On the Decomposition of Posets

Yangjun Chen[1], Yibin Chen[2]

Dept. Applied Computer Science, University of Winnipeg, Canada

[1]y.chen@uwinnipeg.ca, [2]chenyibin@gmail.com

***Abstract*— In this paper, we propose an efficient algorithm to decompose a partially ordered set *S* into a minimum set of chains. It requires only $O(\kappa \cdot n^2)$ time and space, where *n* is the number of the elements in *S* and κ is the size of a maximum anti-chain of *S*.**

I. INTRODUCTION

A *partially ordered* set (*poset* for short) $\boldsymbol{S}$ is a pair $(S, \succeq)$ of a set S and a binary relation $\succeq$ such that for each a, b, and c in S:

1. $a \succeq a$ is true ($\succeq$ is *reflexive*),
2. $a \succeq b$ and $b \succeq c$ imply $a \succeq c$ ($\succeq$ is *transitive*), and
3. $a \succeq b$ and $b \succeq a$ imply $a = b$ ($\succeq$ is *antisymmetric*).

If we have a *poset* $\boldsymbol{S} = (S, \succeq)$, a *chain* in S is a non-empty subset $C = \{a_1, a_2, ..., a_k\} \subseteq S$ such that $a_1 \succ a_2 \succ ... \succ a_k$.

Two elements of S are called *comparable* if they appear together in some chain in $\boldsymbol{S}$; elements which are not comparable are called *incomparable*. A non-empty set, in which every pair of elements is not comparable, is called an *antichain*.

Since each single element in S is itself a chain, it is always possible to partition the elements in S into disjoint chains. Such a partition is called a decomposition, and a decomposition consisting of the smallest number of disjoint chains is called minimum. According to Dilworth [8], the size of a minimum decomposition equals the size of a maximum antichain.

In 1956, Fulkerson [10] provided a very simple proof of the Dilworth's theorem. He constructed a bipartite graph G_S with bipartite (V_1, V_2) for $S = \{a_1, a_2, ..., a_n\}$, where $V_1 = \{x_1, x_2, ..., x_n\}$, $V_2 = \{y_1, y_2, ..., y_n\}$ and an edge joining $x_i \in V_1$ to $y_j \in V_2$ whenever $a_i \succ a_j$. Let M be a maximum matching of G_S and D a minimum decomposoition of $\boldsymbol{S}$. Fulkerson proved that $|D| = n - |M|$. On the other hand, by the König's theorem ([2], page 180), we also have $\kappa = n - |M|$, where κ is the size of a maximum antichain of $\boldsymbol{S}$. So, $|D| = \kappa$. Using the algorithm proposed by Hopcroft and Karp [11], M can be found in $O(m \cdot \sqrt{n})$ time, where m is the number of all pairs (a, c) such that $a \succ c$. So the maximum size of antichains can be determined in $O(m \cdot \sqrt{n})$ time. However, Fulkerson did not show how to decompose $\boldsymbol{S}$ into a minimum set of chains. In fact, it is a problem that has not yet been solved. In their book, Asratian *et al*. [2] wrote:

> "Less clear is how to decompose $\boldsymbol{S}$ into as few chains as possible." (See [2], page 190.)

In this paper, we address this problem and propose an efficient algorithm to find a minimized set of chains for $\boldsymbol{S}$. For this purpose, we represent $\boldsymbol{S}$ as a *DAG* (Directed Acyclic Graph, containing no cycles), in which we have an arc $u \rightarrow v$ if $u \succ v$. Removing any arc $u \rightarrow v$ if there is path of length ≥ 2 from u to v, we get another graph G'. A minimum set of chains that covers G' must be a decomposition of $\boldsymbol{S}$. (On a chain, if u appears above v, there is a path from u to v.) The minimal number of chains is also called the width of G.

Our algorithm runs in $O(\kappa \cdot n^2)$ time and in $O(\kappa \cdot n^2)$ space.

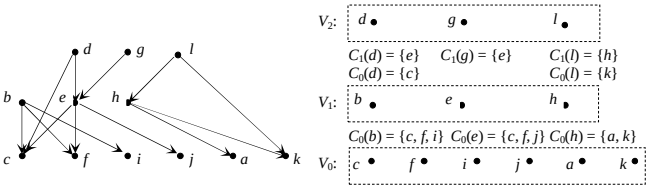
The poset decomposition is quite useful in practice. As shown in [4], if we can decompose a DAG G into a minimum set of chains, the transitive closure [24] of G can be effectively compressed to support the so-called reachability queries [5, 6].

II. GRAPH STRATIFICAION AND BIPARTITE GRAPH

Our method is based on a DAG stratification strategy and an algorithm for finding a maximum matching in a bipartite graph. Therefore, the relevant concepts and techniques should be first reviewed and discussed.

Let $G(V, E)$ be a DAG. We decompose V into subsets V_0, V_1,..., V_h such that $V = V_0 \cup V_1 \cup ... \cup V_h$ and each node in V_i has its children appearing only in V_{i-1}, ..., V_0 (i = 1, ..., 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, we say, its level is i, denoted $l(v) = i$. We also use $C_j(v)$ ($j < i$) to represent 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 i_1, ..., i_k ($i_l < i$, l = 1, ..., k) such that the set of its children equals $C_{i_1} \cup ... \cup C_{i_k}$. Let $V_i = \{v_1, v_2, ..., v_l\}$. We use $(j < i)$ $\boldsymbol{C}_j^i$ to represent $C_j(v_1) \cup ... \cup C_j(v_l)$.

As an example, consider the graph shown in Fig. 1(a). We can divide it into three levels as shown in Fig. 1(b).



(a) (b)

Fig. 1 Illustration for DAG stratification

In Fig. 1(b), the nodes of the DAG shown in Fig. 1(a) are divided into three levels: $V_0 = \{c, f, i, j, a, k\}$, $V_1 = \{b, e, h\}$, and $V_2 = \{d, g, l\}$. Associated with each node at each level is a set of links pointing to its children at different levels.

Using the algorithm discussed in [4], we need only $O(m)$ time to stratify a DAG G.

978-1-4673-0719-2/12/$26.00 ©2012 IEEE

Now we restate two concepts from the graph theory which will be used in the subsequent discussion.

Definition 1 (*bipartite graph* [2]) An undirected graph $G(V, E)$ is bipartite if the node set V can be partitioned into two sets T and S in such a way that no two nodes from the same set are adjacent. We also denote such a graph as $G(T, S; E)$. □

For any node $v \in G$, *neighbour*(v) represents a set containing all the nodes connected to v.

Definition 2 (*matching* [2]) Let $G(V, E)$ be a bipartite graph. A subset of edges $E' \subseteq E$ is called a *matching* if no two edges have a common end node. A matching with the largest possible number of edges is called a *maximal matching*, denoted as M_G. □

Let M be a matching of a bipartite graph $G(T, S; E)$. A node v is said to be *covered* by M, if some edge of M is incident with v. We will also call an uncovered node *free*. A path or cycle is *alternating*, relative to M, if its edges are alternately in $E\backslash M$ and M. A path is an *augmenting path* if it is an alternating path with free origin and terminus. Let $v_1 - v_2 - ... - v_k$ be an alternating path with $(v_i, v_{i+1}) \in E\backslash M$ and $(v_{i+1}, v_{i+2}) \in M$ (i = 1, 3, ...). By transferring the edges on the path, we change it to another alternating path with $(v_i, v_{i+1}) \in M$ and $(v_{i+1}, v_{i+2}) \in E\backslash M$ (i = 1, 3, ...). In addition, we will use $free_M(T)$ and $free_M(S)$ to represent all the free nodes in T and S, respectively. Finally, if $(u, v) \in M$, we say, u covers v with respect to M, and *vice versa*.

Much research on finding a maximal matching in a bipartite graph has been done. The best algorithm for this task is due to Hopcroft and Karp [11] and runs in $O(m\cdot\sqrt{n})$ time, where $n = |V|$ and $m = |E|$. The algorithm proposed by Alt, Blum, Melhorn and Paul [1] needs $O(n^{1.5}\cdot\sqrt{m/(\log n)})$ time. In the case of large m, the latter is better than the former.

In addition, for a graph G, we will use $V(G)$ to represent all its nodes and $E(G)$ all its edges (arcs).

III. Algorithm description

In this section, we describe our algorithm for the DAG decomposition. The main idea of our algorithm is to construct a series of bipartite graphs for $G(V, E)$ and then find a maximum matching for each of such bipartite graphs using Hopcroft-Karp algorithm. All these matchings make up a set of disjoint chains and the size of this set is equal to the size of a maximum antichain. During the process, some virtual nodes may be introduced into V_i (i = 1, ..., h - 1; $V = V_0 \cup V_1 \cup ... V_h$) to facilitate the computation. However, such virtual nodes will be eventually resolved to obtain the final result.

In the following, we first give a formal definition of virtual nodes and show how a virtual node can be efficiently constructed in Subsection *A*. Then, in Subsection *B*, we discuss how the virtual nodes can be resolved (removed) from created chains.

A. Virtual nodes

We start our discussion with the following specification:

$V_0' = V_0$.

$V_i' = V_i \cup$ {virtual nodes added to V_i} for $1 \le i \le h$ - 1.}

$\boldsymbol{C}_i = \boldsymbol{C}_{i-1}^i \cup$ {all the new arcs from the nodes in V_i to the virtual nodes added to V_{i-1}'} for $1 \le i \le h$ - 1.}

$G(V_i, V_{i-1}'; \boldsymbol{C}_i)$ - the bipartite graph containing V_i and V_{i-1}'.

M_i - a maximum matching of $G(V_i, V_{i-1}'; \boldsymbol{C}_i)$.

Definition 3 (*virtual nodes*) Let $G(V, E)$ be a DAG, divided into V_0, ..., V_h (i.e., $V = V_0 \cup ... \cup V_h$). Let M_i be a maximum matching of the bipartite graph $G(V_i, V_{i-1}'; \boldsymbol{C}_i)$. For each free node v in V_{i-1}' with respect to M_i, a virtual node v' created for v is a new node added to V_i ($1 \le i \le h$ - 1). □

The goal of virtual nodes is to establish the connection between the free nodes (with respect to a certain maximum matching) and the nodes that may be several levels apart. Therefore, for each virtual node v' (added to V_i and created for v in V_{i-1}'), a bunch of virtual arcs incident to it will be created. According to three different ways to create a virtual arc, a virtual arc can be labeled or not:

inherited arcs - If there is $u \in V_j$ ($j > i$) such that $u \to v \in E$, add $u \to v'$, which is not labeled. However, if $u \to v$ itself is a virtual arc, $u \to v'$ will inherit the label of $u \to v$. In both cases, $u \to v'$ is referred to as an inherited arc.

transitive arc - If there exist $u \to w \in E$ and $w \to v \in \boldsymbol{C}_i$ with $u \in V_j$ ($j > i$) and $w \in V_i$, add $u \to v'$ if it has not yet been created as an inherited arc. Such an arc is labeled with α and referred to as a transitive arc (α-arc for short).

alternating arc - If there exist $w \in V_{i-1}'$ such that v is connected to w through an alternating path, and $u \in V_j$ ($j > i$) such that one of the two conditions holds:

- $u \to w \in E$, or
- there is a node $w' \in V_i$ such that $u \to w' \in E$ and $w' \to w \in \boldsymbol{C}_i$,

add $u \to v'$ if it has not yet been created as an inherited or a transitive arc. We label such an arc with β and call it an alternating arc (β-arc for short).

A virtual arc from v' to v is also generated to indicate the relationship between v and v'. v is also called the source of v', denoted as $s(v')$.

Example 1 Consider the graph shown in Fig. 1(a) and the graph stratification shown in Fig. 1(b). The bipartite graph made up of V_0 and V_1, $G(V_1, V_0; E_1)$, is shown in Fig. 2(a) and a possible maximum matching M_1 of it is shown in Fig. 2(b). Relative to M_1, i, j and k are three free nodes. Then, three virtual nodes i', j' and k' (for i, j and k, respectively) will be created and added to V_1. Thus, we have $V_1' = \{b, e, h, i', j', k'\}$. Especially, five virtual arcs: $d \to i'$, $d \to j'$, $g \to i'$, $g \to j'$, and $l \to k'$ will be generated, as shown in Fig. 2(c). □

Among these virtual arcs, $l \to k'$ is an inherited arc since in the original graph (see Fig. 1(a)) we have an arc $l \to k$.

$d \to j'$ and $g \to j'$ are two α-arcs since j is reachable respectively from d and g through e, a node in V_1 (see Fig. 1(a)).

Finally, $d \to i'$ and $g \to i'$ are two β-arcs. We join d and i' since there is a node f that is connected to i through an alternating path: $f - e - c - b - i$ (see Fig. 1(a)) and f is reachable

from d through a node e in V_1. (We also note that c is another node connected to i through an alternating path: $c - b - i$, and $d \rightarrow c \in E$. If one such node exists, the corresponding arc should be established.) For the same reason, we join g and i'.

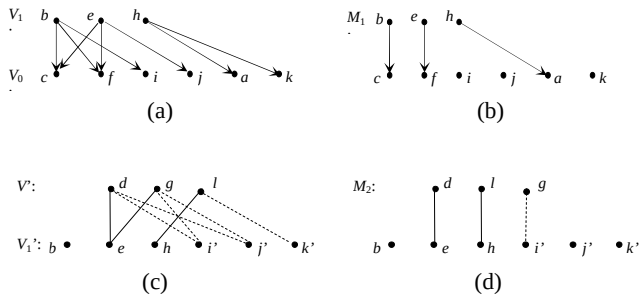


Fig. 2 Illustration for virtual nodes

In Fig. 2(d), we show a possible maximum matching M_2 of $G(V_2, V_1'; \boldsymbol{C}_2)$. Combining M_2 and M_1, we get a set of six chains as shown in Fig. 3(a).

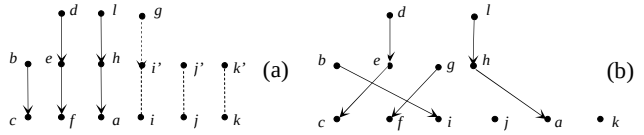


Fig. 3 Illustration chains with and without virtual nodes

The virtual nodes j' and k' can be simply removed. In order to remove i', however, we have to transfer the edges on the alternating path: $f - e - c - b - i$ and then connect g and f, obtaining the chains shown in Fig. 3(b). □

From the above discussion, we can see that any virtual node will be eventually resolved. Its roll is just to bridge the nodes at different levels. Each time a virtual node is removed, a node at a higher level may be connected to a node at a lower level in terms of the information represented by the corresponding virtual arc. Concretely, how to establish a connection depends on the property of the virtual arc that connects the virtual node and its parent along the corresponding chain.

For this purpose, we associate each α-arc and β-arc with a data structure to facilitate the virtual node resolution.

The data structure for a β-arc $e = u \rightarrow v'$, denoted by $\beta(e)$, is a pair of the form $<i, \{w_1, ..., w_k\}>$, where

- i is the level number, to which v' is added,
- each w_j is connected to v (= $s(v')$) through an alternating path, and
- for each w_j we have $u \rightarrow w_j \in E$, or there is a node $w' \in V_i$ such that $u \rightarrow w' \in E$ and $w' \rightarrow w \in \boldsymbol{C}_i$.

For example, the data structure for the β-arc $d \rightarrow i'$ shown in Fig. 2(c) should be $\beta(d \rightarrow i') = <1, \{c, f\}>$ for the following reason:

- i' is a virtual node added to V_1;
- c is connected to i through an alternating path $c - b - i$, and f is also connected to i through an alternating path $f - e - c - b - i$; and
- $d \rightarrow c \in E$, and $d \rightarrow e \in E$, $e \rightarrow f \in E$.

Similarly, the β-arc $g \rightarrow i'$ shown in Fig. 2(c) should also be associated with a data structure $\beta(g \rightarrow i') = <1, \{c, f\}>$.

For a $\beta(e) = <i, W>$, we use $\beta_1(e)$ and $\beta_2(e)$ to refer to i and W, respectively.

In the following, we discuss the data structure associated with an α-arc.

Let v' be a virtual node created for v, added to V_i. Let $e = u \rightarrow v'$ be an α-arc. Then, there must exist $w_1, ..., w_k$ ($k \geq 1$) in V_i such that for each w_j ($1 \leq j \leq k$) $u \rightarrow w_j \in E$ and $w_j \rightarrow v \in \boldsymbol{C}_i$ with $u \in V_l$ (for some $l > i$), as illustrated in Fig. 4.

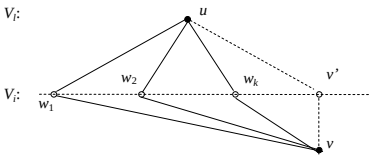


Fig. 4 Illustration for the creation of α-arcs

We distinguish among three cases:

i) There exists $w_{j_1}, ..., w_{j_p}$ ($1 \leq j_1 \leq j_p \leq k$) such that for each w_{j_q} ($1 \leq q \leq p$) $w_{j_q} \rightarrow v$ is an arc in the original graph or an unlabeled virtual arc.
ii) Each $w_j \rightarrow v$ is neither an arc in the original graph nor an unlabeled virtual arc. However, There exists $w_{j_1}, ..., w_{j_a}$ ($1 \leq j_1 \leq j_a \leq k$) such that for each w_{j_b} ($1 \leq b \leq a$), $w_{j_b} \rightarrow v$ is an α-arc.
iii) Each $w_j \rightarrow v$ is a β-arc.

In case (i), the data structure is set to be $\alpha(e) = <\text{I}, \phi>$. In case (ii) and (iii), the data structure is set to be $\alpha(e) = <\text{II}, \{w_{j_1}, ..., w_{j_a}\}>$ and $\alpha(e) = <\text{III}, \{w_1, ..., w_k\}>$, respectively.

For an $\alpha(e) = <\delta, W>$, we use $\alpha_1(e)$ and $\alpha_2(e)$ to refer to δ and W, respectively.

B. Virtual node resolution

All the virtual nodes have to be resolved. For this purpose, we work top-down level by level. Thus, when we try to remove the virtual nodes in V_i', all the virtual nodes appearing above V_i' must have been resolved. So we need only to clarify how the virtual nodes in V_{h-1}' are resolved. All the other virtual nodes at lower levels can be removed in the same way.

Consider $G(V_h, V_{h-1}'; \boldsymbol{C}_h)$. Relative to the found maximum matching M_h of it, all the virtual nodes in V_{h-1}' can be classified into four groups: uncovered (free nodes), *unlabeled-covered*, *transitive-covered*, and *alternating-covered*. A virtual node is unlabeled-covered, transitive-covered, or alternating-covered if it is covered by an edge in M_h, which corresponds to an unlabeled, transitive, or alternating arc, respectively.

Each uncovered virtual node can be simply removed. But for an unlabeled-covered virtual node v, we will connect its

parent u and its child $s(v)$ (along the corresponding chain) and then remove v. The new arc $u \to s(v)$ is handled as unlabeled. However, the treatments of transitive-covered and alternating-covered nodes are a little bit more difficult.

In the following, we discuss their resolution in great detail.

- Resolution of transitive-covered virtual nodes

Let v be a transitive-covered virtual node. Let u be a node in V_h such that $(u, v) \in M_h$. We will remove v and connect u to $s(v)$. For $u \to s(v)$, we need to do two tasks:

- Determine whether it is an α-arc, a β-arc, or unlabeled.
- If it is labeled, figure out the data structure for it.

To this end, we will do the following operations:

1. Let $\alpha(u \to v)$ be $<\delta, W>$, where $\delta \in \{\text{I, II, III}\}$ and W is a subset of V_{h-1}.
2. Remove v.
3. If δ = I, create an unlabeled arc $u \to s(v)$.
4. If δ = II, create an α-arc $u \to s(v)$. Let $W = \{w_1, ..., w_k\}$. Let $\alpha(w_j \to s(v)) = <\delta_j, W_j>$ (j = 1, ..., k). If there exists an j such that δ_j = I, set $\alpha(u \to s(v)) = <\text{I}, \phi>$.

   Otherwise, $X := \bigcup_{\delta_j=\text{II}} W_j$, $Y := \bigcup_{\delta_j=\text{III}} W_j$.

   i) If $X \neq \phi$, set $\alpha(u \to s(v))$ to be $<\text{II}, X>$.

   ii) If $X = \phi$, set $\alpha(u \to s(v))$ to be $<\text{III}, Y>$.
5. If δ = III, find $W' \subseteq W$ such that for each $x \in W'$ $\beta(x \to s(v))$ is of the form $<h - 2, W_x>$, where W_x is a subset of nodes in V_{h-3}'.

   i) If $W' \neq \phi$, create a β-arc $u \to s(v)$ and set $\beta(u \to s(v)) = <h - 2, \bigcup_{x \in W'} W_x>$.

   ii) If $W' = \phi$, create a transitive arc $u \to s(v)$ and set $\alpha(u \to s(v)) = \alpha(u \to v)$.

In the above process, replacing h with i, we get a general working process.

- Resolution of alternating-covered virtual nodes

Now we discuss how to resolve an alternating-covered virtual node.

First, we define a new concept.

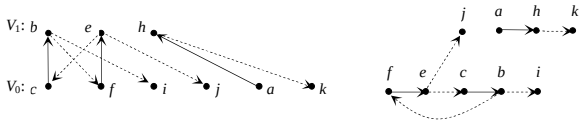
Definition 4 (*alternating graph*) Let M_i be a maximum matching of $G(V_i, V_{i-1}'; \boldsymbol{C}_i)$. The alternating graph $\vec{G}_i$ with respect to M_i is a directed graph with the following sets of nodes and arcs:

$V(\vec{G}_i) = V_i \cup V_{i-1}'$, and

$E(\vec{G}_i) = \{u \to v \mid u \in V_{i-1}', v \in V_i, \text{ and } (u, v) \in M_i\} \cup$

$\{v \to u \mid u \in V_{i-1}', v \in V_i, \text{ and } (u, v) \in \boldsymbol{C}_i \backslash M_i\}$. □

Example 2 Consider the graph shown in Fig. 1(a) once again. Its stratification is shown in Fig. 1(b). Assume that M_1 of $G_1 = G(V_1, V_0; \boldsymbol{C}_1)$ is a set of edges shown in Fig. 2(b). Then, the alternating graph with respect to M_1 is a directed graph shown in Fig. 5(a).



(a) (b)

Fig. 5 An alternating graph

It is redrawn in Fig. 5(b) for a clear explanation. □

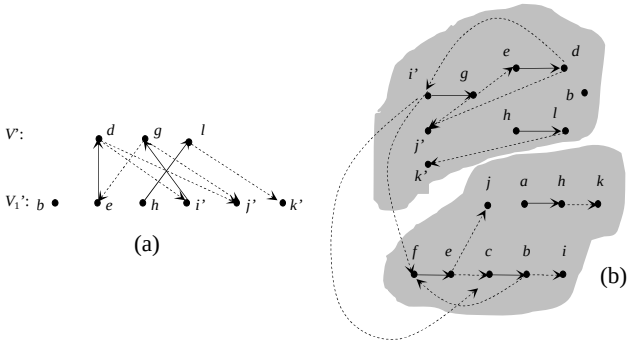
In order to resolve all the alternating-covered virtual nodes in V_i', we combine $\vec{G}_{i+1}$ and $\vec{G}_i$ by connecting each of these nodes in V_i' in $\vec{G}_{i+1}$ to some nodes in V_{i-1}' in $\vec{G}_i$ as follows:

- Let v_1, ..., v_k be all those alternating-covered virtual nodes in V_i' in $\vec{G}_{i+1}$ with $\beta(u_j \to v_j) = <i, W_j>$, where u_j is the parent of v_j along the corresponding chain and W_j is a set of nodes in V_{i-1}' in $\vec{G}_i$.
- For each v_j, connect v_j to every node in W_j (j = 1, ..., k).

We denote such a combined graph by $\vec{G}_{i+1} \oplus \vec{G}_i$.

For illustration, consider $G(V_2, V_1'; \boldsymbol{C}_2)$ shown in Fig. 2(c). Assume that the found maximum matching M_2 is as shown in Fig. 2(d). The alternating graph $\vec{G}_2$ (with respect to M_2) is a graph shown in Fig. 6(a).

Among the three virtual nodes i', j', and k', only i' is an alternating-covered virtual node. That is, $(d, i') \in M_2$ corresponds to a β-arc $d \to i'$ with $\beta(d \to i') = <1, \{c, f\}>$ (since there is an alternating path relative to M_1: i - b - c - e - f such that $d \to c \in E$ and f is reachable from d through e in V_1).



(a) (b)

Fig. 6 Illustration for combined graph

$\vec{G}_2 \oplus \vec{G}_1$ is shown in Fig. 6(b). It is formed by connecting i' in V_1' (in) to c and f in $V_0' = V_0$ (in) in terms of $\beta(d \to i') = <1, \{c, f\}>$.

We also notice that a node in $\vec{G}_{i+1}$ and a node in $\vec{G}_i$ may share the same node name. But they will be handled as different nodes. For example, node e in $\vec{G}_2$ and node e in $\vec{G}_1$ are different.

In order to resolve as many 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 an alternating-covered virtual node in V_i' (in $\vec{G}_{i+1}$) and ending at a actual free node (i.e., a free node that is not

virtual) in V_i' in $\vec{G}_{i+1}$, or ending at a free node in V_{i-1}' in $\vec{G}_i$. For example, to remove *i'*, we need to find a path in the above combined graph, as shown in Fig. 7(a).

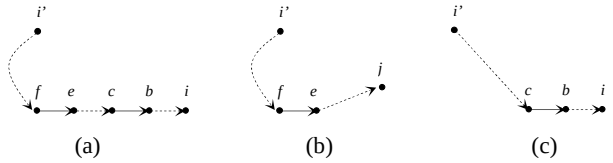


Fig. 7 Illustration for node-disjoint paths

By transferring the arcs on such a path, the corresponding virtual node can be resolved as follows:

- Let $v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_k$ be a found path. Transfer the arcs on the path.
- If v_k is a node in $\vec{G}_{i+1}$, we simply remove the corresponding virtual node v_1.
- If v_k is a node in $\vec{G}_i$, connect the parent of v_1 along the corresponding chain to v_2. Remove v_1.

For instance, by transferring the arcs on the path from *i'* to *i* (in $\vec{G}_1$) in Fig. 7(a), we will make *f* (in $\vec{G}_1$) free. Then, we connect *g* and *f*. Note that *g* is the parent of *i'* along the corresponding chain (see Fig. 3(a)).
In this way, we will change the chains shown in Fig. 5 to the chains shown in Fig. 3(b) with all the virtual nodes being removed. The number of chains is not increased.

IV. CONCLUSION

In this paper, a new algorithm for finding a minimal chain decomposition of a partially ordered set $\boldsymbol{S}$ is proposed. The algorithm needs $O(\kappa \cdot n^2)$ time and $O(\kappa \cdot n^2)$ space, where n is the number of the elements in $\boldsymbol{S}$, and κ is the size of the maximum antichain. The main idea of the algorithm is the concept of virtual nodes and the DAG stratification that generates a series of bipartite graphs which may contain virtual nodes. By executing Hopcropt-Karp's algorithm, we find a maximum matching for each of such bipartite graphs, which make up a set of disjoint chains. A next step is needed to resolve all the virtual nodes appearing on the chains to get the final result.

We also point out that our algorithm can be easily modified to a 0-1 network flow algorithm by defining a chain to be a path and accordingly changing the conditions for creating transitive and alternating arcs.

REFERENCES

[1] H. Alt, N. Blum, K. Mehlhorn, and M. Paul, Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/(\log n)})$, *Information Processing Letters*, 37(1991), 237 -240.

[2] A.S. Asratian, T. Denley, and R. Haggkvist, *Bipartite Graphs and their Applications*, Cambridge University, 1998.

[3] C. Chekuri and M. Bender, An Efficient Approximation Algorithm for Minimizing Makespan on Uniformly Machines, *Journal of Algorithms* 41, 212-224(2001).

[4] Y. Chen and Y.B. Chen, An Efficient Algorithm for Answering Graph Reachability Queries, in *Proc. 24th Int. Conf. on Data Engineering (ICDE 2008)*, IEEE, April 2008, pp. 892-901.

[5] Y. Chen, General Spanning Trees and Reachability Query Evaluation, in Proc. C^3S^2E 2009: 243-252, IEEE, 2009, pp. 243-252.

[6] Y. Chen and Y.B. Chen, Decomposing DAGs into spanning trees: A new way to compress transitive closures, in Proc. 27th Int. Conf. on Data Engineering (ICDE 2011), IEEE, April 2011, pp. 1007-1018.

[7] G.B. Dantzig and A. Hoffman, On a theorem od Dilworth, *Linear Inequalities and related systems* (H.W. Kuhn and A.W. Tucker, eds.) Annals of Math. Studies 38(1966), 207-214.

[8] R.P. Dilworth, A decomposition theorem for partially ordered sets, *Ann. Math.* **51** (1950), pp. 161-166.

[9] T. Gallai and A.N. Milgram, Verallgemeinerung eines Graphentheoretischen Satzes von Reedei. *Acta Sci. Math. Hung*., 21(1960), 429-440.

[10] D.R. Fulkerson, Note on Dilworth's embedding theorem for partially ordered sets, *Proc. Amer. Math. Soc*. 7(1956), 701-702.

[11] J. E. Hopcroft, and R.M. Karp, An $n^{2.5}$ algorithm for maximum matching in bipartite graphs, *SIAM J. Comput*. 2(1973), 225-231.

[12] H. V. Jagadish, "A Compression Technique to Materialize Transitive Closure," *ACM Trans. Database Systems*, Vol. 15, No. 4, 1990, pp. 558 - 598.

[13] A.V. Karzanov, Determining the Maximal Flow in a Network by the Method of Preflow, *Soviet Math. Dokl*., Vol. 15, 1974, pp. 434-437.

[14] E. L. Lawler, *Combinatorial Optimization and Matroids*, Holt, Rinehart, and Winston, New York (1976).

[15] R.-D. Lou, M. Sarrafzadeh, An optimal algorithm for the maximum two-chain problem, *SIAM J. Disc. Math*. **5**(2), 1992, pp. 285-304.

[16] V.M. Malhotra, M.P. Kumar, and S.N. Maheshwari, An $O(|V|^3)$ Algorithm For Finding Maximum Flows in Networks, Computer Science Program, Indian Institute of Technology, Kanpur 208016, India, 1978.

[17] M.A. Perles, A proof of Dilworth's decomposition theorem for partially ordered sets, *Israel J. of Math*. 1(1963), 105-107.

[18] H. Tverberg, On Dilworth's decomposition theorem for partially ordered sets, *J. Comb. Th*. 3(1967), 305-306.

[19] D. Coppersmith, and S. Winograd. Matrix multiplication via arithmetic progression. *Journal of Symbolic Computation*, vol. 9, pp. 251-280, 1990.

[20] S. Even, *Graph Algorithms*, Computer Science Press, Inc., Rockville, Maryland, 1979.

[21] L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Communication of the ACM* 21(7), July 1978, 95-114.

[22] H. Goeman, Time and Space Efficient Algorithms for Decomposing Certain Patially Ordered Sets, PhD thesis, Department of Mathematics-Science, Rheinischen Friedrich-Wilhelms Universität Bonn, Germany, Dec. 1999.

[23] R. Tarjan: Depth-first Search and Linear Graph Algorithms, *SIAM J. Compt*. Vol. 1. No. 2. June 1972, pp. 146 -140.

[24] H.S. Warren, A Modification of Warshall's Algorithm for the Transitive Closure of Binary Relations, *Commun. ACM* 18, 4 (April 1975), 218 - 220.