A Declarative Approach To Customize Workflow Provenance

Saumen Dey
Advisor: Bertram Ludascher
University of California at Davis
1 Shields Avenue
Davis, California
scdey@ucdavis.edu

ABSTRACT
Provenance describes the origin, context, derivation, and ownership of data products and is becoming increasingly important in scientific applications. This information can be used, e.g., to explain, debug, and reproduce the results of computational experiments, or to determine the validity and quality of data products. In contrast, it may be infeasible or undesirable to share complete provenance of a scientific experiment. Towards finding a balance between these requirements, we develop a framework and a system that allows scientists to declaratively specify their provenance data publication and customization requirements. Using this system, scientists can specify which parts of the provenance data are to be included in the result and which parts should be hidden, or anonymized. However, arbitrary application of these specifications may not maintain provenance data integrity. Thus, we allow scientists to specify provenance data integrity requirements, in form of provenance policies, along with their provenance data publication and customization requirements. Our system then systematically applies all the publication and customization requirements on the provenance data and ensures all the provenance policies as specified by the scientist.

1. INTRODUCTION
In the emerging paradigm of collaborative, data-intensive science, scientific experiments have become increasingly complex and traditional scripting methods are getting inadequate to meet the needs of these scientific experiments. Scientific workflow systems, such as Kepler [14], Taverna [12], Vistrails [10], etc., provide an environment in which experiment design become very easy. These systems automate computational processes to support data-driven science and as an added benefit, most of these workflow systems capture provenance [15, 6]. Provenance describes the origin, context, derivation, ownership of the output data products [13]. Over the past decade, researchers have understood how provenance helps to explain the output data products and enables reuse of the output data products [3, 20, 4, 11, 5]. Also, provenance can grow larger than the data it describes [20]. Thus, to support collaboration and the reproducibility of experimental results, one needs to deal with the challenges of the provenance data management.

In collaborative settings, scientists execute experiments and produce data products, which may be useful in other experiments. The collaborators need to understand these data products before considering to use in their workflows. From a collaborator’s point of view, provenance should be adequate in explaining these data products, so retaining more provenance is preferable. At the same time, workflow systems capture provenance in a general sense and often different parts of it are relevant for different kinds of analysis. For example, the provenance recorded of a process that is used in a workflow to format the input data products, is good for debugging, but not crucial for functionally understanding the output data products. Thus, while providing adequate provenance, which can explain the data product, the provenance data needs to be relevant. While sharing provenance, scientists are required to deal with another important issue: provenance may contain sensitive, private, or intellectual property information that should not be revealed [5, 7, 4]. Thus, before publishing provenance information, scientists need to protect sensitive information and share only relevant information.

Consequently, one has to balance, as depicted in Fig. 1, between (i) the desire to provide provenance so that collaborators can understand and rely on the shared data products, and (ii) the need to protect information, e.g., due to sensitive, privacy or intellectual property issues and to publish only relevant information.

To address the privacy concerns or hide irrelevant information, a scientist can remove sensitive, private or the irrelevant information by removing respective nodes and edges from the provenance data.
Motivating Example Scenarios. Consider the provenance graph (PG) in Fig. 2(a)\(^1\). Here data nodes are represented as circles and process invocation nodes as boxes. Dependencies among them are shown as directed edges. These edges capture the lineage of data nodes and thus are typically drawn from newer nodes (right) to older nodes (left), i.e., in the opposite direction of the dataflow edges in a workflow specification. For example, \(d_{16}\) was generated by an invocation \(s_2\), and was in turn used by invocation \(c_2\), denoted by \(s_2 \xrightarrow{\text{gen, by}} d_{16}\) and \(d_{16} \xrightarrow{\text{used}} c_2\), respectively.

Assume the scientist wants to share data products \(d_{16}\) and \(d_{19}\) along with their provenance information. This publication request is shown in Fig. 2(a). A recursive query is used to retrieve all the data and invocation nodes upstream from \(d_{18}\) and \(d_{19}\), i.e., the nodes on which the latter depend. The resulting subgraph (PG') is shown in Figure 2(b). Note that the lineage of \(d_{16}\) up to \(s_2\) is not in the lineage of \(d_{18}\) and \(d_{19}\) and hence not included in PG'. Further assume the scientist can’t share PG’ as it has some sensitive, proprietary, and irrelevant (i.e., low-level technical details) information as shown in Fig. 2(b).

Fig. 2(c) shows the provenance graph after (i) removing the value references from data nodes \(d_{11}\) and \(d_{12}\), (ii) abstracting nodes \(s_1\), \(d_{14}\), and \(s_1\) into a group node \(g_1\), and (iii) removing nodes \(c_1\), \(d_{15}\), and \(c_2\). In this modified provenance graph, we see that there is a cycle between the data node \(d_{13}\) and invocation node \(g_1\); a structural error for the edge between invocation nodes \(s_2\) and \(g_1\) (the graph should be bipartite); and there are no dependencies from data nodes \(d_{18}\) and \(d_{19}\) to the rest of the graph. Thus, we need a systematic way to customize provenance while respecting general properties of provenance graph (e.g., acyclicity, bipartiteness, etc.) and preserving correctness and completeness of the provenance information.

**Our Approach.** We develop a framework using which users declaratively can (i) state publication and customization requests, and (ii) specify provenance policies to maintain data integrity. We also develop a system to execute the publication and customization requests and then check for policy violations. We implement two different algorithms to repair the policy violations (a) HidingNodes, and (b) InventingNodes. In HidingNodes, we hide the policy violating nodes and edges to ensure that the customized provenance graph satisfies all the policies. In InventingNodes we introduce non-functional nodes to avoid policy violations. As HidingNodes algorithm hides nodes and edges to fix policy violations, many of the relevant nodes and dependencies may be removed and thus it may make the customized provenance graph less useful. Similarly, the InventingNodes algorithm may make the provenance graph difficult to understand as it adds new non-functional nodes and dependencies. Thus, instead of trying to find a solution with all the publication requests, customization requests, and provenance policies, we evaluate all possible non-empty combinations of the publication requests, customization requests, and provenance policies and compute scores of all the customized provenance graphs towards finding the best possible solutions. This scoring system helps to understand which customized provenance graph is better than another.

**Outline.** In Section 2, we discuss the Open Provenance Model based on the reference specification provided in [18]. In Section 3, we describe all the inputs and outputs of our provenance customization process. In Section 4, we describe the implementation, which consists of five main steps: (i) detect direct conflicts, (ii) compute all combinations of user requests and provenance policies, (iii) customize provenance data, (iv) calculate score of the customized provenance graph, and (v) select top-k combinations of combinations of user requests and provenance policies, i.e., combinations with best scores. In Section 5, we discuss the possible future works that we want to undertake.

### 2. PROVENANCE MODEL

We base our provenance model on the OPM [18]. In OPM, primarily there are two kinds of nodes: data artifacts, and process invocations and there are two kinds of edges: used, and wasGeneratedBy. We represent OPM as an semi-structured data model, which consists of labeled, directed graphs of the form \(PG = (V, E, L)\), with

\(^1\)A simplified version of the provenance graph of the First Provenance Challenge (FPC) [19].
3. PROVENANCE CUSTOMIZATION

The provenance customization framework takes a provenance graph, a set of user requests, and a set of provenance policies, applies all the user requests on the provenance graph and produces a customized provenance graph, which satisfies all the provenance policies. In this section, we describe a provenance graph, an user request, a provenance policy, fix to a provenance policy violation, and a customized provenance graph. We have explained these concepts and their internal workings in further details in [8].

3.1 Provenance Graph

Most of today’s scientific workflow systems capture provenance using their in-built provenance recorders. While executing a workflow, a workflow system reads a set input data products as specified by the workflow specification, produces a set of output data products, and captures the provenance. The provenance is a dependency graph as shown in Fig. 2(a). Here, boxes are the representations of process invocations and circles are the representations of data artifacts. We consider provenance graphs, which are captured based on the provenance model discussed in Section 2.

3.2 User Requests

The user requests (UR) supported by this framework are summarized below. In this framework the user requests are to be formulated as relations (or facts) as shown in Fig. 3(a). We envision that in the future these relations are created by graphical user-interfaces, based on provenance browsing tools (such as [1]), extended with capabilities to mark sets of edges and nodes that are then used to populate the relations associated with the user requests.

Publish. This request allows the user to specify a set of nodes for which lineage information to be fetched from the initial provenance graph using the :lineage(X) construct. Here X is the identifier of a node. As provenance graph may have many output data products and this user request may be for some of them, the lineage for the selected data artifacts is a subset of the initial provenance graph.

Anonymize. For a process invocation node, the unary relation :anonymize(N) removes the reference process(i, name) (i.e., link to the source code of the process). In a similar way, for a data product d this request removes the tuple data(d, x) (i.e., the data value or reference x), which otherwise could be used to obtain the data value. This request does not remove the selected nodes nor the adjacent edges from the provenance graph.

Hide. To remove nodes or edges, which are sensitive or irrelevant from the provenance graph, the hide(X) or hide_deprecated(X, Y) user requests respectively can be used. The system will remove the selected node and all adjacent edges of the selected edges.

Retain. This request is used to explicitly mark nodes or edges in the provenance graph to be retained in the customized provenance graph. To fix provenance policies, this framework may automatically apply deletions to existing nodes and edges. A retain(X) or a retain_deprecated(X, Y) request will create an implied conflict if the marked node or associated edges are to be removed, notifying the user either to relax the provenance policy or to remove the retain user request.

An application of these requests is shown in Fig. 3(b), which captures the requests as described in Fig. 2(a) and Fig. 2(b).

3.3 Provenance Policies

Now, we explain the provenance policies (PP) considered by this framework.

NWC (No-Write Conflict). A write conflict is when there are multiple gen_by edges for a single data product. This can occur if two different data nodes, which were generated by different process invocations, are selected to be abstracted into single group node.

NCD (No-Cycle Dependency). When multiple nodes are contracted to a group node by an abstract user request, cycles can be introduced. For example, the requests to abstract m_1, d_14, and...
z_i into an abstraction node g_i, as shown in Fig. 2(b) creates a cycle between nodes g_i and d_{13} after the abstract request has been executed, in Fig. 2(c).

**NFS (No-False Structure).** A false structure is when there is a node dependent on another of the same type. For example, the requests to abstract n_1, d_{14}, and z_i into an abstraction node g_i as shown in Fig. 2(b) creates a structural conflict between nodes g_i and z_i, as in Fig. 2(c), should the abstract request be executed.

**NFD (No-False Dependency).** A customization from PG to CG exhibits a false dependency, if two concrete (non-abstracted) nodes n_1 and n_2 in CG are transitively dependent on each other but the corresponding nodes in the original provenance graph PG are not.

**NFI (No-False Independency).** These is a false independency when two concrete nodes n_1 and n_2 are not transitively dependent on each other in CG, but there exists a transitive dependency between n_1 and n_2 in the original graph PG. For example, the hide requests to remove the propitious information as shown in Fig. 2(b) makes d_{15} independent of d_{16}, d_{18} independent of d_{13}, etc., as in Fig. 2(c), should the hide requests be executed.

Provenance policy violations are observed using a set of integrity constraints (IC) via witness relations as shown in Fig. 8. For example, the witness relation \( wc(X, Y) \) is used to observe violations for the provenance policy NWC.

### 3.4 Fixing Provenance Policy Violations

Policy violations can be fixed in many ways, e.g., by hiding violating nodes, by introducing new edges, etc. We develop two approaches (i) hiding violating nodes, and (ii) inventing new non-functional nodes (i.e., artificial nodes). To fix the three structural policy (i.e., NWC, NCD, and NFS) violations, we abstract the violating nodes into the adjacent abstraction group. To do so, we create an equivalence relation \( \text{same\_group} \) for the nodes in the modified provenance graph. Nodes that are not taking part in an abstract user request will have their own equivalence classes. All nodes that are mapped to the same group id \( g \) will be in the same class \( \{ n | \text{same\_group}(n, g) \} \).

When applying the updates containing the same group relation, we proceed as follows: A class with only a single member, is replaced by the member itself. Classes that contain more than one member are replaced by the group ID that represents this class. We discuss this approach in further detail in Section 4.4.4 and the inventing new nodes approach in Section 4.4.

### 3.5 Customized Provenance Graph

Now, we describe the properties that a customized provenance graph should have. The customized provenance graph (CG) is a non-empty provenance graph (i.e., with at least an used or a gen_by edge), which is derived from PG after applying all user requests, and which satisfies all provenance policies.

In a general case, if all the user requests are applied, one or many provenance policies may be violated. These violations can be fixed by using the approaches discussed in Section 4.3 and Section 4.4. These fixes may reduce the quality of the customized provenance graphs. Hiding a node means some of the relevant dependency information is removed. Thus, it is not guaranteed that just by applying all the user requests and fixing the policy violations, the best customized provenance graph is found. So, we need to have a way to assess the quality of a customized provenance graph.

To evaluate a CG, we consider three scores: (i) \( u_{\text{score}} \), which specifies the significance of the user requests in CG, (ii) \( p_{\text{score}} \), which specifies the significance of the provenance policies in CG and (iii) \( v_{\text{score}} \), which specifies the significance of relevant nodes. A desired CG must do better than other CGs with respect to these scores.

### 4. IMPLEMENTATION

Our system takes three inputs (i) a provenance graph, (ii) a list of publication and customization user requests, and (iii) a list of provenance policies. The publication and customization user requests are expressed using the constructs as described in Section 3.2, and the provenance policies to maintain data integrity in the output provenance graph are expressed using the constructs as discussed in Section 3.3. The system then computes the custom prove- nance graph. We now describe the system, based on its individual modules and their interactions with each other.

#### 4.1 Direct Conflict Detection

This is the first module and it detects obvious conflicts among all user requests. These can occur if some user requests that require nodes or edges (i.e., lineage, retain) to be carried over to CG from PG but some other user requests (i.e., hide, or abstract) require the same model elements to be deleted in CG.

We detect direct conflicts via the logic rules given in Fig. 5. If a conflict(X), a conflict_dep(X,Y), or a conflict_abst(X) has been derived, we show its derivation tree (including the URs that lead to this conflict) to the user. The user has then the opportunity to prioritize one of the conflicting requests in order to resolve the conflict.

#### 4.2 Prospective Solution Finder

Our system combines user requests and provenance policies to satisfy them simultaneously. Application of user requests may violate one or more provenance policies. It repairs policy violations as described in Section 3.4. However, these repairs in turn may undo certain user requests. For example, a request may introduce a hide(n_i) whereas there is a user specified retain request on
% Auxiliary dep relation
dep'(X,Y) :- lineage(X), dep(X,Y).

dep'(Y,Z) :- dep'(X,Y), dep(Y,Z).

Figure 5: Detection of conflicts among user requests.

Figure 6: Application of lineage(X) user requests.

the same node. A valid model can’t satisfy them together. Thus, combing all Us and Ps or a certain combination of them may not be satisfied together.

In building a systematic approach, we define \( \mathcal{P}(S) \) as the powerset of \( S \), where \( S=\text{UR} \cup \text{PP} \). Let’s assume \( |S|=n \), then there are \( 2^n \) elements in \( \mathcal{P}(S) \). We expect scientists to provide a mapping \( \tau:S \rightarrow \mathbb{Z} \) in addition to providing PG, UR, and PP. Each element in \( \mathcal{P}(S) \) contains a subset of UR and a subset of PP. Let’s assume \( |S|\leq n \), then there are \( 2^n \) elements in \( \mathcal{P}(S) \). We expect scientists to provide a mapping \( \tau:S \rightarrow \mathbb{Z} \) in addition to providing PG, UR, and PP. Each element in \( \mathcal{P}(S) \) contains a subset of UR and a subset of PP. A \( \mathbb{Z} \) value of an element in \( S \) (either a UR or a PP) specifies the relative importance of the element in the customized provenance graph. A higher \( \mathbb{Z} \) value means that the scientist wants to have this element in the customized provenance graph in compare to the elements with a lower \( \mathbb{Z} \) value.

For an element in \( \mathcal{P}(S) \), now the PG is customized by applying all the user requests taken from the element while satisfying all provenance policies in the element using the modules as described below.

### 4.3 Hiding Nodes

This module accepts a PG, an element from \( \mathcal{P}(S) \), which has a set of user requests and a set of provenance policies. It first applies all the \( \text{lineage}(X) \) user requests using the logic rules shown in Fig. 6. Here \( \text{dep}'(X,Y) \) is a subset of the \( \text{dep}(X,Y) \) containing only the edges, which lie on the lineages of nodes as specified by \( \text{lineage}(X) \). Note that, this is not a transitive closure computation. This is a reachability computation starting from the nodes in \( \text{lineage}(X) \).

After all the \( \text{lineage}(X) \) user requests are applied, all the customization user requests are applied on the result using the logic specified by the logic rules as specified in Fig. 7. Now, this application of customization user requests creates an intermediate provenance graph, which may violate one or more provenance policies as described in Section 3.3. These violations can be observed using the the witness relations shown in Fig. 8. Here all relations are computed based on the results of application of the customization user requests. The relation \( \text{tcdep}'(X,Y) \) is computed based on the input provenance graph. Here \( \text{tcdep}(X,Y) \) and \( \text{tcdep}'(X,Y) \) are both transitive closures on \( \text{dep}(X,Y) \) and \( \text{dep}'(X,Y) \) respectively. If any of these witness relations is derived, then there is a provenance policy violation and our system resolves the violation using the process described below.

**Repairing Policy Violation.** To repair policy a violation, we propagate respective user requests (i.e., abstract, hide, etc.) for the violating nodes. We create an equivalence relation \( \text{same\_group} \), as shown in Fig. 9, for all the violated nodes. Nodes that are not taking part in an abstract user request will have their “own” equivalence class. All nodes n that are mapped to the same group id \( g \) will be in the same class \( [g] = \{ n | \text{same\_group}(n,g) \} \). We then develop new requests as follows: For a class with only a single member, we find the user request which caused this violation and extend the user request to apply on this violated node. Classes that contain more than one member are replaced by the group ID that represents this class. Implied conflicts are checked like inconsistency checks.

\[
\begin{align*}
\text{conflict}_{\text{abst}}(X) & \leftarrow \text{conflict}_{\text{dep}}(X, Y) \\
\text{conflict}_{\text{dep}}(X, Y) & \leftarrow \text{conflict}(X) \\
\text{remove}_{\text{dep}}(X, Y) & \leftarrow \text{abstract}(Y, G), \text{dep}'(X, Y) \\
\text{remove}_{\text{dep}}(X, Y) & \leftarrow \text{abstract}(X, G), \text{dep}(X, Y) \\
\% \text{ In conflict user requests}\\n\text{conflict}(X) & \leftarrow \text{remove}(X), \text{keep}(X) \\
\text{conflict}_{\text{dep}}(X, Y) & \leftarrow \text{remove}_{\text{dep}}(X, Y), \text{keep}_{\text{dep}}(X, Y) \\
\text{conflict}_{\text{abst}}(X) & \leftarrow \text{abstract}(X, G_1), \text{abstract}(X, G_2), \text{not } G_1=G_2.
\end{align*}
\]
Stage II. This approach computes \textit{in} and \textit{out} as follows: \textit{in} is a set of nodes, on which one or more nodes from the \textit{hide} relation are dependent, while \textit{out} is another set of nodes, which are dependent on one or more nodes in the \textit{hide} set. For each of the elements of \textit{out}, it computes their dependencies on elements of \textit{in} and captures that in \textit{indep} sets. Computation of this stage can be visually understand in Fig. 10(a).

Stage II. In this stage our system analyzes the dependencies among the nodes of a specific \textit{indep} set of a \textit{out} node \(o\). In case there is a node \(i \in \textit{indep}\) which depends on another node \(i' \in \textit{indep}\), the framework removes \(i'\) from \textit{indep} as \(o\) transitively depends on \(i'\) via \(i\) (in a sense, \(i'\) is "covered" by \(i\), since the lineage of \(i\) includes the lineage of \(i'\)). For example, node \(d_{18}\) from the \textit{out} set has an \textit{indep} set with nodes \(d_{6}, d_{40}, d_{41}, d_{12}\) and \(d_{13}\). Now, since \(d_{13}\) is dependent on all of \(d_{6}, d_{40}, d_{41}\), and \(d_{12}\) as shown in Fig. 2(b), the framework optimizes this \textit{indep} set by removing all nodes except the node \(d_{13}\). This process is performed for all elements from the \textit{out} set. The result is shown in Fig. 10(b). Now, if there is any (transitive) dependency that uses only non-deleted nodes and edges between a node from the \textit{out} set and the nodes of its \textit{indep} set. In case there is the respective node from the \textit{indep} set is removed, since the required dependency is already present in the graph. For example, there is a direct dependency from \(s_{2}\) to \(d_{13}\). Thus \(d_{13}\) is removed from the \textit{indep} set of the invocation node \(s_{2}\). Finally, if an \textit{indep} set for an \textit{out} node becomes empty, the node is removed from the \textit{out} set. This is the case for the invocation node \(s_{2}\) as shown in Fig. 10(c). Since there is already an edge between \(s_{2}\) and \(d_{13}\), the framework does not add an additional edge.

Stage III. In the final stage, our system introduces dependencies (i.e., bringing them back) of all the nodes in \textit{out} to the nodes of \textit{in} based on \textit{indep}. It introduces artificial data nodes for an invocation node in \textit{in} or \textit{out} and connect them and artificial invocation node for a data node in the \textit{out} set and introduces dependencies to all the nodes in \textit{indep} for this \textit{out} node. The result is shown in Fig. 10(d) and this provides the customized provenance graph as shown in Fig. 3(c).

Here, process invocation \(B\) transitively depends on process invocation \(A\) as the process invocation \(B\) depends on data artifact \(d_{13}\), which in turn depends on process invocation \(A\). Process invocation \(C\) also transitively depends on process invocation \(A\) as the process invocation \(C\) depends on data artifact \(d_{16}\), which in turn depends on process invocation \(s_{2}\), which in turn on data artifact \(d_{13}\). Here the dependencies from \(d_{16}\) to \(s_{2}\) and \(s_{2}\) to \(d_{13}\) were not deleted by the user requests. Thus, they were not shown in Fig. 10(d).

In summary, the key ideas are to (i) maintain all relevant nodes, (ii) maintain their dependencies, and (iii) invent new, non-functional nodes to avoid policy violations. We have discussed the complete algorithm and analyzed its execution in [9].

4.4 Inventing Nodes

In this approach, all the abstract user requests are converted into \textit{hide} requests. It executes all the publication (i.e., \textit{lineage}) requests on PG and produces the PG and then applies all the \textit{hide} (both user specified and converted), \textit{anonymize}, and \textit{retain} requests on PG. At this point, the modified graph has only the relevant nodes and some of them may have been anonymized. One or more dependencies (i.e., direct or transitive) among the relevant nodes, which are there in PG, may be missing in the modified graph. Now, this approach reconnects the relevant nodes (i.e., bringing back all the lost dependencies) by introducing minimum number of non-functional (i.e., artificial) data and process invocation nodes. This is a 3-stage process as described below:

Stage I. This approach computes \textit{in} and \textit{out} as follows: \textit{in} is a set of nodes, on which one or more nodes from the \textit{hide} relation are dependent, while \textit{out} is another set of nodes, which are dependent on one or more nodes in the \textit{hide} set. For each of the elements of \textit{out}, it computes their dependencies on elements of \textit{in} and captures that in \textit{indep} sets. Computation of this stage can be visually understand in Fig. 10(a).

Stage II. In this stage our system analyzes the dependencies among the nodes of a specific \textit{indep} set of an \textit{out} node \(o\). In case there is a node \(i \in \textit{indep}\) which depends on another node \(i' \in \textit{indep}\), the framework removes \(i'\) from \textit{indep} as \(o\) transitively depends on \(i'\) via \(i\) (in a sense, \(i'\) is "covered" by \(i\), since the lineage of \(i\) includes the lineage of \(i'\)). For example, node \(d_{18}\) from the \textit{out} set has an \textit{indep} set with nodes \(d_{6}, d_{40}, d_{41}, d_{12}\) and \(d_{13}\). Now, since \(d_{13}\) is dependent on all of \(d_{6}, d_{40}, d_{41}\), and \(d_{12}\) as shown in Fig. 2(b), the framework optimizes this \textit{indep} set by removing all nodes except the node \(d_{13}\). This process is performed for all elements from the \textit{out} set. The result is shown in Fig. 10(b). Now, if there is any (transitive) dependency that uses only non-deleted nodes and edges between a node from the \textit{out} set and the nodes of its \textit{indep} set. In case there is the respective node from the \textit{indep} set is removed, since the required dependency is already present in the graph. For example, there is a direct dependency from \(s_{2}\) to \(d_{13}\). Thus \(d_{13}\) is removed from the \textit{indep} set of the invocation node \(s_{2}\). Finally, if an \textit{indep} set for an \textit{out} node becomes empty, the node is removed from the \textit{out} set. This is the case for the invocation node \(s_{2}\) as shown in Fig. 10(c). Since there is already an edge between \(s_{2}\) and \(d_{13}\), the framework does not add an additional edge.

Stage III. In the final stage, our system introduces dependencies (i.e., bringing them back) of all the nodes in \textit{out} to the nodes of \textit{in} based on \textit{indep}. It introduces artificial data nodes for an invocation node in \textit{in} or \textit{out} and connect them and artificial invocation node for a data node in the \textit{out} set and introduces dependencies to all the nodes in \textit{indep} for this \textit{out} node. The result is shown in Fig. 10(d) and this provides the customized provenance graph as shown in Fig. 3(c).

Here, process invocation \(B\) transitively depends on process invocation \(A\) as the process invocation \(B\) depends on data artifact \(d_{13}\), which in turn depends on process invocation \(A\). Process invocation \(C\) also transitively depends on process invocation \(A\) as the process invocation \(C\) depends on data artifact \(d_{16}\), which in turn depends on process invocation \(s_{2}\), which in turn on data artifact \(d_{13}\). Here the dependencies from \(d_{16}\) to \(s_{2}\) and \(s_{2}\) to \(d_{13}\) were not deleted by the user requests. Thus, they were not shown in Fig. 10(d).

In summary, the key ideas are to (i) maintain all relevant nodes, (ii) maintain their dependencies, and (iii) invent new, non-functional nodes to avoid policy violations. We have discussed the complete algorithm and analyzed its execution in [9].

4.5 Score Assignment

The HidingNodes and InventingNodes take a provenance graph (PG), a set of conflict free user requests (UR), and a set of provenance policies (PP) as inputs and outputs a customized provenance graph (CG), a set of satisfied user requests (\(U_s\)) and a set of satisfied provenance policies (\(P_s\)). These approaches may hide some of the relevant nodes or invent artificial nodes respectively in order to satisfy all the provanence policies. In PG, the nodes for which there are no \textit{hide} or \textit{abstract} user requests are considered as relevant nodes (\(V^CG_s\)). In CG, all nodes, which are not invented, are considered as relevant nodes (\(V^CG_s\)).

Consider \(e_i\) is the \(i^{th}\) element of \(P(S)\) with \(CG_1\) is the customized provenance graph produced by HidingNodes module and \(CG_2\) is the customized provenance graph produced by InventingNodes module. Now, we compute \(u_{score}\), which specifies the significance of the user requests in CG and \(p_{score}\), which specifies the significance of the provenance policies in CG. We compute them as \(u_{score} = \sum_{e \in U_s} \tau(e)\) and \(u_{score} = \sum_{e \in P_s} \tau(e)\). We also define \(r_{score}\) as the node relevancy weight and \(r_{score} = \|V^CG_s\| \div |V^CG_s|\).
In the similar way, we compute these three scores for all the elements in \( P(S) \).

### 4.6 Best Solution Generator

Consider \( e_i \) and \( e_j \) are the \( i^{th} \) and \( j^{th} \) elements of \( P(S) \). We compare all three scores (i.e., \( u \), \( p \), and \( r \)) between \( CG_i \) and \( CG_j \). If \( CG_i \) is better (i.e., strictly greater than) in at least one score and equal in all other scores, then \( CG_i \) dominates \( CG_j \). Thus, user should select \( e_i \) over \( e_j \). But, if there is at least one score for which \( CG_i \) is better than \( CG_j \), and there is at least another score for which \( CG_i \) is better than \( CG_j \), then we can’t say which one is better. Thus, both CGs should be reviewed by the user to pick the best for his requirements. To find the elements of \( P(S) \), which are not dominated in all three scores by some other elements, we use the skyline computation as developed by Borzsonyi et al. [2].

After computing CGs for all elements of \( P(S) \), we compute the respective \( u \), \( p \), and \( r \) scores and compute the skyline.

In the skyline, we get only the CGs which are not dominated by any other CGs in all three scores. Let’s assume we store the result of this skyline computation in a skyline table, then we can issue the following query to get the skyline: `SELECT * FROM skyline. SKYLINE OF u, p, r. MAX. MAX. MAX. MAX.` The user now should review all the CGs in skyline and select the best one for his need.

### 5. Future Work

**Higher Level User Request.** In our current implementation, user requests are expressed at the individual node and edge level. As the size of a provenance graph may be very big, it is often infeasible for scientists to specify provenance policies and customization user requests at this level. Instead, provenance graph properties along with the domain (i.e., the scientific domain) properties can be utilized to develop an higher order declarative user request specification language, using which scientists may specify their publication and customization requests at a higher level.

**Optimizing Solution Space.** Currently, our system is evaluating all \( 2^n \) elements of \( P(S) \) and is computing respective \( u \), \( p \), and \( r \) scores, which can be very costly. Instead, we want to study the properties of elements of \( P(S) \) to verify if evaluating a subset of \( P(S) \) will be sufficient. For example, let’s consider \( e \) is an element of \( P(S) \). After evaluating this element we compute \( u \), \( p \), and \( r \) scores. If the \( r \) score is 1, then the CG is best possible customized graph for any proper subset of \( e \). An exhaustive study in this direction is very important to discover such properties so that all of \( 2^n \) elements of \( P(S) \) do not have to be evaluated.

**Privacy of Workflow Specification.** Scientific workflow systems often automatically record provenance data [15, 6], and a provenance graph may resemble the workflow graph, i.e., the former can be seen as an instance of the latter [17]. Using the framework discussed above, we customize provenance graphs to remove private and irrelevant information. Thus, scientists may have two issues: (i) this customized provenance graph may not be a proper instance of the workflow specification, and (ii) workflow specification may allows unintended inferences causing privacy issues. Thus, before sharing the modified provenance graph along with the workflow specification, we need to customize the workflow specification.

### 6. Conclusion

Provenance is extremely useful [3, 20, 4, 11, 5]. But, there are issues, such as privacy, and relevancy. Towards finding a balance between (i) sharing sufficient provenance, which helps to understand the data products for which the provenance data is, and (ii) protecting the sensitive data from the provenance data, we develop a declarative approach to publish and customize the provenance data. Our declarative approach provides constructs to specify the publication, customization, and data integrity (in form of provenance policies) requirements. It also provides two implementation approaches using which provenance graph can be customized so that all the publication and customization requests are satisfied. These approaches finds solution by balancing the need to share more information where as maintaining the required data integrity. We have extended our effort towards finding the best possible solution by exploring all possible combinations of the user requests and provenance policies and computing the skyline. In our future work, we want to investigate techniques to avoid the exponential step in our proposed solution.


