching of \(G(V_i, V_i' \setminus C_i)\) is bounded by 
\[
O(\sqrt{|V_i'| + |V_i'| |C_i'|}). \text{ (see Chen et al., 2008)}
\]
Therefore, \(\text{cost}_{i2}\) is bounded by 
\[
O(\sum_{i=1}^{n} |V_i'| + |V_i'| |C_i'|) 
\]
\[
\leq O(\sqrt{b} \sum_{i=1}^{n} b_i |V_i'|) = O(b \sqrt{b} n).
\]

\(\text{cost}_{22}\) can be analyzed as follows. We construct a small boolean \(n \times m_i\) matrix \(A_i\), where \(n_i\) is the number of free actual nodes in \(V_i\) and \(m_i\) is the number of all the covered actual nodes in \(V_i\). Each entry \(a_{jk} = 1\) in \(A_i\) indicates that there exists an alternating path (relative to \(M_i\)) connects node \(j\) and \(k\). Using the algorithm discussed in (Coppersmith et al., 1990) for matrix multiplication, \(\text{cost}_{22}\) can be estimated by 
\[
O(\sum_{i=1}^{n} (|V_i'| + |V_i'|)^{2^{376}})
\]
\[
= O(\sum_{i=1}^{n} (|V_i'| + |V_i'| + |V_i'|)^{376})
\]
\[
\leq O(\sqrt{b} \sum_{i=1}^{n} b_i |V_i'|) = O(b \sqrt{b} n).
\]

During the virtual-resolution process, the virtual nodes are resolved level by level. At each level, the number of the newly added edges in each bipartite graph \(G(V_i, V_i' \setminus C_i)\) is bounded by \(O(b |V_i'|)\), and the size of each matrix \(A_i\) is bounded by \(O(|V_i'|^2)\).

\section{Conclusions}

In this paper, a new algorithm for resolving virtual nodes is discussed, which is a critical step in an algorithm proposed by Chen et al. (Chen and Chen, 2008) to decompose a DAG into a set of disjoint chains. In addition, the virtual node resolution process of Chen’s algorithm is analyzed, showing that in some cases Chen’s algorithm fails to find a minimal set of disjoint chains. The main idea of our algorithm is the construction of alternating graphs. By finding a maximal set of disjoint paths in such a graph to resolve virtual nodes, we are able to guarantee that at each step of virtual node resolution, the number of increased chains is minimum.

\section*{References}


N. H. Cohen, "Type-extension tests can be performed in constant time," ACM Transactions on Programming Languages and Systems, 13:626-629, 1991.


23
H. Wang, H. He, J. Yang, P.S. Yu, and J. X. Yu, Dual Labeling: Answering Graph Reachability Queries in Constant time, in *Proc. of Int. Conf. on Data Engineering*, Atlanta, USA, April -8 2006.