Optimizing User Views for Workflows

Olivier Biton
University of Pennsylvania
Philadelphia, PA 19104
biton@cis.upenn.edu

Susan B. Davidson[*]
University of Pennsylvania
Philadelphia, PA 19104
susan@cis.upenn.edu

Sanjeev Khanna[†]
University of Pennsylvania
Philadelphia, PA 19104
sanjeev@cis.upenn.edu

Sudeepa Roy[‡]
University of Pennsylvania
Philadelphia, PA 19104
sudeepa@cis.upenn.edu

ABSTRACT

A technique called *user views* has recently been proposed to focus user attention on *relevant* information in response to provenance queries over workflow executions [1, 2]: Given user input on what modules in the workflow specification are *relevant* to the user, a user view is a concise representation that clusters together modules to create a small number of composite modules (or clusters) such that (1) each composite module in a user view contains at most one relevant (atomic) module, thus assuming the "meaning" of that module; and (2) no control or data dependencies (either direct or indirect) are introduced (soundness) or removed (completeness) between relevant modules. The goal is to find a user view with a smallest number of composite modules.

We show that for workflow specifications that are general graphs, regardless of the number of distinct modules in the input workflow and the structure of interaction between them, there always exists a user view of size at most $(2^{k-1} - k)^2 + k$, where k is the number of relevant modules. Moreover, a good user view with at most $(2^{k-1} - k)^2 + k$ clusters can be computed in polynomial time in the size of the graph. We also show that this upper bound is tight. Thus in general graphs, the number of composite modules can be exponentially large in k even in an optimum user view for the specification. We also give a characterization of a *good* user view in terms of structural properties of each cluster in the user view.

However, for *series-parallel* workflow graphs, we show that there is always a user-view with at most $2k - 3$ composite modules; further, there exist series-parallel graphs where every user view requires at least $2k - 3$ composite modules. Such graphs capture the structure of many scientific and other workflows that we have encountered in practice. For this class of graphs, we give a *simple, linear time* algorithm for constructing an *optimum* user view for a given specification.

[*]Supported by NSF Awards IIS-0513778, IIS-0629846 and SEI-II0612177.
[†]Supported in part by a Guggenheim Fellowship, an IBM Faculty Award, and by NSF Award CCF-0635084.
[‡]Supported by NSF Award IIS-0803524.

1. INTRODUCTION

Workflow management systems have become increasingly popular as a way of specifying and executing data-intensive scientific analyses (*e.g.*, myGrid/Taverna [14], Kepler [5], VisTrails [12], and Chimera [11]) as well as business processes [7]. In such systems, a workflow can be graphically designed by chaining together modules, where each module may take as input data from previous modules, parameter settings, and data coming from external data sources. The workflow can then be executed multiple times, using different initial input data and parameter settings, and potentially generating a large amount of intermediate and final data products.

Due to the explosion of data being produced by these "in-silico" experiments, provenance support in workflow systems has become of paramount importance, as evidenced by recent workshops [3, 13] and surveys [4, 15]. By maintaining the provenance of data, its validity and reliability can be understood and results be made reproducible. Many workflow systems are therefore beginning to provide tools to capture, query, and manage provenance (*e.g.*, COMAD-Kepler [6] and VisTrails [12]). In these and other systems, the provenance of a data object is defined as the dependency graph of module executions, their parameters, and the data objects passed between module executions [9]. This information can be gleaned from the log of operations performed, or may be explicitly stored by the system.

However, since a workflow execution (or *run*) may comprise many module executions (*steps*) and intermediate data objects, the amount of information provided in response to a provenance query can be overwhelming.

Furthermore, many of the modules and intermediate data products may not be of interest to the user, for example, modules that represent reformatting of data. Recent work (ZOOM [2]) has therefore presented a technique called "user views" for focusing user attention on provenance information that is *relevant* to the user. The technique takes as input a workflow specification and a set of relevant modules, and creates a set of *composite* modules, each of which represents a sub-workflow. The set of composite modules forms the user's view of the workflow specification, and is used to present provenance information that is relevant to the user by hiding the intermediate data and sequence of module executions within each composite module.

More formally, a *user view* is a partition of the workflow modules. It induces a higher level workflow in which nodes represent composite modules in the partition and edges are induced by dataflow between modules in different composite modules. Provenance information is seen by the user with respect to the flow of data between modules in his view. In [1, 2], views are automatically constructed given input on what modules the user finds relevant such that (1) a composite module contains at most one relevant (atomic) module, thus assuming the "meaning" of that module; (2) no data dependencies (either direct or indirect) are introduced or removed between relevant modules; and (3) the view cannot be made smaller by combining two composite modules. In this way, the meaning of the original workflow specification is preserved, and relevant provenance information is provided to the user.

As an example, consider the workflow specification in Figure 1 which represents a common task in modern biology: Phylogenomic tree construction using sequence, annotation and functional data. This workflow first accepts a set of entries selected by the user from a database and supplies them to the `Split Entries` module. The entries are split into sequence information, which is passed to the `Align Sequences`, and annotation information, which is passed to the `Curate Annotations` module. Additional `Functional data` may be extracted from a database. The annotation, alignment, and functional data are then reformatted and passed to the `Construct tree` module, which creates the phylogenomic tree. One user of the system may indicate that the `Align Sequences` and `Construct tree` modules are relevant, but that the formatting steps, and annotation and functional steps are not. Given this input, his user view would be constructed as $\{C_1, C_2, C_3, C_4\}$, where C_1 and C_2 are the composite modules indicated by dotted boxes. When asking about the provenance of a final tree, only the information passed between (C_1 , C_2) and (C_2, C_3) would be displayed.

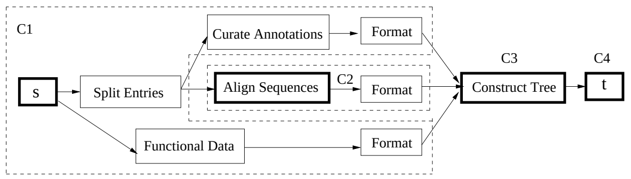


Figure 1: Phylogenomic workflow

However, the algorithm of [2] does not always guarantee a minimum size user view, and leaves it as an open problem as to whether there are efficient, optimal algorithms for constructing a user view given a workflow specification and a set of relevant modules. This is the problem we address in this paper.

Contributions. The goal of a *good user view* is to faithfully represent the control and data flow between relevant nodes while abstracting away information pertaining to the non-relevant nodes. We start by formalizing the properties of a good user view. These properties enforce global constraints so that the structure of paths in the user view accurately reflects the underlying structure in the input graph.

Our first result is a characterization theorem for good user views that shows that certain local conditions on the structure of each cluster in a user view, precisely capture the global constraints on the structure of paths. This characterization theorem holds for general specification graphs, and plays a central role in the design and analysis of our algorithms for computing good user views.

Then we study the problem of constructing a minimum user view for *series-parallel* workflow graphs. Series-parallel workflow graphs, such as the one in Figure 1, capture the structure of many scientific workflows we have encountered in practice (see *www.myexperiment.org* for examples). Furthermore, most scientific workflow systems allow only stateless, functional behavior, and do not allow looping [10]. For this common case of workflows, we give a linear time algorithm, **SP-View**, to create an optimum user view given a specification and a set of k relevant modules. We also show that there is always a user view with at most $2k - 3$ composite classes, and that this bound is tight. That is, there exist directed series-parallel graphs for which any user view must have at least $2k - 3$ composite modules. Our analysis relies on an elegant forbidden subgraph based characterization of directed series-parallel graphs, and develops several novel insights about the structure of control and data flow in these graphs.

We then again turn our attention to general workflows. While the problem of finding an optimum user view remains open for this class of workflows, we show the following, somewhat surprising result. Regardless of the number of distinct modules in the input workflow and the structure of interaction between them, there always exists a user view of size at most $(2^{k-1} - k)^2 + k$. Moreover, we show that this upper bound is tight. We also compare the performance of the ZOOM algorithm on series-parallel workflow graphs with our algorithm, and show that **SP-View** has considerably better time complexity than ZOOM on this class of graphs: **SP-View**has complexity O(n) while ZOOM has complexity O(n^2), where n is the number of modules in the specification.

Thus, the following framework can be used for finding a good user view given a specification S and a set of relevant nodes: First, test whether S is a series parallel graph. If it is, then use **SP-View** to find an *optimum* good user view. If not (a rare case), then use ZOOM to find a good user view that is *optimal* in the sense that no two composite modules can be combined to create a smaller good user view.

Outline. Section 2 presents the model of workflows. Section 3 characterizes "good" user views. Section 4 presents a linear-time algorithm for finding an optimum (good) user view for workflows represented by series-parallel graphs. General graphs are considered in Section 5, together with an analysis of the ZOOM algorithm. Section 6 compares the performance of **SP-View** with ZOOM on series-parallel graphs. We conclude in Section 7.

2. MODEL AND DEFINITIONS

Given a graph G, we will use $V(G)$ and $E(G)$ to denote the set of the vertices and the edges in G respectively.

2.1 Workflow Specifications

Definition 1. A **workflow specification** (or simply a *specification*) (G, s, t, R) consists of

- a directed graph G such that each node $v \in V(G)$ denotes a unique module in the workflow,
- a source node $s \in V(G)$ and a sink node $t \in V(G)$ such that s has no incoming edges, t has no outgoing edges and every node $v \in V(G)$ lies on some $s \rightsquigarrow t$ path in G, and
- a set $R \subseteq V(G)$ of *relevant modules* in the workflow, where by convention, $s, t \in R$.

The special nodes s and t correspond to the input and the output modules in the workflow. We will use *NR* to denote the set of *non relevant* modules in the workflow (i.e. *NR* $= V(G) \setminus R$) and k to denote size of R.

A node $v \in V(G)$ is called an *R-node*, if $v \in R$ and an *NR-node* if $v \in NR$. For any edge $(u, v) \in E(G)$, u is called a *predecessor* of v and v is called a *successor* of u. A predecessor u of v in G is an *R-predecessor* of v if $u \in R$. *NR-predecessor*, *R-successor* and *NR-successor* are defined similarly. A path p of length ≥ 1 is called an *elementary path* if no intermediate node on p is an R-node.

2.2 Good User Views

A user view for a workflow specification G is another directed graph H such that the nodes of H are "clusters" of nodes of G and the edges of H correspond to edges between nodes in different clusters. The formal definition is given below.

Definition 2. A **user view** for a workflow specification (G, s, t, R) is a pair (H, ϕ) where H is a directed graph and $\phi : V(G) \rightarrow V(H)$ is a homomorphism such that

- if $(u, v) \in E(G)$ then $(\phi(u), \phi(v)) \in E(H)$, and conversely, if $(C, C') \in E(H)$ then there exists $u \in \phi^{-1}(C)$, $v \in \phi^{-1}(C')$ with $(u, v) \in E(G)$.
- A self-loop (C, C), $C \in V(H)$ is preserved in $E(H)$ if and only if there exists a cycle ρ such that for each vertex u on ρ, $\phi(u) = C$ and there exists an R-node on ρ.

We will refer to a node $C \in V(H)$ as a *composite module* or simply a *cluster*; it represents all modules $u \in V(G)$ such that $\phi(u) = C$. We say an edge $e = (u, v)$ in G is an *origin* of an edge $e' = (C, C')$ in H (or that, e *induces* e') if $\phi(u) = C$ and $\phi(v) = C'$. Note that an edge in H can have multiple origins in G; equivalently, multiple edges in G may induce the same edge in H.

We discard any self-loop in H that does not correspond to a cycle containing an R-node in G. In this case we assume that all the edges $(u, v) \in E(G)$ on cycle ρ induce the self loop. This preserves the presence of non-trivial elementary paths from an R-node to itself and allows the execution of a composite module containing an R-node r multiple times if multiple executions of the module r were possible in the original specification.

The *size* of a user view is the number of composite modules in it, that is, $|V(H)|$. Our goal is to find a minimum size user view H for a given specification while requiring that the view obey certain properties that we describe next.

We start by extending the notions of relevant nodes and elementary paths to a user view H. We will say a cluster $C \in V(H)$ is a *relevant cluster* or an *R-cluster* if $\phi^{-1}(C) \cap R \neq \emptyset$. Similarly, we will say a cluster $C \in V(H)$ is *non-relevant cluster* or an *NR-cluster* if $\phi^{-1}(C) \cap R = \emptyset$. A path p of length ≥ 1 in H is called an *elementary path* if no intermediate cluster on p is an R-cluster.

Definition 3. A user view (H, ϕ) for a workflow specification (G, s, t, R) is said to be good if it satisfies the following properties [2]:

1. H is *well-formed*, that is, for any node $C \in V(H)$, the set $\phi^{-1}(C)$ contains at most one node from R.
2. H is *sound* w.r.t. data flow, that is, for every edge e' on an elementary path that connects an R-cluster C to an R-cluster C' in H, each origin of the edge e' lies on an elementary path from an R-node r to an R-node r' in G such that $r \in \phi^{-1}(C), r' \in \phi^{-1}(C')$.
3. H is *complete* w.r.t data flow, that is, for every edge $e = (u, v)$ on an elementary path from an R-node r to an R-node r' in G, either $\phi(u) = \phi(v)$, or the edge e' induced by e lies on an elementary path from $\phi(r)$ to $\phi(r')$ in H.

We now state the motivation behind the above properties. As we said before, the R-nodes are relevant to the user, and the goal is to be able to *see* the workflow relationship among the relevant nodes. Thus an R-cluster containing an R-node should take on the *meaning* or *role* of the R-node that it contains. So to preserve the identity of the R-clusters, we require that each cluster in the workflow be well-formed. Since we are interested in preserving the "dependency" between relevant modules in a good user view, two R-clusters should be connected by an elementary path in the user view if and only if the corresponding R-nodes are connected by an elementary path in the given specification. The soundness property above ensures that the user view does not create any new elementary paths between a pair of R-clusters, that is, elementary paths that did not exist between the corresponding R-nodes in the original workflow specification. The completeness property, on the other hand, ensures that every elementary path connecting a pair of R-clusters is preserved in the user view, modulo merging of nodes along such a path into a composite module. Thus the provenance information provided by a good user view is always consistent with the original specification.

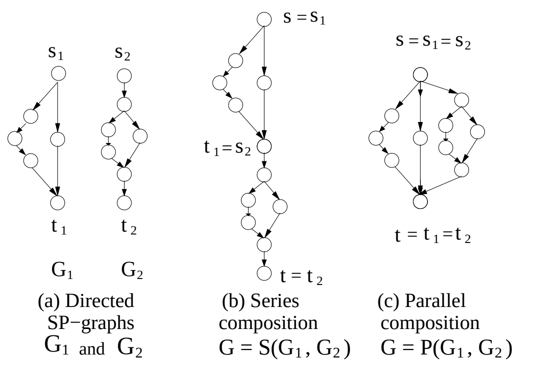
Note that the above properties trivially hold in any given specification G, hence the specification G is always a good user view of itself with $\phi(v) = \{v\}$ for each $v \in V(G)$. In any user view H, it is easy to see that for each $v \in V(G)$, $\phi(v)$ is on a path from $\phi(s)$ to $\phi(t)$, as v is on a path from s to t in G. Later in Corollary 2 we will also show that in a good user view, $\phi(s)$ acts as the source cluster and $\phi(t)$ acts as the sink cluster, i.e. $\phi(s)$ and $\phi(t)$ do not have any incoming edge and any outgoing edge respectively.

2.3 Series-Parallel Graphs

A natural class of workflows is *directed two terminal series-parallel* (SP) graphs defined below.

Definition 4. A **directed two terminal series-parallel graph** is a directed multigraph G with a single source s and a single sink t (two terminals) that can be produced by a sequence of the following operations:

- **Basic SP-graph:** Create a new graph consisting of a single edge directed from s to t.
- **Series Composition:** Given two SP-graphs G_1 and G_2 with sources s_1, s_2 and sinks t_1, t_2 respectively, form a new graph $G = S(G_1, G_2)$ by identifying $s = s_1$, $t_1 = s_2$ and $t = t_2$.
- **Parallel Composition:** Given two SP-graphs G_1 and G_2 with sources s_1, s_2 and sinks t_1, t_2 respectively, form a new graph $G = P(G_1, G_2)$ by identifying $s = s_1 = s_2$ and $t = t_1 = t_2$.



s₁
s₂
t₁
t₂
G₁
G₂
(a) Directed SP−graphs G₁ and G₂
s = s₁
t₁=s₂
t = t₂
(b) Series composition G = S(G₁, G₂)
s = s₁= s₂
t = t₁=t₂
(c) Parallel composition G = P(G₁, G₂)

Figure 2: An example of series and parallel compositions

Examples of series and parallel compositions are shown in Figure 2. The following observations can be made from the inductive definition above.

Observation 1. Let G be a directed SP graph. Then,

1. Each node $v \in V(G)$ is on a path from the start node s to the sink node t, and
2. G is acyclic (that is, it is a DAG).

Definition 5. A directed graph G_1 is said to contain a subgraph[1] **homeomorphic** to a directed graph G_2 if G_2 can be obtained from G_1 by a sequence of operations: (1) removal of an edge, or (2) replacement of two edges of the form (u, w) and (w, v) by the edge (u, v) when w has indegree $= 1$ and outdegree $= 1$.

We will use the following characterization of two terminal directed SP graphs from [16].

Theorem 1. Let G be a directed acyclic graph such that G has a unique source node s and a unique sink node t, and each node $v \in V(G)$, is on a path from s to t. Then G is a two terminal directed SP graph if and only if G does not contain a subgraph homeomorphic to the graph $\mathcal{F}$ given in Figure 3.

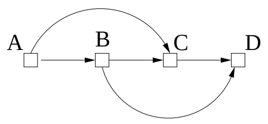


Figure 3: The forbidden subgraph $\mathcal{F}$ of directed two-terminal SP graphs

Next we derive a necessary condition from the above characterization for a two terminal directed acyclic graph to be a two terminal directed SP graph. We will use this condition in analyzing our algorithms.

Definition 6. For a directed acyclic graph G and two nodes $v_1, v_2 \in G$, $v_1 \neq v_2$,

- the **last common ancestor set** of v_1, v_2, denoted by $\text{LCA}(v_1, v_2)$, is defined as $\text{LCA}(v_1, v_2) = \{u_1 \in V(G) \mid \exists$ internally vertex-disjoint paths $p_1 = u_1 \rightsquigarrow v_1$ and $p_2 = u_1 \rightsquigarrow v_2$ in $G\}$
- the **first common descendant set** of v_1, v_2, denoted by $\text{FCD}(v_1, v_2)$), is defined as $\text{FCD}(v_1, v_2) = \{u_2 \in V(G) \mid \exists$ internally vertex-disjoint paths $p_1 = v_1 \rightsquigarrow u_2$ and $p_2 = v_2 \rightsquigarrow u_2$ in $G\}$.

The paths p_1, p_2 can be trivial paths of length 0, and in that case u_1(or u_2) $\in \{v_1, v_2\}$.

Note that if G has a unique source s and a unique sink t, and each node $v \in V(G)$ is on a path from s to t, then for any two nodes $v_1, v_2 \in V(G)$, the sets $\text{LCA}(v_1, v_2)$ and $\text{FCD}(v_1, v_2)$ are non empty.

[1]A directed graph G_1 is a *subgraph* of a directed graph G if $V(G_1) \subseteq V(G)$ and $E(G_1) \subseteq E(G)$.

Lemma 1. Let G be a directed acyclic graph with a unique source node s and a unique sink node t such that each node $v \in V(G)$ is on a path from s to t. Suppose G contains four distinct nodes $W, X, Y, Z \in V(G)$ such that

1. $\exists U_{WX} \in \text{LCA}(W, X)$ with $U_{WX} \neq W$ and $\exists U_{YZ} \in \text{FCD}(Y, Z)$ with $U_{YZ} \neq Z$, and
2. there exist pairwise vertex-disjoint paths $p_W = U_{WX} \rightsquigarrow W$, $p_X = U_{WX} \rightsquigarrow X$, $p_{WY} = W \rightsquigarrow Y$, $p_{WZ} = W \rightsquigarrow Z$, $p_{XZ} = X \rightsquigarrow Z$, $p_Y = Y \rightsquigarrow U_{YZ}$, and $p_Z = Z \rightsquigarrow u_{YZ}$.

Then G is not a two terminal directed SP graph.

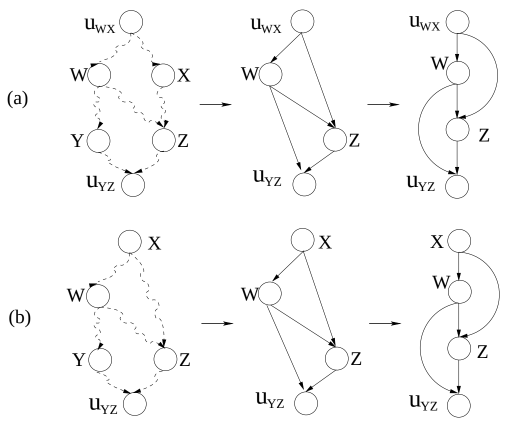


Figure 4: Forming forbidden subgraph $\mathcal{F}$

PROOF. First consider the case when $U_{WX} \neq X$ and $U_{YZ} \neq Y$.

1. Delete all other edges in G except these paths to form a subgraph G' of G (ref. Figure 4 (a)).
2. Note that all internal vertices on the paths $U_{WX} \rightsquigarrow X \rightsquigarrow Z$ and $W \rightsquigarrow Y \rightsquigarrow U_{YZ}$ have indegree = 1 and outdegree = 1 and so they can be contracted to edges (U_{WX}, Z) and (W, U_{YZ}) respectively.
3. Again all internal vertices on the paths $U_{WX} \rightsquigarrow W$ and $Z \rightsquigarrow U_{YZ}$ have indegree = 1 and outdegree = 1 and so they can be contracted to edges (U_{WX}, W) and (W, U_{YZ}) respectively.

This gives a subgraph homeomorphic to $\mathcal{F}$ given in Figure 3.

Next consider the case $U_{WX} = X$ and $U_{YZ} \notin \{Y, Z\}$. Again we can obtain the subgraph homeomorphic to $\mathcal{F}$ in G (see Figure 4(b)). The case when $U_{WX} \notin \{W, X\}$ and $U_{YZ} = Y$ can be handled analogously.

Finally, if $U_{WX} = X$ and $U_{YZ} = Y$, then we can simply contract each of the vertex-disjoint paths $p_W = U_{WX} \rightsquigarrow W$, $p_{WY} = W \rightsquigarrow Y$, $p_{WZ} = W \rightsquigarrow Z$, $p_{XZ} = X \rightsquigarrow Z$, and $p_Z = Z \rightsquigarrow U_{YZ}$ to single edges and obtain the forbidden subgraph $\mathcal{F}$. $\square$

Now we state a stronger necessary condition than given in Lemma 1 for a directed acyclic graph to be a two terminal directed SP graph. The proof is deferred to the full version of the paper.

Lemma 2. Let G be a directed acyclic graph with a unique source s and a unique sink t such that each node $v \in V(G)$, is on an s to t path. Suppose G contains four distinct nodes $w, x, y, z \in V(G)$ such that

1. no $w \rightsquigarrow x$ or $y \rightsquigarrow z$ path exists.
2. there exist paths $p_{wy} = w \rightsquigarrow y$, $p_{wz} = w \rightsquigarrow z$, $p_{xz} = x \rightsquigarrow z$ such that the path pairs $\langle p_{wy}, p_{wz} \rangle$ and $\langle p_{wz}, p_{xz} \rangle$ are vertex disjoint.

Then G is not a two terminal directed SP graph.

We will also use the following property of directed SP graphs in developing our algorithms in Section 4.

Lemma 3. In a directed SP graph G, if a node $v \in V(G)$ has ≥ 2 predecessors, then the last predecessor of v in any topological ordering of the vertices of G can not have another successor $v' \neq v$.

PROOF. Let us list the predecessors in any topological order $u_1, \cdots, u_\ell$, $\ell \geq 2$, and assume the contradiction that u_ℓ has another successor $v' \neq v$. Note that $u_{\ell-1}, u_\ell, v, v'$ are four pairwise distinct points. As the u_js are listed in topological order, there cannot be any path from u_ℓ to $u_{\ell-1}$. Since the edge (u_ℓ, v') exists and u_ℓ is the last predecessor in topological order, no v' to v path can exist. Also the edges $(u_{\ell-1}, u_\ell)$, (u_ℓ, v) and (u_ℓ, v') trivially form three mutually vertex disjoint paths. So we can map these four nodes to w, x, y, z given in Lemma 2 as $u_\ell \to w$, $u_{\ell-1} \to x$, $v' \to y$ and $v \to z$. This contradicts the assumption that G is a directed SP graph. $\square$

3. A CHARACTERIZATION OF GOOD USER VIEWS

We next derive a useful characterization of good user views for any general directed graph given as a specification. Throughout this section, we will assume that (H, ϕ) is a user view for a given workflow specification (G, s, t, R).

Let us first define the $\mathsf{R}^-(v)$ and $\mathsf{R}^+(v)$ sets for a node $v \in V(G)$.

Definition 7. • For an *NR*-node $v \in V(G)$, $\mathsf{R}^-(v) = \{r \in R \mid$ there is an elementary path from r to v in $G\}$ and $\mathsf{R}^+(v) = \{r \in R \mid$ there is an elementary path from v to r in $G\}$.

- For an R-node $r \in V(G)$, $\mathsf{R}^-(r) = \mathsf{R}^+(r) = \{r\}$.

The intuition behind the above definition is that, if there is an elementary path p from u to v, and both u and v are NR-nodes, then we would like to have $\mathsf{R}^-(u) \subseteq \mathsf{R}^-(v)$ and $\mathsf{R}^+(u) \supseteq \mathsf{R}^+(v)$, i.e. (i) v should "inherit" the R^- set of u and (ii) u should inherit the R^+ set of v along the elementary path p. But if u is an R-node and v is an NR-node, and r is another R-node such that there is an elementary path p' from r to u, then v cannot inherit r in $R^-(v)$ along the elementary path p, as the path p' followed by p from r to v is not elementary any more, though in this case $u \in R^-(v)$. Definition 7 makes the inheritance of R^- and R^+ sets consistent.

Note that the sets $\mathsf{R}^-(v)$ and $\mathsf{R}^+(v)$ for any node $v \in V(G)$ are non empty since all nodes in $V(G)$ are on some path from s to t in G. We also extend the functions R^- and R^+ to the clusters in H by considering them as subsets of nodes in $V(G)$.

For a node $C \in V(H)$, we define,
$\mathsf{R}^-(C) = \bigcup_{v\in\phi^{-1}(C)} \mathsf{R}^-(v)$ and $\mathsf{R}^+(C) = \bigcup_{v\in\phi^{-1}(C)} \mathsf{R}^+(v)$.

Also, for any cluster $C \in V(H)$, we define

- $\mathsf{IN}(C) = \{v \in \phi^{-1}(C) \mid \exists v'$ such that $(v', v) \in E(G)$, $\phi(v') \neq C\}$ and
- $\mathsf{OUT}(C) = \{v \in \phi^{-1}(C) \mid \exists v'$ such that $(v, v') \in E(G)$, $\phi(v') \neq C\}$.

3.1 Structure of *R*-clusters in Good User Views

Definition 8. An R-cluster $C \in V(H)$ containing $r \in R$ is called R-**Valid** if $\forall v \in \mathsf{OUT}(C)$, $\mathsf{R}^-(v) = \{r\}$, and $\forall v \in \mathsf{IN}(C)$, $\mathsf{R}^+(v) = \{r\}$.

Since for an R-node r, we have defined $R^-(r) = R^+(r) = \{r\}$, the above definition also holds for $v = r$. If each R-cluster $C \in V(H)$ is R-Valid, H is called an *R-Valid user view.*

Lemma 4. Any well-formed and complete user view H is also an R-Valid user view.

Proof. Note that since H is well-formed, each $C \in V(H)$ contains at most one R-node. Assume by way of contradiction, that H is not R-Valid. Then there is an R-cluster $C \in V(H)$, containing an R-node $r \in R$, such that either there exists $v \in \mathsf{OUT}(C)$ with $\mathsf{R}^-(v) \neq \{r\}$, or there exists $v \in \mathsf{IN}(C)$ with $\mathsf{R}^+(v) \neq \{r\}$. Assume it is the former, so there exists $r_1 \neq r \in \mathsf{R}^-(v)$, the other case can be handled analogously. As H is well-formed, $\phi(r_1) \neq \phi(r)$. Since $v \in \mathsf{OUT}(C)$, there exists an edge $e = (v, v') \in E(G)$ such that $\phi(v') \neq C$ and therefore e induces the edge $e' = (\phi(v), \phi(v')) \in E(H)$. Consider the case when v' is an NR-node and let $r_2 \in \mathsf{R}^+(v')$ (recall that the set $R^+(v)$ is non empty).
The edge e is on an elementary path $r_1 \rightsquigarrow r_2$ in G and since H satisfies the completeness property, e' must lie on an elementary path $\phi(r_1) \rightsquigarrow \phi(r_2)$ in H. But this is impossible since $\phi(v) = \phi(r)$ is an R-cluster. A contradiction. Note that the above argument holds when $r_2 = r$ or $r_2 = r_1$ or $r \in \{s, t\}$. If v' is an R-node, assume $r_2 = v'$ and the above argument holds again. Thus H must be R-Valid. $\square$

The following corollary easily follows from the above lemma.

Corollary 1. For any NR-node $v \in V(G)$ such that $|\mathsf{R}^-(v)| \geq 2$ and $|\mathsf{R}^+(v)| \geq 2$, the cluster $\phi(v)$ can never be an R-cluster in any good user view.

Proof. Suppose not. Then there exists a good user view H and an NR-node $v \in V(G)$ such that $C = \phi(v)$ contains a R-node r, and $|\mathsf{R}^-(v)| \geq 2$ and $|\mathsf{R}^+(v)| \geq 2$. If $v \in \mathsf{IN}(C)$ then it violates Lemma 4 since $\mathsf{R}^+(v) \neq \{r\}$. Similarly, if $v \in \mathsf{OUT}(C)$ then also it violates Lemma 4 since $\mathsf{R}^-(v) \neq \{r\}$. Finally, if $v \notin \mathsf{IN}(C) \cup \mathsf{OUT}(C)$, then let $r_1 \neq r \in \mathsf{R}^-(v)$. As H is good and therefore well-formed, $C \neq \phi(r_1)$. Now let $u \in \mathsf{IN}(C)$ be such that u is on an elementary path $r_1 \rightsquigarrow v$ in G. Since there is an elementary path from u to v, $R^+(u) \supseteq R^+(v)$ and as $|R^+(v)| \geq 2$, $R^+(u) \neq \{r\}$, contradicting Lemma 4 once again. $\square$

Note that self-loops are not taken into account while defining $\mathsf{IN}(C)$ and $\mathsf{OUT}(C)$ for a $C \in V(H)$. If C is an R-cluster with a self-loop in a good user view, a node u with $\phi(u) = C$ on a cycle that induces the self-loop can have $|\mathsf{R}^-(u)| \geq 2$ or $|\mathsf{R}^+(u)| \geq 2$ (see Figure 5).

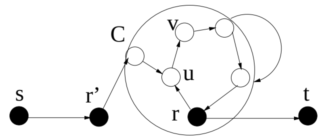


Figure 5: (u, v) induces the self-loop in C, $\mathsf{R}^-(u) = \mathsf{R}^-(v) = \{r, r'\}$ but the user view is R-Valid and good

A direct consequence of the R-Valid property is that $\phi(s)$ is the source cluster and $\phi(t)$ is the sink cluster in any good user view as stated in the next corollary. The proof is straightforward and hence omitted.

Corollary 2. In a good user view, $\phi(s)$ (resp. $\phi(t)$) has no incoming (resp. outgoing) edges.

3.2 Structure of *NR*-clusters in Good User Views

Definition 9. An NR-cluster $C \in V(H)$ is called NR-**Valid** if $\forall v \in \mathsf{OUT}(C)$, $\mathsf{R}^-(v) = \mathsf{R}^-(C)$, and $\forall v \in \mathsf{IN}(C)$, $\mathsf{R}^+(v) = \mathsf{R}^+(C)$.

A user view is called an *NR-Valid user view* if each NR-cluster in the user view is NR-Valid.

Lemma 5. Any well-formed, sound and complete user view H is also an NR-Valid user view.

Proof. Assume by way of contradiction that H is not NR-Valid. Here we consider the case when there exists an NR-cluster $C \in V(H)$ and an R-node $r \in \mathsf{R}^-(C)$ such that for some $v \in \mathsf{OUT}(C)$, the node $r \notin \mathsf{R}^-(v)$ (by definition, $R^-(C) \supseteq \mathsf{R}^-(v)$). The other case can be handled analogously. Since $r \in \mathsf{R}^-(C)$, $\exists u \in \mathsf{IN}(C)$ such that $r \in \mathsf{R}^-(u)$ and there exists an edge $e_u = (u', u)$ on an elementary path from r to u, where $\phi(u') \neq C$. This edge induces an edge $e'_u = (\phi(u'), C)$ in H. As the set $R^+(u)$ is non empty there exists $r_u \in \mathsf{R}^+(u)$ and as H is complete, e'_u is on an elementary path p_u from $\phi(r)$ to $\phi(r_u)$.

Similarly, as $v \in \mathsf{OUT}(C)$, there exists an edge $e_v = (v, v') \in E(G)$ such that $\phi(v') \neq C$ and e induces the edge $e'_v = (C, \phi(v'))$ in H. Consider the case when v' is an NR-node and $r' \in \mathsf{R}^+(v')$. Again as $R^-(v)$ is non empty, $\exists r_v \in \mathsf{R}^-(v)$, $r_v \neq r$. The edge e_v was on an elementary path from r_v to r' in G, so e'_v is on an elementary path p_v from $\phi(r_v)$ to $\phi(r')$ in the complete user view H.

Since H is well formed and $r_v \neq r$, $\phi(r_v) \neq \phi(r)$. The elementary paths p_u (from $\phi(r)$ to $\phi(r_u)$) and p_v (from $\phi(r_v)$ to $\phi(r')$) intersect at the cluster C and thus creates an elementary path from $\phi(r)$ to $\phi(r')$ in H with the edge $e'_v = (C, \phi(v'))$ lying on this path. But as $r \notin \mathsf{R}^-(v)$, an origin e_v of e'_v is not on an elementary path from r to r'. This contradicts the fact that H is a sound user view. The above argument also holds if any two of r, r', r_u, r_v are same node, with the restriction that $r \neq r_v$. If v' is an R-node, consider r' as v' and the above argument holds once again. Thus H must be NR-Valid. $\square$

3.3 Characterization of Good User Views using R-Valid and NR-Valid Properties

The following lemma gives a sufficient condition for a user view to be a good user view and therefore, in combination with Lemma 4 and Lemma 5, completes the characterization of a good user view given by Theorem 2.

Lemma 6. A well-formed user view (H, ϕ) that is both R-Valid and NR-Valid, is a good user view.

Proof. We start by arguing the soundness of H. Assume by way of contradiction that H is both R-Valid and NR-Valid but it is not sound. Then there exists an edge $e' = (C, C') \in E(H)$ and R-nodes $r_1, r_2 \in R$ such that e' is on an elementary path connecting cluster $\phi(r_1)$ to cluster $\phi(r_2)$ but an edge $e = (v_1, v_2) \in E(G)$ that is origin of e' is not on an elementary path from r_1 to r_2 in G. We will now show that $r_1 \in \mathsf{R}^-(v_1)$. Consider an elementary path in H from $\phi(r_1)$ to C, say, $\phi(r_1)$, $C_{i_1}, \cdots, C_{i_l}$, C and let this path be induced by the edges $(v^+_{r_1}, v^-_{i_1})$, $(v^+_{i_1}, v^-_{i_2})$, $\cdots$, $(v^+_{i_{l-1}}, v^-_{i_l})$, $(v^+_{i_l}, v^-_1)$, where $v^+_{r_1} \in \mathsf{OUT}(\phi(r_1))$, for $j = 1$ to l, $v^-_{i_j} \in \mathsf{IN}(C_{i_j})$, $v^+_{i_j} \in \mathsf{OUT}(C_{i_j})$ and $v^-_1 \in \mathsf{IN}(C)$. Since $\phi(r_1)$ is R-Valid, $r_1 \in \mathsf{R}^-(v^+_{r_1})$, an elementary path $r_1 \rightsquigarrow v^-_{i_1}$ exists, and therefore $r_1 \in \mathsf{R}^-(v^-_{i_1})$. Also, since H is NR-Valid, it is easy to see that $r_1 \in \mathsf{R}^-(v^+_{i_1})$ as well, and therefore, the elementary path $r_1 \rightsquigarrow v^+_{i_1}$ exists. This elementary path can be extended up to v_1 using the same argument and thus establishing that $r_1 \in \mathsf{R}^-(v_1)$. Using a similar argument, we can show that $r_2 \in \mathsf{R}^+(v_2)$. This contradicts the assumption that $e = (v_1, v_2)$ is not on any elementary path from r_1 to r_2.

We next argue the completeness of H. Assume by way of contradiction that there exists an edge $e \in E(G)$ on an elementary path p from r_1 to r_2, $r_1, r_2 \in R$, such that the edge $e' \in E(H)$ induced by e, is not on any elementary path from $\phi(r_1)$ to $\phi(r_2)$ in H. Let the $r_1 \rightsquigarrow r_2$ elementary path p be $r_1, v_1, \cdots, v_l, r_2$. The subgraph of H induced by the nodes in the set $\{\phi(r_1), \phi(v_1), \phi(v_2), \cdots, \phi(v_l), \phi(r_2)\}$ must induce a path from $\phi(r_1)$ to $\phi(r_2)$ and let the path be $\phi(r_1)$, $C_1, \cdots, C_q$, $\phi(r_2)$. We will now prove that the path $\phi(r_1)$, C_1, $\cdots$, C_q, $\phi(r_2)$ is elementary. For each node v_i on the path p, $r_1 \in \mathsf{R}^-(v_i)$ and $r_2 \in \mathsf{R}^+(v_i)$. Therefore, $\phi(v_i) \neq \phi(r)$ for any R-node $r \notin \{r_1, r_2\}$, as otherwise, we violate the assumption that H is R-Valid. We also claim that for each C_j, $C_j \notin \{\phi(r_1), \phi(r_2)\}$. If $C_j = \phi(r_1)$ for some j, then there exists a $v \in IN(C_j)$, with $r_2 \in \mathsf{R}^+(v)$, contradicting that $C_j = \phi(r_1)$ is R-Valid. Similarly if $C_j = \phi(r_2)$ for some j, then there exists a $v \in OUT(C_j)$, with $r_1 \in \mathsf{R}^-(v)$, again contradicting that $C_j = \phi(r_2)$ is R-Valid. $\square$

Theorem 2. A well-formed user view is good if and only if it is both R-Valid and NR-Valid.

The significance of Theorem 2 is that it maps global constraints on the structure of a good user view to a simple collection of locally-testable properties.

4. SERIES-PARALLEL WORKFLOWS

Given a specification such that the underlying graph G is a (two terminal) directed SP graph, we now present an algorithm for finding a good user view with minimum size for G. Since the completeness and soundness properties are not altered by the presence or absence of multiple copies of an edge, we will assume w.l.o.g. that the input graph G is a simple directed SP graph (i.e. it has no parallel edges).

4.1 The Algorithm SP-View

We start with an overview of the algorithm. Our algorithm processes the vertices in a *topologically sorted order* [8], making a forward and a backward pass, and incrementally builds the clusters in the final user view.

Definition 10. Given a directed acyclic graph G, a *topologically sorted order* (or, simply a *topological order)* is a linear ordering τ on $V(G)$ such that for each edge $(u, v) \in E(G)$, u is listed before v in τ.

The fact that such a topological ordering exists follows from Observation 1.

In the forward pass, if possible, Procedure **SPV-Forward** merges each newly encountered NR-node with one of the (already formed) clusters which its predecessors in G belong to. We will prove that all intermediate user views in this process are directed SP graphs and are good. **SPV-Forward** takes G as input and outputs an intermediate SP user view H'.

Algorithm 1 Algorithm **SP-View**
Input: A directed SP graph G
Output: A good user view H''

```
– Run SPV-Forward on $G$ to produce $(H', \phi')$.
– Run SPV-Reverse on $(H', \phi')$ to produce $(H'', \phi'')$.
– output $(H'', \phi'')$.
```

In the backward pass, Procedure **SPV-Reverse** performs a mirror step. **SPV-Reverse** takes H' as input and outputs the final user view H''. This algorithm is similar to **SPV-Forward** but processes nodes in H' (which are clusters of nodes in G) in reverse topological order , i.e. merges nodes with their successors instead of their predecessors. We also show that, there is a simple linear ($O(n)$) time implementation of **SP-View** and given an SP workflow graph G as input, **SP-View** outputs a user view which is optimum in size for G.

Then we show that, for each *NR*-cluster that remains in the final user view H'', a unique *R*-cluster can be identified as a witness for its existence. In addition to that, each *R*-cluster serves as a witness of at most one *NR*-cluster in the final user view. This observation with slight care in handling of the boundary cases suffices to argue that the total number of clusters in the output can not exceed $2k - 3$.

A key step in these algorithms is *merging* two clusters C and C' in a user view (H^1, ϕ^1) to create a new cluster C^*. This creates a new user view (H^2, ϕ^2) defined as follows.

1. (Merge C and C') $\forall u \in V(G)$ if $\phi^1(u) \in \{C, C'\}$, define $\phi^2(u) = C^*$ and add C^* to $V(H^2)$.
2. (Keep other clusters unchanged) $\forall u \in V(G)$ such that $\phi^1(u) \notin \{C, C'\}$, define $\phi^2(u) = \phi^1(u)$. Also add each $C'' \notin \{C, C'\}$ to $V(H^2)$.
3. (Edge set) For each edge $(u, u') \in E(G)$, if $\phi^2(u) \neq \phi^2(u')$, add the edge $(\phi^2(u), \phi^2(u'))$ to $E(H^2)$. Note that as G is a directed SP graph and therefore is acyclic from Observation 1, any user view of G will not have a self-loop.

4.2 Algorithm SPV-Forward

Let $v_1, v_2, ..., v_n$ denote nodes of G in a topological order. Algorithm **SPV-Forward** proceeds iteratively, processing the node v_i of G in iteration i. At the end of an iteration i, it creates an intermediate (good) user view (H_i, ϕ_i). It starts with (H_0, ϕ_0) where ϕ_0 is a one-to-one mapping $\forall v,\ \phi_0(v) = \{v\}$. In iteration i, v_i is considered for merging with one of its predecessors. Upon termination, **SPV-Forward** returns $(H', \phi') = (H_n, \phi_n)$.

It is easy to see that in any topological order of the nodes in $V(G)$, $v_1 = s$ and $v_n = t$. We will show that each intermediate user view H_i is a good user view. Hence from Corollary 2, in each intermediate graph H_i, only $\phi_i(s)$ has 0-indegree, and only $\phi_i(t)$ has 0-outdegree. Therefore, if an *NR*-node v_i is processed in iteration i, then v_i must satisfy one of the three cases (II), (III) and (IV).

Table 1: Actions taken by SPV-Forward to process an *NR*-node v_i based on the number of *R*-predecessors and *NR*-predecessors of v_i in H_{i-1}

Case	#*R*-predecessor	#*NR*-predecessor	Action
(I)	$= 0$	$= 0$	Does not arise
(II)	$= 0$	≥ 1	Merge with the last predecessor in a topological order
(III)	$= 1$	$= 0$	Merge with the predecessor
(IV)	≥ 1 ≥ 2	≥ 1 ≥ 0	Do nothing

Algorithm 2 Algorithm **SPV-Forward**
Input: A directed SP graph G
Output: A good SP user view (H', ϕ')

```
– Let $n = |V(G)|$
– Let $(v_1, v_2, \cdots, v_n)$ be a topological order of vertices in $V(G)$
for $i = 1$ to $n$ do {/* $H_0 = G$ */}
  – $\phi_0(v_i) = \{v_i\}$
end for
for $i = 1$ to $n$ do {/* $\phi_{i-1}(v_i) = \{v_i\}$ */}
  if $v_i$ is an R-node then {/* do nothing */}
    –$(H_i, \phi_i) = (H_{i-1}, \phi_{i-1})$
  else {/* $v_i$ is an NR-node */}
    if $\phi_{i-1}(v_i)$ has no R-predecessor in $H_{i-1}$ then {/* Case (II) */}
      – Let the NR-predecessors of $\phi_{i-1}(v_i)$ in $H_{i-1}$ be the clusters $C_1, C_2, \cdots, C_\ell$ listed in a topological order in $H_{i-1}$
      – Merge clusters $C_\ell$ and $\phi_{i-1}(v_i)$
    else if $\phi_{i-1}(v_i)$ has exactly one R-predecessor and no NR-predecessor in $H_{i-1}$ then {/* Case (III) */}
      – Let $C$ be the unique predecessor of $\phi_{i-1}(v_i)$ in $H_{i-1}$ (and $C$ is an R-cluster)
      – Merge clusters $C$ and $\phi_{i-1}(v_i)$
    else {/* Case (IV): $\phi_{i-1}(v_i)$ has $\geq 2$ predecessors including $\geq 1$ R-predecessors in $H_{i-1}$; do nothing */}
      – $(H_i, \phi_i) = (H_{i-1}, \phi_{i-1})$
    end if
  end if
end for
– $(H', \phi') = (H_n, \phi_n)$
– output $(H', \phi')$
```

4.2.1 *Invariants of* SPV-Forward

As mentioned earlier, **SPV-Forward** incrementally forms an intermediate user view processing the node v_i in iteration i so that each intermediate user view formed satisfies some properties. The following lemma lists a set of structural properties of the user view formed at the end of each iteration. We defer the proof to the full version of the paper.

Lemma 7. For any $i \in [1..n]$, upon termination of the i-th iteration of **SPV-Forward**, the pair (H_i, ϕ_i) satisfies the following invariants:

1. H_i is acyclic.
2. H_i is a directed SP-graph.
3. If $\phi_i(v_i)$ is an NR-cluster, then $\phi_i(v_i)$ has at least two predecessors in H_i, at least one of which is an R-predecessor.

Hence, after the execution of **SPV-Forward** on G, we get a user view H' satisfying the properties given by the following corollary.

Corollary 3. The output graph H' produced by **SPV-Forward** is a directed SP graph. Moreover, if C is an NR-cluster in H', C has at least two predecessors including an R-predecessor.

4.2.2 *Correctness of* SPV-Forward

We will use the characterization of a good user view given in Theorem 2 to prove that the graph H' output by **SPV-Forward** is a good user view.

Lemma 8. After each iteration i of **SPV-Forward**, the intermediate graph H_i is a good user view.

Proof. We prove the lemma by induction on the number of iterations. At $i = 1$, the start node s gets processed, and since it is an R-node, $H_1 = H_0 = G$, and hence trivially is a good user view.

Now suppose that the hypothesis holds until iteration $i-1$, just before the node v_i is processed. If v_i is an R-node or v_i is an NR-node that satisfies Case (IV), $H_i = H_{i-1}$ and we are done. Hence we need to only consider the Cases (II) and (III) for an NR-node v_i. Say in these cases $C = \phi_{i-1}(v_i)$ is merged with a predecessor C' in H_{i-1} to form the cluster C^* in H_i. Note that H_i is well-formed.

Case (II): H_i is R-Valid as no R-node is being changed. First consider the case when C' is the unique NR-predecessor of $C = \{v_i\}$ in H_{i-1}. Since C' is NR-Valid, for all $u \in \mathsf{OUT}(C')$ the $\mathsf{R}^-(u)$ sets are same and are equal to $\mathsf{R}^-(C')$. C' is the unique predecessor of C, so for all edges $(u, v_i) \in E(G)$, $u \in C'$ in H_{i-1} and therefore, $u \in \mathsf{OUT}(C')$. Hence $\mathsf{R}^-(v_i) = \bigcup_{(u,v_i)\in E(G)} \mathsf{R}^-(u) = \mathsf{R}^-(C')$. Therefore $\mathsf{R}^-(C^*) = \mathsf{R}^-(v_i) \cup \mathsf{R}^-(C') = \mathsf{R}^-(C')$. In H_i, $v_i \notin \mathsf{IN}(C^*)$, $v_i \in \mathsf{OUT}(C^*)$ and $\mathsf{R}^-(v_i) = \mathsf{R}^-(C^*)$. Hence for all $u \in \mathsf{OUT}(C^*)$, $R^-(u) = \mathsf{R}^-(C) = \mathsf{R}^-(C^*)$. Also note that $R^+(C^*) = \mathsf{R}^+(C')$ (as $\exists u \in C'$ such that $(u, v_i) \in E$, $R^+(C) \subseteq \mathsf{R}^+(C')$, so $R^+(C^*) = \mathsf{R}^+(C') \cup \mathsf{R}^+(C) = \mathsf{R}^+(C')$). Since C' is NR-Valid and $v_i \notin \mathsf{IN}(C^*)$, for all $v \in \mathsf{IN}(C^*)$, $\mathsf{R}^+(v) = \mathsf{R}^+(C') = \mathsf{R}^+(C^*)$. So C^* remains NR-Valid in H_i, and as all other NR-nodes are unchanged, H_i is NR-Valid. Since H_i is well-formed, it is a good user view.

Now we consider the case when $C = \{v_i\}$ has $\ell > 1$ NR-predecessors $C_1, \cdots, C_\ell$, listed in the topological order of clusters in H_{i-1} and C is merged with C_ℓ to form C^* in H_i. From Invariant 2 of Lemma 7 H_{i-1} is a directed SP graph, hence from Lemma 3 C is the unique successor of C_ℓ. Therefore, for each $u \in \mathsf{OUT}(C_\ell)$, $(u, v) \in E(G)$ with $v \notin C_\ell$ if and only if $v = v_i$. Hence in H_i, $\mathsf{OUT}(C^*) = \{v_i\}$ and $\mathsf{IN}(C^*) = \{v_i\} \cup \mathsf{IN}(C_\ell)$. Since $\mathsf{OUT}(C^*) = \{v_i\}$, $\forall v \in C^*$, $\mathsf{R}^+(v) = \mathsf{R}^+(v_i) = \mathsf{R}^+(C^*)$. Therefore $\forall v \in \mathsf{IN}(C^*)$, $\mathsf{R}^+(v) = \mathsf{R}^+(C^*)$. Again as $\mathsf{OUT}(C^*) = \{v_i\}$, $\mathsf{R}^-(v_i) = \bigcup_{v\in C^*} \mathsf{R}^-(v) = \mathsf{R}^-(C^*)$. These two facts together imply that C^* is NR-Valid. All other NR-nodes are unchanged in H_i. Hence H_i is NR-Valid.

Case (III): Here C' is the unique R-predecessor of C. Since H_{i-1} is a good user view, each R-cluster in H_{i-1} is R-Valid and each NR-cluster is NR-Valid. Now all NR-clusters except the singleton cluster $C = \{v_i\}$ are unchanged in H_i, and C is merged with an R-cluster in H_i. Therefore, R^- and R^+ sets of all NR-clusters remain unchanged in H_i (recall that R^- and R^+ sets of a cluster are defined over the specification graph G) and H_i is NR-Valid. Now we show that C^* is R-Valid. Since H_{i-1} is a good user view, R-cluster C' in H_{i-1} is R-Valid. Let $r \in R$ be the unique R-node in C' (as H_{i-1} is good, it is well-formed). Since C' is the unique predecessor of $C = \{v_i\}$ in H_{i-1}, for all edges $(u, v_i) \in E(G)$, $u \in C'$ in H_{i-1}, and therefore, $u \in \mathsf{OUT}(C')$. Since C' is R-Valid, $R^-(u) = \{r\}$, and therefore $R^-(v_i) = \{r\}$. In H_i, $v_i \in \mathsf{OUT}(C^*)$ (because v_i is an NR-node, $v_i \neq$ sink node $t \in V(G)$, and v_i can not have 0 outdegree), and $R^-(u) = \{r\}$. For all other nodes $u \in \mathsf{OUT}(C^*)$ in H_i, $R^-(u) = \{r\}$ and for all nodes $v \in \mathsf{IN}(C')$, $v \in \mathsf{IN}(C^*)$, hence $R^+(v) = \{r\}$ (note that $v_i \notin \mathsf{IN}(C^*)$). So C^* remains R-Valid in H_i. So H_i is both R-Valid and NR-Valid, and it is well-formed. Hence from Theorem 2, H_i is a good user view. □

Hence after the execution of **SPV-Forward**, we have a good user view as stated in the following corollary.

Corollary 4. H' is a directed SP graph that is a good user view of the input specification graph G.

4.3 Algorithm SPV-Reverse

SPV-Reverse takes the intermediate user view (H', ϕ') as input, and performs the merging process of **SPV-Forward** on a reverse topological order of the clusters in H'. More precisely, we can view **SPV-Reverse** as reversing the direction of all edges in H' and running **SPV-Forward** on this reversed copy of H'. Thus **SPV-Reverse** attempts to merge a cluster in H' with one of its successor instead of its predecessors.

It can be verified that the property of an NR-cluster of graph H' given by Invariant 3 is not destroyed by **SPV-Reverse**. Assume that $C^* = \{C_{i_p}, C_{i_{p-1}}, \cdots, C_{i_1}\}$ is an NR-cluster in H'' where $C_{i_p}, C_{i_{p-1}}, \cdots, C_{i_1}$ are NR-clusters of H' and they entered C^* in the order $i_p, i_{p-1}, \cdots, i_1$. From Corollary 3, C_{i_1} had two predecessors including an R-predecessor. If C_{i_1} had two R-predecessors in H', they will form two R-clusters in H'' (as H'' is well-formed), and the property will hold in H''. Otherwise, C_{i_1} had at least one NR-predecessor, say C_{nr}, and an R-predecessor, say C_r in H'. As C_{i_1} is the last cluster of H' added in C^*, C_{nr} is not included in C^* and hence must belong to an NR-cluster C^*_{nr} in H'' (note that C_{nr} can not be merged with one of its R-successor as C_{i_1} is one of its NR-successor in H'). The R-predecessor C_r of C_{i_1} will form an R-cluster, say C_r^* in H' and will be a predecessor of C^*. Hence the cluster C^* in H'' will have at least one R-predecessor C_r^* and at least one NR-predecessor C_{nr}^*.

Thus using Corollary 3 and Lemma 8, we can conclude the following.

Lemma 9. The output graph H'' produced by **SP-View** is a directed SP graph. Moreover, if C is an NR-cluster in H'', C has at least two predecessors including an R-predecessor and at least two successors including an R-successor.

The correctness proof of **SPV-Forward** on G as done in Lemma 8, along with the R-Valid and NR-Valid properties of the clusters in H', can also be extended to **SPV-Reverse**, which runs **SPV-Forward** on the "reverse" of H'. Combining all these, the next theorem follows.

Theorem 3. H'' is a two terminal directed SP graph that is a good user view of the input specification graph G.

4.4 An Example

Figure 6 (a) shows a directed SP graph G given as a specification with $n = 14$ and $k = 6$. The numbers next to each node in the figure denote the position of the nodes in an arbitrary topological order. The nodes in G are merged with their predecessor clusters in the intermediate user views. For example, when node v_5 is processed (in the 5-th iteration), the R-cluster $\{v_2, v_4\}$ is its unique predecessor in H_4, and v_5 is merged with this cluster to form a new cluster $\{v_2, v_4, v_5\}$ (Case (III)). Again, when the node v_8 is processed (in the 8-th iteration), the clusters $\{v_2, v_4, v_5\}, \{v_1, v_6\}$ and $\{v_7\}$ are its predecessors in H_7. As it has two R-predecessors and one NR-predecessor, v_8 satisfies Case (IV) and remains a singleton NR-cluster. Later in the 9-th iteration, $\{v_9\}$ gets merged with $\{v_8\}$ to form the cluster $\{v_8, v_9\}$ and in the 12-th iteration, $\{v_{12}\}$ gets merged with $\{v_8, v_9\}$ to form the final cluster $\{v_8, v_9, v_{12}\}$. The output H' of **SPV-Forward**, with nine clusters, is given in Figure 6 (b).

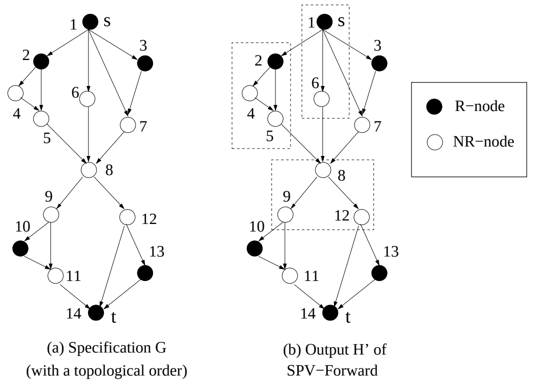


Figure 6: An example (SPV-Forward): input specification G and output (intermediate) user view H'

SPV-Reverse takes H' as input and merges clusters in H' with their successors, if possible. Let $C_1, \cdots, C_9$ be an arbitrary topological order of the clusters in H' (Figure 7 (a)). Then **SPV-Reverse** processes the clusters in the order $C_9, \cdots, C_1$. For example, since the NR-cluster C_5 is the unique successor of C_4, these two clusters are merged together when the cluster C_4 is processed. The final good and optimum user view H'' is given in Figure 7 (b). Note that the user view H'' has only seven clusters, whereas the original specification contains fourteen modules. A provenance query on H'' will only involve information flow between these clusters. Thus a significant amount of *non-relevant* information flow between modules will be suppressed, while still preserving the *identity* of relevant modules and their *interaction*.

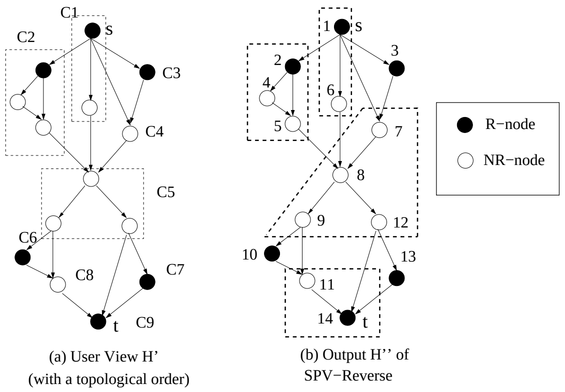


Figure 7: An example (SPV-Reverse): input (intermediate) user view H' and output (final) user view H''

4.5 Time Complexity

This section gives a simple linear time implementation of the algorithm **SP-View**, and thus we have the following theorem.

Theorem 4. The algorithm **SP-View** can be implemented in $O(n)$ time where $n = |V(G)|$.

We assume that the workflow specification is given in adjacency list representation and we have the outgoing and incoming edges for a node $v \in V(G)$ available in linked lists in(v) and out(v) respectively. We also assume that the information whether a node v is an R-node in G is given in an array so the look up can be done in $O(1)$ time. Now we describe an implementation of the algorithm **SPV-Forward**, and as **SPV-Reverse** takes the graph H' output by **SPV-Forward** as blackbox, it can be implemented in similar way.

First we do a topological ordering of the vertices in G, and that takes $O(m + n)$ time. Next we describe how to update the function $\phi_i(v)$ for a vertex v in an iteration i. We give the inductive definition of name of a cluster $\phi(v)$ as follows. Note that no vertex is ever removed from a cluster, so if a vertex is assigned to a cluster that is fixed till the end of the algorithm.

- (Base Case): Assign $\phi(s) = C_1$.

- (Iteration i): Let $C_1, \cdots, C_{j-1}$ be the clusters allocated so far till iteration $i-1$. If the node processed in iteration i is not merged with any predecessor, allocate a new cluster C_j and assign $\phi(v_i) = C_j$.

  Otherwise, if v_i is merged with a predecessor cluster C_{j_1}, assign $\phi(v_i) = C_{j_1}$.

We claim that as the v_is are listed in topological order, the clusters C_js as they are being allocated, are always listed in topological order. If a node v_i is assigned a new singleton cluster C_j (i.e. not merged with any predecessor), then topological order of the clusters is not violated as v_i can not have an edge to a previously allocated cluster C_{j_1}, $j_1 < j$. Same is the case when v_i is merged with its unique predecessor cluster in H_{i-1}. If v_i has ≥ 2 NR-predecessors and no R-predecessor, we will merge v_i with the predecessor C_{j_1} with the highest index j_1, and again the topological order of the clusters remains maintained.

We also keep track of whether a cluster C_j is an R-cluster or NR-cluster by updating this information each time a node enters the cluster. For a vertex v, the function $\phi(v)$ can be computed as follows. If $v = v_i$ is processed in iteration i, we go over the list $\mathsf{in}(v)$. If we find all $u \in \mathsf{in}(v)$ have same $\phi(u)$, we know that v has unique predecessor in G_{i-1} (that may be an R-cluster or an NR-cluster). If not, but for all the $u \in \mathsf{in}(v)$, $\phi(u)$ are NR-clusters, we compute the highest index ℓ such that for an $u \in \mathsf{in}(v)$, $\phi(u) = C_\ell$ and assign $\phi(v_i) = C_\ell$. This way we find the last predecessor in a topological order in H_{i-1}. Otherwise we know that v_i has at least two predecessors in G_{i-1} including one R-cluster and we allocate new C_j for $\phi(v_i)$. Note that for a vertex v, the function $\phi(v)$ can be computed in time $= O(|\mathsf{in}(v)|)$. Hence total time complexity for all the iterations is $\max(n, \sum_{v \in V(G)} (O(|\mathsf{in}(v)|) = O(m+n)$. Finally we output the graph H' by going over all $v \in V(G)$ and looking at $\phi(v)$ to build the sets $\mathsf{in}(C)$ and $\mathsf{out}(C)$ for each cluster C in $V(H')$, which will be the input for **SPV-Reverse**. Again it can be done in $O(m+n)$ time. Hence the algorithm **SP-View** can be implemented in $O(m+n)$ time which is linear in terms of the size of the input graph G. Since for an SP graph, $m = O(n)$, the overall time complexity is $O(n)$, that is, linear in the size of $V(G)$.

4.6 Optimality of the Algorithm SP-View

In this section we show that the number of clusters in any good user view is as large as the number of clusters in H'', i.e. we prove the following theorem.

Theorem 5. For a given specification (G, s, t, R) where G is a two terminal directed SP graph, the user view (H'', ϕ'') output by Algorithm **SP-View** is optimum in size.

Here we will give an overview of the proof and state the main lemmas, and defer the details to the full version of the paper.

Overview of the Proof:. Suppose the output user view H'' contains $N_r = k$ R-clusters and N_{nr} NR-clusters. As any good user view H is well-formed, it has to contain N_r R-clusters. Hence it suffices to prove that any good user view also has at least N_{nr} NR-clusters.

Let us recall from Corollary 1 that if an NR-node v is such that $|R^-(v)| \geq 2$ and $|R^+(v)| \geq 2$ then v cannot be included in an R-cluster in any good user view. We will call these NR-nodes as $\mathsf{Essential}$ NR-nodes, that must be in NR-clusters in any good user view.

Definition 11. An NR-node v is called $\mathsf{Essential}$ if $|R^-(v)| \geq 2$ and $|R^+(v)| \geq 2$.

First we show that each NR-cluster in H'' contains at least one $\mathsf{Essential}$ NR-node. Then we deduce a sufficient condition when two $\mathsf{Essential}$ NR-nodes v_i, v_j, are included in the same NR-cluster in H'' by **SP-View**. Finally we show that when two $\mathsf{Essential}$ NR-nodes v_i, v_j are put in different NR-clusters by **SP-View**, then no good user view can put v_i, v_j in the same NR-cluster. In particular, if C_1 and C_2 are two distinct NR-clusters of H'', then an $\mathsf{Essential}$ NR-node from C_1 and an $\mathsf{Essential}$ NR-node from C_2 can never be included in the same NR-cluster in any good user view. This will complete the proof of Theorem 5.

(1) Existence of $\mathsf{Essential}$ nodes in each NR-cluster in H'':. For an NR-node v, we will prove that if $\phi'(v)$ is an NR-cluster in H', then $|R^-(v)| \geq 2$. Similarly, if C is an NR-cluster in H'', $|\mathsf{R}^+(C)| \geq 2$. As H'' is good, any NR-cluster C is NR-Valid, hence for all $v \in \mathsf{IN}(C)$, $\mathsf{R}^+(v) = \mathsf{R}^+(C)$. Hence for any $v \in \mathsf{IN}(C)$, $|\mathsf{R}^+(v)| \geq 2$. If $\phi''(v) = C$ is an NR-cluster, $\phi'(v)$ is also an NR-cluster, therefore $|\mathsf{R}^-(v)| \geq 2$. Hence, for each NR-cluster C in H'', there is an NR-node v such that $|\mathsf{R}^-(v)| \geq 2$ and $\mathsf{R}^+(v) \geq 2$.

Lemma 10. Each NR-cluster C in H'' contains at least one $\mathsf{Essential}$ NR-node.

(2) A sufficient condition for $\phi''(v_i) = \phi''(v_j)$, where v_i, v_j are two $\mathsf{Essential}$ NR-nodes. The following lemma gives a sufficient condition.

Lemma 11. For two $\mathsf{Essential}$ NR-nodes v_i, v_j, if (i) $R^-(v_i) = R^-(v_j)$ or $R^+(v_i) = R^+(v_j)$, or, (ii) there exists a $v_i \rightsquigarrow v_j$ path and all paths from v_i to v_j are elementary paths, then $\phi''(v_i) = \phi''(v_j)$.

Lemma 11 will be proved in two steps. First we will prove that for two $\mathsf{Essential}$ nodes v_i, v_j, if $R^-(v_i) = R^-(v_j)$ or $R^+(v_i) = R^+(v_j)$, then $\phi''(v_i) = \phi''(v_j)$. Next we will show that if a $v_i \rightsquigarrow v_j$ path exists and all paths from v_i to v_j are elementary paths, then there exists an NR-node v_ℓ such that both $R^-(v_\ell) = R^-(v_j)$ and $R^+(v_i) = R^+(v_\ell)$ hold. Combining these two facts Lemma 11 will be proved.

(3) The number of NR-clusters in H'' is optimum:. To complete the proof of optimality of the algorithm, for two

Essential NR-nodes v_i, v_j, we will show that if (i) there is any R-node on any path from v_i to v_j, or (ii) $R^-(v_i) \neq R^-(v_j)$ and $R^+(v_i) \neq R^+(v_j)$ and no v_i to v_j path exists, then v_i and v_j cannot be included in the same NR-cluster in any good user view. Combining the above observation with Lemma 11 the next lemma proving the optimality of **SP-View** follows.

Lemma 12. For two Essential NR-nodes v_i, v_j, if $\phi''(v_i) \neq \phi''(v_j)$ then no good user view can put v_i, v_j in the same NR-cluster.

Thus the number of NR-clusters in H'' cannot be reduced in any good user view and the user view H'' output by **SP-View** is optimum in size.

4.7 Extremal Bounds on Size of a Good User View for SP Graphs

We now give tight extremal bounds on the size of good user views when the specification graph is an SP graph. We show that, whatever be the total number of nodes n, the size of H'' output by **SP-View** is at most $2k-3$, and there exist SP graphs which always require $2k-3$ clusters in a good user view.

4.7.1 An Upper Bound on the Size of a good User View for SP Graphs

First we show that if there is an NR-cluster C in the final user view H'' by the algorithm **SP-View**, there is an R-predecessor of C which "accounts for" the existence of C in H''. The following lemma will be used in proving the upper bound; the proof is deferred to the full version.

Lemma 13. Let $C \in V(H'')$ be an NR-cluster. Then in any topological order of the clusters in $V(H'')$, the last predecessor C' of C is an R-cluster and C is the unique successor of C' in H''.

The following corollary immediately follows from Lemma 13.

Corollary 5. In H'', each NR-cluster $C \in V(H'')$ has at least one R-predecessor C' such that C is the unique successor of C' in H''.

Note that since $s, t \in R$, $k \geq 2$, and therefore any good user view must contain at least 2 clusters. Next we show that $|V(H'')| \leq 2k-3$, thereby proving the following theorem.

Theorem 6. Given a workflow specification (G, s, t, R) where G is a directed SP graph, there exists a good user view of size at most $2k-3$ that is also a directed SP graph, whenever $k = |R| \geq 3$.

Proof. We will show this by a simple "charging argument" on the good user view H'' by our algorithm. If the number of NR-clusters in H'' is 0, then the upper bound of $2k-3$ trivially holds because $2k-3 \geq k$ for $k \geq 3$. Otherwise, let $C_1, C_2, \cdots, C_\ell$ be the NR-clusters in H'', listed in the topological order for some $\ell \geq 1$. We start with an initial assignment of a charge of 1 to each R-cluster. The analysis above shows that this charge assignment, a total of k units, suffices to account for all NR-clusters. We will show that this charge can be reduced to $k-3$ and still all NR-clusters will get accounted for, proving the desired bound. This is achieved by noting that Lemma 9 requires the cluster C_1 to have at least two predecessors. But each predecessor of C_1 must be necessarily an R-predecessor as C_1 is the first NR-cluster in the topological sorted order. We argue that these R-predecessors of C_1 are not charged by any other NR-cluster. This is because if C_1 is the single NR-successor of these clusters, we are done. If an R-predecessor C_r of C has another NR-successor $C'_1 \neq C_1$, by Corollary 5, C'_1 has one R-predecessor $C'_r \neq C_r$, such that C'_1 is unique NR-successor of C'_r, and therefore C'_r is charged for the existence of C'_1. Thus we can drop the charge on one of these two clusters and still account for all NR-clusters. Similarly, C_ℓ must have at least two R-successors such that neither of them is an R-predecessor of any NR-cluster. We can drop the charge on both of these clusters. Thus the total remaining charge on R-clusters is exactly $k-3$, proving that the total number of clusters in H'' is at most $2k-3$. □

4.7.2 A Lower Bound on the Size of a good User View for SP Graphs

In this section we show that the upper bound given in Theorem 6 for directed SP graphs is tight.

Theorem 7. There is a directed SP graph H_k with k R-nodes that needs exactly $2k-3$ nodes in any good user view.

The example graph H_k with $2k-3$ nodes is given in Figure 8 where the r_i's are R-nodes and nr_j's are NR-nodes. It is easy

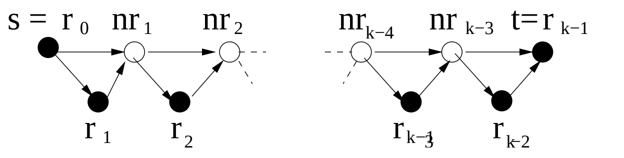


Figure 8: SP graph H_k that needs $2k-3$ clusters

to see that no two nodes in this graph can be included in the same cluster in any good user view, hence we omit the proof.

5. GENERAL GRAPHS

In this section we give an upper bound on the number of clusters for the general graph model using the ZOOM algorithm given in [2]. The upper bound we get is an (exponential) function of number of relevant modules k and is independent of n (the total number of modules in the specification). We also show that our upper bound is tight, i.e., there exist graphs that need exponentially many nodes in any good user view. However, given a general graph as a specification, the question of whether there exists a polynomial time algorithm for constructing an optimum user view is an open problem.

Given any graph G, the algorithm in [2] can be described as follows in terms of our notations.

1. If for an *NR*-node v, $\mathsf{R}^-(v) = \{r\}$ or $\mathsf{R}^+(v) = \{r\}$, v and r are put in the same cluster.
2. For an *NR*-node v and an *NR*-cluster C (after Step 1), if $\mathsf{R}^-(C) = \mathsf{R}^-(v)$ and $\mathsf{R}^+(C) = \mathsf{R}^+(v)$, merge C and v in the same cluster. Repeat Step 2 until no more merging is possible.
3. (To make the user view *minimal*) For two *NR*-clusters C_1 and C_2 after Step 2, let $C = C_1 \cup C_2$. If $\forall v \in \mathsf{OUT}(C), \mathsf{R}^-(v) = \mathsf{R}^-(C)$ and $\forall v \in \mathsf{IN}(C), \mathsf{R}^+(v) = \mathsf{R}^+(C)$, merge C_1 and C_2 in the user view. Repeat Step 3 until no more merging is possible.

For simplicity, we also assume that *NR*-nodes can be merged with s and t.

5.1 Upper bound on Size of Good User View

The following theorem shows that for general directed graphs there exists a good user view whose size is a function of k (and not of n); moreover, this good user view can be computed in polynomial time using the ZOOM algorithm.

Theorem 8. For a workflow specification (G, s, t, R), with $k = |R|$, the ZOOM algorithm [2] outputs a good user view with size $\le (2^{k-1} - k)^2 + k$.

Proof. We look at the good user view output by the ZOOM algorithm on G. Since in the specification only s can have 0-indegree and only t can have 0-outdegree, for all *NR*-nodes in G, the R^- and R^+ sets are non empty. Since the ZOOM algorithm merges an *NR*-node with its unique *R*-predecessor or its unique *R*-successor in Step 1, each *NR*-node v which is not in a cluster with some *R*-node after Step 1 has $|\mathsf{R}^-(v)| \ge 2$ and $|\mathsf{R}^+(v)| \ge 2$. Also note that for any *NR*-node $v \in V(G)$, $s \notin \mathsf{R}^+(v)$ and $t \notin \mathsf{R}^-(v)$, and therefore, $|\mathsf{R}^-(v)|,\ |\mathsf{R}^+(v)| \le k-1$. Step 2 puts *NR*-nodes with the same R^- and R^+ clusters in same cluster. The number of distinct pairs $(\mathsf{R}^-,\ \mathsf{R}^+)$ such that $2 \le |\mathsf{R}^-| \le k-1$ and $2 \le |\mathsf{R}^+| \le k-1$ is $(2^{k-1} - 1 - (k-1))^2$. Step 3 does further minimalization if possible. Taking into account the *R*-clusters in the user view, the total number of clusters output by the algorithm is therefore bounded by $(2^{k-1} - 1 - (k-1))^2 + k$. $\square$

5.2 Lower bound on Size of Good User view

We prove that the upper bound given by Theorem 8 is tight by constructing an example specification graph with k *R*-nodes such that any good user view for G will have at least $(2^{k-1} - k)^2 + k$ clusters.

Theorem 9. There exists a workflow specification (G, s, t, R), with $k = |R|$, such that number of clusters in any good user view is exactly $(2^{k-1} - k)^2 + k$.

Proof. We construct a specification as follows. Label the *R*-nodes in G as $r_1, \cdots, r_k$, where $r_1 = s$ and $r_k = t$. Let $v_1, v_2, ..., v_N$ be the *NR*-nodes in G (the value of N will be computed later). Let P (resp. S) be the set of all possible subsets of size $\in [2, k-1]$ of $\{r_1, \cdots, r_{k-1}\}$ (resp. $\{r_2, \cdots, r_k\}$). Now form all possible pairs $(X,\ Y)$, $X \in P,\ Y \in S$, and make a one-to-one assignment between the pairs (X, Y) and *NR*-nodes v_i, and create the edges such that $\mathsf{R}^-(v_i) = X$ and $\mathsf{R}^+(v_i) = Y$. Note that X and Y can be same set for some v_i. As $|P| = |S| = (2^{k-1} - 1 - (k-1)) = (2^{k-1} - k)$ and the number of distinct pairs (X, Y) is $(2^{k-1} - k)^2 = N$. To make this construction a feasible workflow specification (i.e. to keep all the nodes in $V(G)$ on some path from s to t), we also add the edges $\{(r_1,\ r_i) \mid 2 \le i \le k-2\}$ and $\{(r_i,\ r_{k-1}) \mid 2 \le i \le k-2\}$ to E.

For each v_i, $|\mathsf{R}^-(v_i)| \ge 2$ and $|\mathsf{R}^+(v_i)| \ge 2$ - by Corollary 1 they can not be put together in an *R*-cluster. If $v_{i_1}, v_{i_2}, \cdots, v_{i_\ell}$ are put in the same cluster C, then each $v_{i_j} \in \mathsf{IN}(C)$ and each $v_{i_j} \in \mathsf{OUT}(C)$. Since no two of the v_{i_j}s have both of R^- and R^+ sets same, C is not *NR*-Valid and the user view would not be good (from Theorem 2). Hence the number of clusters in any good user view is $|V(G)| = N + k = (2^{k-1} - k)^2 + k$. $\square$

6. COMPARISON OF SP-VIEW AND ZOOM ON SERIES-PARALLEL GRAPHS

First, note that [2] assumes nodes are not allowed to be merged with s or t. To model this, **SP-View** needs another step in which it finds all *NR*-nodes v such that $\mathsf{R}^-(v) = \{s\}$ and $\mathsf{R}^+(v) = \{t\}$ and puts them together in the same cluster. With this constraint, there is an example where **SP-View** outputs an optimum user view while the ZOOM algorithm in [2] outputs a user view which is twice the size (see Figure 9). It is not known whether or not ZOOM outputs an optimum user view for SP workflows without this constraint.

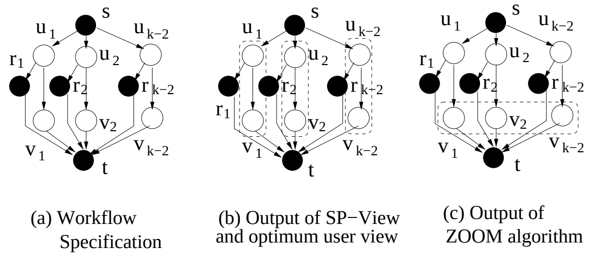
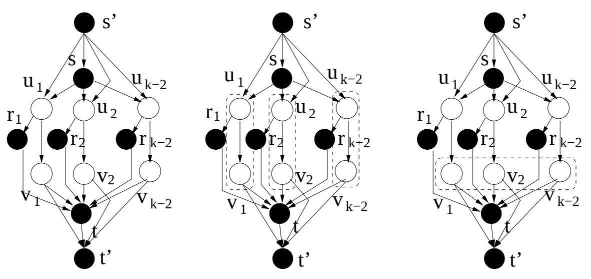

s
u 1
u 2
u k−2
r1
r2
r k−2
v1
v2
v k−2
t
(a) Workflow Specification
(b) Output of SP−View and optimum user view
(c) Output of ZOOM algorithm

Figure 9: An example where SP-View outperforms ZOOM algorithm (when no node can be merged with s and t)

For two terminal directed SP graphs, an advantage of **SP-View** over the ZOOM algorithm is its running time. Even ignoring the time to compare the R^- and R^+ sets of two nodes, ZOOM takes $O(n^2)$ time, where $n = |V(G)|$, whereas **SP-View** takes $O(n)$ time. Moreover, no preprocessing is needed for **SP-View** whereas ZOOM needs to compute the R^- and R^+ sets for all nodes.

ZOOM may fail to output the optimum user view for general graphs, even without the requirement that no node can be merged with s or t. We can extend the example given in Figure 9 to form a DAG where ZOOM does not output an optimum user view (see Figure 10).

7. CONCLUSIONS



(a) Workflow Specification (b) Optimum user view (c) Output of ZOOM algorithm

Figure 10: An example for which the ZOOM algorithm does not give optimum solution (even when nodes are allowed to be merged with s' and t')

An interesting open question is whether there is a polynomial-time algorithm to find an optimum good user view for any general directed graph. Another natural open question is to find out if the size of a view can still be exponential in number of relevant modules for arbitrary DAGs. It would also be interesting to see if **SP-View** can be extended to obtain an optimum (or near optimum) algorithm for a larger family of directed graphs, like SP graphs with a *laminar family* of subgraphs specifying forks and cycles.

8. REFERENCES

[1] O. Biton, S. Cohen-Boulakia, and S. Davidson. Zoom*UserViews: Querying relevant provenance in workflow systems (demo). In *VLDB*, 2007.

[2] O. Biton, S. Cohen-Boulakia, S. Davidson, and C. Hara. Querying and managing provenance through user views in scientific workflows. In *ICDE*, 2008.

[3] R. Bose, I. Foster, and L. Moreau. Report on the International Provenance and Annotation Workshop. *SIGMOD Rec.*, 35(3), 2006.

[4] R. Bose and J. Frew. Lineage retrieval for scientific data processing: a survey. *ACM Comp. Surveys*, 37(1):1–28, 2005.

[5] S. Bowers and B. Ludäscher. Actor-oriented design of scientific workflows. In *ER*, 2005.

[6] S. Bowers, T. M. McPhillips, and B. Ludäscher. Provenance in collection-oriented scientific workflows. In *Concurrency and Computation: Practice and Experience*. Wiley, 2007 (in press).

[7] BPEL. business process execution language for web services. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[9] S. B. Davidson, S. Cohen-Boulakia, A. Eyal, B. Ludĺascher, T. McPhillips, S. Bowers, M. K. Anand, and J. Freire. Provenance in scientific workflow systems. *IEEE Data Engineering Bulletin*, 30(4):44–50, December 2007.

[10] S. B. Davidson and J. Freire. Provenance and scientific workflows: challenges and opportunities. In *SIGMOD Conference*, pages 1345–1350, 2008.

[11] I. Foster, J. Vockler, M. Woilde, and Y. Zhao. Chimera: A virtual data system for representing, querying, and automating data derivation. In *SSDBM*, pages 37–46, 2002.

[12] J. Freire, C. T. Silva, S. P. Callahan, E. Santos, C. E. Scheidegger, and H. T. Vo. Managing rapidly-evolving scientific workflows. In *IPAW*, 2006.

[13] L. Moreau and B. Ludäscher, editors. *Concurrency and Computation: Practice and Experience – Special Issue on the First Provenance Challenge*. Wiley, 2007 (in press). (see also http://twiki.ipaw.info/bin/view/Challenge/).

[14] T. Oinn *et al.* Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(1), 2003.

[15] Y. Simmhan, B. Plale, and D. Gannon. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, 2005.

[16] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. In *STOC '79: Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 1–12, 1979.