Searching Workflows with Hierarchical Views *

Ziyang Liu Qihong Shao Yi Chen
Arizona State University
{ziyang.liu,qihong.shao,yi}@asu.edu

ABSTRACT

Workflows are prevalent in diverse applications, which can be scientific experiments, business processes, web services, or recipes. With the dramatically growing number of workflows, there is an increasing need for people to search a workflow repository using keywords and to retrieve the relevant ones. A workflow hierarchy is a three dimensional object containing multiple abstraction views of different granularity on the same workflow. This unique structure poses a new set of challenges compared to keyword search on tree or graph structures typically found in relational or XML data.

In this paper, we define an informative, self-contained and concise search result on workflows to be a projection of a workflow hierarchy on a two dimensional viewing plane inferred from user queries. We then design and develop an efficient keyword search engine for workflows. Experimental evaluation demonstrates the effectiveness of our approach.

1. INTRODUCTION

Workflows with hierarchical multi-resolution views (referred to as workflow hierarchies) are widely used in scientific [10, 21, 34, 38, 14, 37] and business [12, 38] domains, which ease the analysis, maintenance and reusability of workflows. As an example, a workflow hierarchy describing the recipe of curry chicken is shown in Fig. 1.[1] A node represents a task, which can be a step in a recipe, a web service invocation, a database query, a program run, or an experiment step, etc. A directed solid edge between nodes represents their dependency, dataflows, or control flows (AND/OR/XOR), referred as *dataflow edge*. For instance, in the bottom box in Figure 1, after tasks *add tenderizer* (0.1.0.0), we need to *wait 10 min* (0.1.0.3), and then have the data fed into the next task *put into skillet* (0.1.1.0).

In order to reduce the analysis complexity, enable modularity and re-use [9, 33], simplify provenance analysis [10, 13, 17], and achieve security [15], *composite task* is often defined to abstract a group of tasks into a single task, as supported in many workflow management systems, such as Kepler [2] and myExperiment [4]. For example, composite task *tenderize chicken breast* (0.1.0) is an abstraction of a group of tasks consisting of *add tenderizer*, *sprinkle curry powder*, *add garlic* and *wait 10 min*. Dotted lines connecting group of tasks to its abstraction composite task are referred to as *abstraction edges*. Composite tasks can be recursively defined to form a workflow hierarchy, such as the one in Figure 1. On the other hand, the most detailed tasks (those shown in italic in bottom box Figure 1) are called *atomic tasks*.

It is highly desirable if a user can search relevant workflow hierarchies in a repository using keywords, and then re-use or revise them as needed when designing new workflows, so that the design phase will be easier and be shortened compared with designing new ones from scratch.Suppose that a user would like to make a dish using chicken breast and coconut milk by sauting, but doesn't have a recipe in mind. She would issue a keyword query "*chicken breast, coconut milk, saute*" (Q_1 in Figure 2(a)) on a repository of recipes to find useful ones.

We can easily find the workflow hierarchies in the repository that contain matches to query keywords. Suppose Fig. 1 is one of such workflows in the repository, where keyword matches are in bold font. Obviously, returning the whole workflow hierarchy as a query result is not *concise*, as an overwhelming volume of information is delivered to the user (e.g., many workflow hierarchies in the repository [2, 7, 6, 4] contain hundreds of nodes).

The immediate challenge is how to define query results for keyword search on workflows. Given much research done on keyword search on graph-structured data (e.g., relational) and tree-structured data (e.g.,XML), a natural question is whether we can adopt their approaches: defining a query result on workflow hierarchies as a smallest tree in the data that contains the query keywords. A result for query Q_1 "*chicken breast, coconut milk, saute*" using these approaches is shown in Fig. 2(d).

However, such query results are not desirable for two reasons. First, the results do not necessarily capture the dataflows among keyword matches, and thus fail to be *informative* on node relationships. For example, the relationship of tasks containing *chicken* (0.1.0) and *saute* (0.1.1.2) is expressed as a path of both dataflow edges and cross-layer expansion edges, while their dataflow is not captured, which should be: *tenderize chicken breast* (0.1.0) → *put into skillet* (0.1.1.0) → *add green pepper & union* (0.1.1.1) → *saute until tender* (0.1.1.2).

Besides, returning smallest subtrees does not necessarily produce *self-contained* query results. Consider another query Q_2 "*brown rice, bake*" in Figure 2(a). The smallest subtree is the path from *cook brown rice* (0.3.2) to *bake* (0.3.4). However, such a path itself does not have a semantic meaning. A clear meaning can only be

*This material is based on work partially supported by NSF CAREER award IIS-0845647, IIS-0740129, IIS-0915438.

[1]Every node in the figure is associated with an identifier, whose construction will be presented in Section 4.1.

Proceedings of the VLDB Endowment, Vol. 3, No. 1

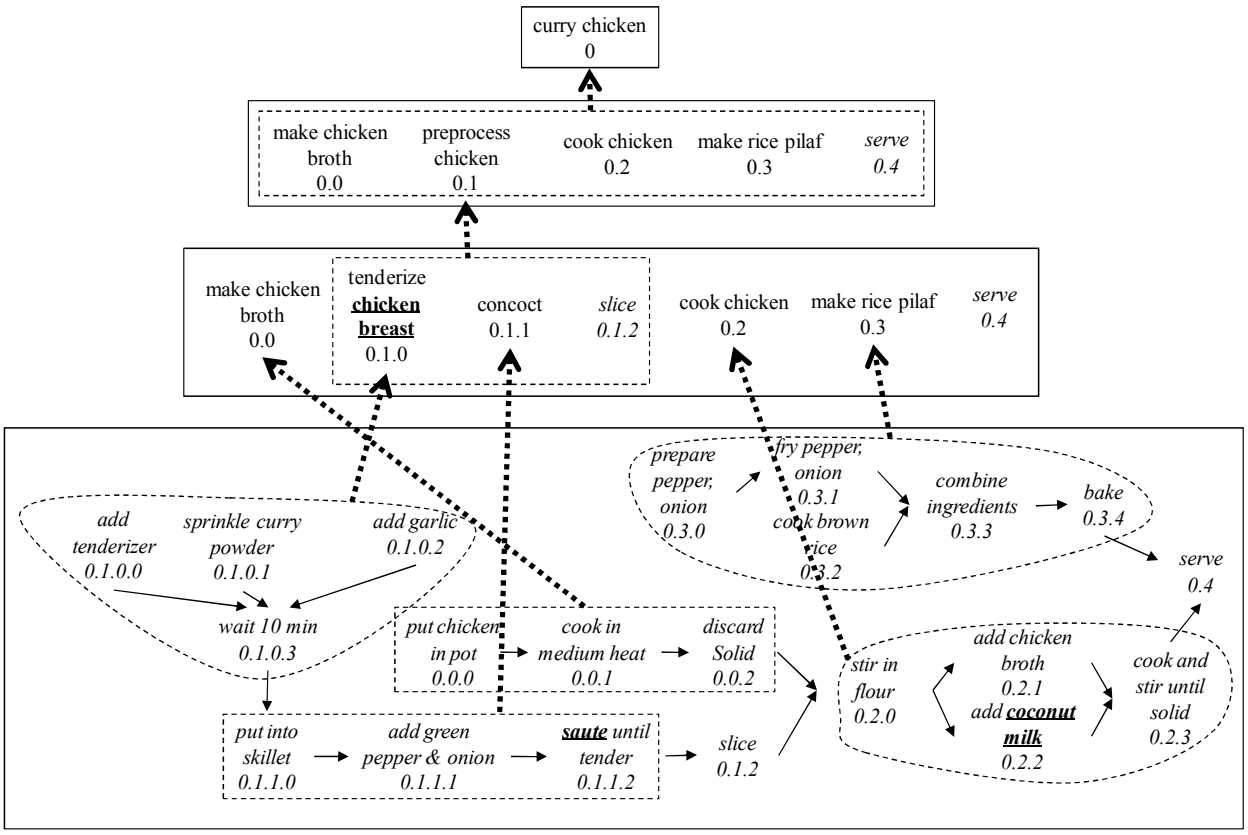


Figure 1: A Workflow Hierarchy Describing a Recipe

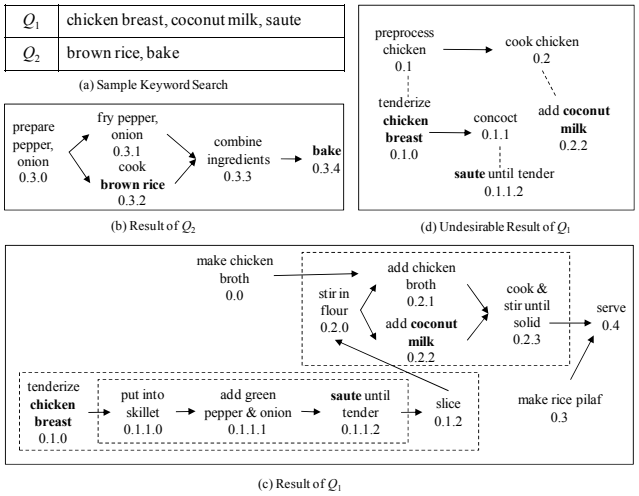


Figure 2: Sample Queries and Results

obtained if we consider the nodes in this path together with nodes (0.3.0, 0.3.1) (shown Figure 2(b)), which correspond to a composite task *make rice pilaf* (0.3), meaning that these nodes as a whole is a workflow about *making rice pilaf*.

According to our user studies (Section 5.2), when a user issues a keyword query on a repository of workflow hierarchies, each of which has abstractions with different granularities of atomic tasks, it is likely that s/he is interested in retrieving *workflow views* that contain these keywords and show their relationships. In our example, it is desirable to present Figure 2(c) as a query result of Q_1. This result explicitly captures the dataflow among keyword matches (shown in bold). For example, after *saute (the chicken) until tender* (0.1.1.2), we *slice (it)* (0.1.2), *stir (it) in flour* (0.2.0) and then *add coconut milk (0.2.2)*. Note that the dataflows among nodes in different composite tasks (i.e., different dashed line boxes) are not explicitly shown in the workflow hierarchy, but are *dynamically synthesized* from the workflow hierarchy. Such a query result is also self-contained, corresponding to a composite task *curry chicken*.

In this paper, we present WISE, a Workflow Information keyword Search Engine, which, to the best of our knowledge, is the first work that returns query results capturing query keywords and their dataflows. WISE has been demonstrated at ICDE '09 [36]. The contributions of this paper include:

First, we address an open problem of defining query results for keyword search on hierarchical workflows. As we have discussed, a good query result should be informative (i.e., capturing the keyword matches and their dataflows), self-contained (i.e., having a name/goal), and concise (i.e., the minimal graph that is informative and self-contained). To achieve this, we start with formally defining the concept of a *view* of workflows, which is a graph defined in the same spirit as defining tree using Tree-Adjoining Grammar (TAG) [22] and defining strings using context-free grammar. Then we define a workflow search result as a *minimal view* of a workflow that contains query keywords.

Second, we develop efficient algorithms for query result generation. Unlike generating search results on graphs/trees, where

only extractions of source data are needed, WISE dynamically constructs query results by *synthesizing* the dataflows among keyword matches. Given the workflow hierarchies containing all keywords in the query, the algorithm for generating results has optimal time complexity.

Experiments show the effectiveness and efficiency of WISE, compared with existing workflow search engines [2, 7, 6, 4] and a search engine on graph data [20].

Although the running example in this paper is a simple recipe workflow hierarchy, the techniques that we propose is applicable for all workflow hierarchies. Such a multi-resolution data structure is a generalization of graphs and trees, and is widely used in many domains, such as scientific experiments, web services, spatial and temporal data, hierarchical plans, etc. For instance, in spatial data, there are edges among the data points in a graph, and a graph can be abstracted to a data point in a recursive way.

The paper proceeds as follows. We describe our data models in Section 2, then define results of keyword search on workflow hierarchies in Section 3. Efficient query generation algorithms are presented in Section 4, followed by experimental evaluations in Section 5. Section 6 concludes the paper.

2. KEYWORD QUERY AND DATA MODEL

In this section, we formally define the data model.

Definition 2.1: [workflow] A *workflow* $W = (V, E)$ is a directed graph where each node represents a task and each edge indicates the dataflow, dependency or control flow between two tasks. ∎

The bottom box in Figure 1 shows a workflow of curry chicken recipe.

Definition 2.2: [composite task] A *composite task* c is an abstraction of a group of tasks S, denoted as $c = abs(S)$. ∎

If $c = abs(S)$, c is called the *parent* of the nodes in S, and nodes in S are called the *children* of c. Ancestors and descendants are recursively defined.

In Figure 1, each group of tasks within a dotted border is abstracted into a composite task, pointed to from the group of tasks by a dotted edge. Composite tasks can be recursively defined.

Definition 2.3: [workflow hierarchy] A *workflow hierarchy* $H = (W, root)$ consists of a workflow $W(V, E)$ and a set of composite task specifications. Nodes in V are called *atomic tasks* or *leaf tasks*. $root \in V$ is the root of the hierarchy, whose name represents the name/goal of the workflow hierarchy. The edge set $E = \{E_a, E_d\}$ consists of both *abstraction edges* (E_a) and *dataflow edges* (E_d). An *abstraction edge* (S, $abs(S)$) $\in E_a$ connects a set of nodes to their corresponding composite task. A *dataflow edge* (u, v, d) represents that an output of task u is an input of task v, where $u, v \in V$, and d denotes the data item sent from u to v.[2] A *subworkflow hierarchy* of H is a workflow hierarchy whose root is a node in V. ∎

Note that two composite tasks do not overlap, i.e., $\forall c = abs(S)$ and $c' = abs(S')$, $c \neq c'$, $S \cap S' = \emptyset$. If $c = abs(S)$ and $u \in S$, we say S is the *cluster* of u.

Each node in a workflow hierarchy can have annotations, which record its name, conditions of the task execution, or possibly a deadline, indicating that the task must be finished no later than the deadline during the execution of the workflow (referred to as "Event-based workflow"), etc. The data items transferred between tasks can be of different types, such as materials in the recipe, data files in the experiment, gene sequences, etc. Note that there can be multiple dataflow edges between two nodes if multiple data items are transferred. There can also be annotations on edges, which specify the control flow (AND/OR/XOR) between two tasks.

Figure 1 shows a workflow hierarchy. The bottom box is a workflow, and each dotted edge represents a composite task specification. Note that the edges that involve composite tasks (e.g., the edge from 0.1.2 to 0.2) are not part of the workflow hierarchy, but their relationship can be derived from the edges between atomic tasks, which is illustrated in Definition 3.2.

Although a workflow hierarchy bears some similarity with a tree model, they have some key differences. First, the relationships among "sibling nodes" are different. Siblings in a tree structure are modeled as either a linearly ordered list or a set; whereas the siblings in a workflow hierarchy represent a (possibly cyclic) graph, where the dataflow edges explicitly capture their relationships. Second, the semantics of parent-child relationship are different. A parent-child relationship in a tree generally specifies the relationship between two distinct objects. In a workflow hierarchy, the task represented by a child is part of the detailed procedure of performing the task represented by the parent. As we explained in Section 1, these differences invalidate techniques for keyword search on trees/graphs, pose unique challenges to query processing and demand novel approaches.

3. SEARCH RESULTS OF WISE

Now we discuss how to define query results for keyword search on workflow hierarchies. Each result should satisfy three properties: (1) informative: the result should contain all dataflows between any two tasks matching keywords, so that the user gets the relationships of the query keywords; (2) self-contained: the result should contain all the tasks for achieving a goal; (3) concise: removing any edges from the result will make it violate informativeness or self-containedness.

To achieve these goals, we first identify the minimal workflow hierarchies that contain all query keywords in Section 3.1, then define its minimal views as query results in Section 3.2.

3.1 Identifying Minimal Workflow Hierarchies

To be *informative* and *concise*, we first identify the smallest workflow hierarchies in the repository that contain at least one match to each query keyword.

Definition 3.1: [Minimal Workflow Hierarchy] A *minimal workflow hierarchy* $H = (V, E, root)$ of a keyword search Q on a repository of workflows R is a workflow hierarchy, such that

1. Every keyword in Q has at least one match in H;
2. There does not exist a subworkflow hierarchy of H that satisfies condition 1. ∎

Note that there are typically multiple minimal workflow hierarchies when processing a keyword query on a repository of workflows. Each minimal workflow hierarchy will derive a query result, as to be discussed in Section 3.2, all of which compose the set of results for the query.

For example, consider Q_2 "*brown rice, bake*" in Fig. 2(a). There are two workflow hierarchies in Figure 1 containing keyword matches *brown rice* and *bake*: the one rooted at *curry chicken* (0) and the one rooted at *make rice pilaf* (0.3). The one rooted at *curry chicken* (0) is not considered as a minimal workflow hierarchy as it does not satisfy condition 2 in Definition 3.1: it has a subworkflow rooted at *make rice pilaf* (0.3) containing all query keywords.

[2]For edges that are from or to an external node of the workflow hierarchy, its corresponding u or v nodes are captured by dummy nodes added to the workflow.

Note that we do not "compose" a new workflow hierarchy from several unconnected sub-workflow hierarchies in the repository, in order to guarantee that each query result has a clear semantic meaning.

3.2 Defining Query Results as Minimal Views

3.2.1 Minimal Views

Unfortunately, returning the whole minimal workflow hierarchy itself to users is neither *informative* nor *concise*. Consider Q_1 as an example, where a minimal workflow hierarchy is Fig. 1. Returning the entire *curry chicken* hierarchy makes it difficult for users to find the dataflows among keyword matches, as they have to go up and down the layers and manually construct the dataflows. These manual operations are tedious and time-consuming for the users especially when the workflow hierarchy is large and complex.

Under this observation, we define the notion of *view* of a workflow hierarchy. Views can be considered as a projection of the 3D workflow hierarchy on to a 2D plane which hides less important information and simplifies analysis.

Definition 3.2: [View] A view $View = (V, E)$ of a workflow hierarchy $H = (V', E', root)$ is a directed graph with labels on edges. The node set V, $V \subseteq V'$ satisfies:

1. $\nexists u, v \in V$, u is an ancestor of v in H.
2. $\forall u \in V'$ and u is a leaf node, $\exists v \in V$ such that v is an ancestor-or-self of u.
3. For any $u, v \in V$, $(u, v, d) \in E_d$ if and only if $\exists u', v' \in V'$, u' and v' are descendant-or-self of u and v, respectively, and $(u', v', d) \in E'_d$. ■

Condition 1 indicates that a view is a two dimensional projection of the three dimensional workflow hierarchy, flattening out the nested hierarchy for the ease of user comprehension. Nodes in a view can have dataflow relationships, but not abstraction relationships. Condition 2 ensures that the node set V of a view covers all leaf nodes (atomic tasks) in the workflow hierarchy H: every leaf node in V' must have one corresponding zoomed-out node in V. Since views may contain composite nodes whose edges are not explicitly present in the workflow hierarchy, their edges need to be induced, as specified in Condition 3. For example, since there is a dataflow between *wait 10 min* (0.1.0.3) and *put into skillet* (0.1.1.0), there should also be a dataflow between their parents, *tenderize chicken breast* (0.1.0) and *concoct* (0.1.1). Condition 3 guarantees that the dataflow edge set E_d in a view is faithful with respect to edge set E'_d according to H: the view preserves the dataflow among nodes in the view. Note that a view may hide the dataflows of two nodes in the workflow that are abstracted into a single node in the view, e.g., the edge between nodes 0.1.0.0 and 0.1.0.3 in Figure 2(a). Conditions 2 and 3 together ensure that a view is "semantically complete" with respect to the name/goal of H, and hence a self-contained information unit whose name/goal is the same as H.

Note that Definition 3.2 bears some similarity with the TAG [22] and context-free grammars. Context-free grammars have rules for rewriting symbols as strings of other symbols, tree-adjoining grammars have rules for rewriting the nodes of trees as other trees, and the proposed workflow views allow rewriting the nodes of a workflow as other graphs. That is, a view is a graph defined on a nested graph hierarchy, analogous to the frontier defined on a tree in TAG, and to a string defined in a context-free grammar.

As we can see, a view is a projection of a three dimensional workflow hierarchy to a two dimensional plane that preserves the dataflows among the tasks in the view. Obviously a workflow hierarchy can have many views, as there are many projections of a three dimensional object, depending on the viewing plane. For example, the tasks in each solid rectangle in Figure 1 compose a view of the *curry chicken* workflow hierarchy. Fig. 2(c) is another view of *curry chicken*.

We now define a keyword search result on workflow hierarchies as a minimal view of a minimal workflow hierarchy that preserves all keyword matches in the workflow hierarchy (which is also a philosophy of keyword search on relational databases or XML, where a result is a minimal tree that contains at least one match to each keyword).

Definition 3.3: [Minimal View] For a keyword search Q on a repository of workflows R, the *minimal view* $View(H, Q) = (V, E)$ of a minimal workflow hierarchy H (Definition 3.1) is the view with the smallest number of tasks over all the views of H that contain all the keyword matches of Q in H.[3] ■

Definition 3.4: [Query Result] For a keyword search Q on a repository of workflows R, the set of query results consists of the minimal view of each minimal workflow hierarchies in the repository. ■

Note that such a result definition is general for all types of workflow hierarchies where the nodes and edges may have annotations as discussed in Section 2. Given a minimal workflow hierarchy and a query, the minimal view is unique, which can be considered as a projection of a three dimensional workflow hierarchy on a two dimensional viewing plane defined by the query. The result is *informative* since it captures all keyword matches and their dataflows. The result is *self-contained* since the view serves an integrated goal and has a unique name, which is the same as the corresponding minimal workflow hierarchy. Furthermore, the result is *concise*, as we opt to use the *minimal view* among all views of each minimal workflow hierarchy.

Besides being informative, self-contained and concise, the query results generated by WISE satisfy another three desirable properties proposed in the literature: *soundness* [37], *monotonicity* and *consistency* [27], as shown in the technical report of this paper [29].

4. ALGORITHMS

After defining query results for keyword search on workflow hierarchies, we present the algorithms of the WISE system that achieve the semantics efficiently.

4.1 Data Processing

We design labeling schemes and indexes for workflow hierarchies to efficiently find minimal workflow hierarchies and their minimal views.

Labeling of nodes and edges. Each node n in the workflow hierarchy is assigned a unique label $NID(n)$. Since we need to explore the ancestor-descendant relationships of nodes to generate results, we use the Dewey labeling scheme, as shown underneath each node in Fig. 1. The label of the root is 0, and the label of a node n is composed by the concatenation of the label of its parent and a unique integer ID within the cluster that n is in. The unique in-cluster ID of a node can be arbitrarily set, i.e., the nodes in a cluster can have an arbitrary order, independent of the dataflows (thus nodes can be ordered even in a cyclic graph). NIDs of nodes are ordered alphabetically. The node labels don't record dataflow information, but parent-child information. They enable efficient retrieval of lowest common ancestor (LCA) of two nodes u and v, whose node label

[3]Note that since the nodes in a view can not have ancestor-descendant relationships, only keyword matches that do not have descendant keyword matches are selected as nodes in a view, which are annotated with their ancestor keyword matches (if any).

is the longest common prefix of $NID(u)$ and $NID(v)$. Each edge is also assigned a unique integer ID.

Leaf adjacency lists of nodes. To efficiently find dataflows among keyword matches that may not be explicitly present in the data, we build a *leaf adjacency list* for each node n in the workflow hierarchy, denoted as $LAL(n)$. $LAL(n)$ consists of IDs of the edges between leaf nodes u and v, such that u is a descendant of n and v is not a descendant of n. For each edge in $LAL(n)$, we also record its direction, as well as the data items transferred. For example, suppose the ID of the edge from node 0.1.0.3 to node 0.1.1.0 in Figure 1 is e_1 and it transfers data item *chicken breast*, then e_1 (outgoing, chicken breast) $\in LAL(0.1.0.3)$ and $LAL(0.1.0)$, where "outgoing" means that it is an outgoing edge from 0.1.0.3 to 0.1.0. Similarly, e_1 (incoming, chicken breast) $\in LAL(0.1.1.0)$ and $LAL(0.1.1)$. Leaf adjacency lists are used to efficiently derive dataflow edges in a query result, as will be discussed in Section 4.2. Intuitively, according to Definition 3.2 there is a dataflow edge between two composite nodes u and v if and only if there is an edge e between their leaf descendants, and such an edge e is recorded in $LAL(u)$ and $LAL(v)$. By leveraging leaf adjacency lists and a hash table, an edge can be derived in O(1) time.

Indexes. To speed up query processing, an inverted index is built which maps a keyword to the list of nodes in the workflow repository whose names/descriptions contain the keyword, sorted by their NID. We also build a B+ tree index on NIDs that retrieves the subworkflow rooted at node NID, referred to as Dewey index.

The leaf adjacency list and indexes are built offline. They both take an affordable amount of space: in the worst case, each dataflow edge is recorded in every ancestor of each endpoint of the edge. Thus the leaf adjacency list takes $O(|E_d|h)$ space where $|E_d|$ is the number of dataflow edges in the workflow hierarchy, and h is the height of the workflow hierarchy. If each node contains at most p keywords, then the inverted index takes $O(|V|p)$ space where $|V|$ is the number of nodes in the workflow hierarchy. The Dewey index takes $O(|V|)$ space.

4.2 Query Processing

WISE uses Algorithm 1 (the pseudo code is presented in the Appendix) to retrieve relevant query results for keyword searches on workflow hierarchies. It consists of two steps: identifying minimal workflow hierarchies, and constructing the minimal view for each minimal workflow hierarchy. We use Q_1: "*chicken breast, coconut milk, saute*" as a running example.

Retrieving minimal workflow hierarchies. We begin by obtaining the list of match nodes for each keyword using the inverted index. In our running example (Q_1), we obtain the matches to *chicken breast*: 0.1.0, *coconut milk*: 0.2.2 and *saute*: 0.1.1.2.

Note that sometimes a user may issue a query whose keywords do not exactly match the words in the data, but are semantically related. This can be addressed by looking each keyword up in a dictionary of synonyms. For instance, if the user query contains keyword "saute", we look it up in the dictionary and find its synonyms, e.g., "fry" and "panfry". Then we search the inverted index for the matches to "saute", "fry" and "panfry", and take the union of their matches as the matches to keyword "saute". In the experiments we have tested the efficiency of WISE when synonyms are considered in query processing. Alternatively, we can also use an ontology, which not only records the similarity among keywords but also the containment relationships (e.g. "saute" is a special type of "cook"). Furthermore, the similarity measurement can be used as part of the ranking scheme.

Then procedure $findMWHs$ identifies the minimal workflow hierarchies using the Indexed Lookup Eager Algorithm [40], considering only expansion edges without dataflow edges. In our running example, there is only one minimal workflow hierarchy, which is the entire *curry chicken* hierarchy, as none of the descendants of *curry chicken* (0) contains all three query keywords. As discussed, returning the entire minimal workflow hierarchy is not informative or concise, thus we propose novel algorithms for computing minimal views of each minimal workflow hierarchy.

Identifying minimal views of minimal workflow hierarchies. After identifying minimal workflow hierarchies, procedure $grouping$ groups the keyword matches according to the minimal workflow hierarchies that they belong to. This is done by first merging the lists of keyword matches into a single list $mergedList$, and then grouping the matches using a single traversal of $mergedList$ and the list of the roots of minimal workflow hierarchies.

Finally, we need to identify minimal views of minimal workflow hierarchies. For each task in the minimal workflow hierarchy, we need to determine whether to include it in the view or not, and extract or synthesize the dataflow among the nodes in the view. $genMV$ performs a single depth-first traversal of each minimal workflow hierarchy in the order of NID, and a traversal on the list of keyword matches sorted by NID. Let cn be the node in the workflow hierarchy currently being visited and $currMatch$ be the current keyword match being visited in $mergedList$. During the traversal:

(1) If cn has descendant matches (which is true if cn is an ancestor of $currMatch$, or $cn = currMatch$ and is an ancestor node of the next node in $mergedList$), we do not output cn, but update cn to be the first child of the current cn, i.e., continue to traverse its subworkflow hierarchy. If cn matches a keyword, $currMatch$ is updated to be the next keyword match.

(2) If cn is not a keyword match and has no descendant match (which is true if cn is not an ancestor-or-self of $currMatch$), then we output cn, skip its subworkflow hierarchy and move to the next node in the workflow hierarchy which is not a descendant of cn.

(3) If cn is a keyword match and does not have descendant matches (which is true if $cn = currMatch$, and is not an ancestor of the next match node), then cn is directly interested by the user, and is output as part of the query result. The properties of match nodes can be displayed upon click. Since we do not output the expansion of cn, we move to the next node in the workflow hierarchy which is not a descendant of cn. We also move $currMatch$ to point to the next keyword match in $mergedList$.

In our running example, we have found the minimal workflow hierarchy, rooted at *curry chicken* (0). The keyword matches in the order of their NID are: *chicken breast* (0.1.0), *saute* (0.1.1.2), *coconut milk* (0.2.2). We traverse the minimal workflow hierarchy and the keyword match list in parallel. Initially, cn = *curry chicken* (0) and $currMatch$ = 0.1.0. Since cn is an ancestor of $currMatch$, we do not output *curry chicken*, but expand it and traverse its children. Later on when we come to cn = 0.1.0, since $cn = currMatch$, we output cn as a clickable node, then move cn to 0.1.1 and $currMatch$ to *saute* (0.1.1.2). The procedure continues until all nodes in the results are identified.

Next we discuss how to generate edges in the result. When outputting a node cn, we need to find the edges corresponding to cn. A naive approach would search each leaf descendant of every node u that has been output as well as each leaf descendant of cn, and check whether there is an edge between them. If so, it means an edge should exist between u and cn in the view (as discussed before, a view should preserve the edges between two nodes in the workflow hierarchy that are descendants of different nodes in the view). This approach is very inefficient, as u and cn may both have a large number of descendants, and they may be accessed multiple

times.

We propose a much more efficient approach using LAL and a hash table, which will be shown to find each edge in $O(1)$. When outputting a node cn we traverse $LAL(cn)$; for each edge with ID e_i in $LAL(cn)$, we check it in a hash table, which maps an edge ID to an endpoint of the edge that has been output. The hash table is initially empty. If e_i is not in the hash table, it means the other endpoint of e_i has not been output, and we put an entry (e_i, cn) into the hash table. If e_i is in the hash table with entry (e_i, u), then u has been output, and there should be a dataflow edge between u and cn, whose direction and data item depends on the corresponding entry in $LAL(cn)$.

For example, when we output *tendrize chicken breast* (0.1.0), we check its LAL. Suppose there is an edge from 0.1.0.3 to 0.1.1.0 with ID e_1 and data item *chicken breast*, then $LAL(0.1.0) = \{e_1$ (outgoing, chicken breast)$\}$. Since node 0.1.1.0 has not been output yet, e_1 is not in the hash table, and we insert entry $(e_1, 0.1.0)$ into the hash table. When we output *put into skillet* (0.1.1.0), since $LAL(0.1.1.0) = \{e_1$ (incoming, chicken breast)$\}$, we check e_1 in the hash table, and get the entry $(e_1, 0.1.0)$. Therefore, we output an edge from 0.1.0 to 0.1.1.0 with data item "chicken breast".

Theorem 4.1: The results generated by Algorithm 1 for a keyword search Q on a repository of workflows R are the minimal views of all minimal workflow hierarchies $H(R, Q)$ in the repository (Definition 3.4).

PROOF. The proof can be found in the Appendix. ∎

Theorem 4.2: The time complexity of procedure $genMV$ is $O(N+E)$, where N, E are the number of nodes and edges in the output (minimal view), i.e., the optimal time complexity for finding minimal views. The overall time complexity of Algorithm 1 is $O(M_{min}kd\log M_{max} + M + N + E)$, where M_{min} and M_{max} are the minimum and maximum number of matches to a keyword, respectively, M is the total number of keyword matches, d is the depth of the workflow hierarchy.

PROOF. The proof can be found in the Appendix. ∎

5. EXPERIMENTS

To evaluate the effectiveness and efficiency of WISE, we compare its performance with three methods in the literature. *Node Return* outputs individual tasks that contain query keywords without dataflow information, as supported by the search modules in Kepler [2], Triana [7] and Taverna [6]. *Structure Return* outputs a whole workflow hierarchy if it contains all the query keywords, which is used in myExperiment [4]. The third approach, BLINKS [20], is a state-of-the-art keyword search engine on graphs, which output minimal subtrees in the data graph whose leaf nodes contain matches to query keywords. We implemented Structure Return and Node Return approaches with best-effort, each of which only returns information of the minimal workflow hierarchies for better precision. The implementation of BLINKS is obtained from the authors.

We have tested two aspects: the quality of search results measured by the amount of information returned, as well as user perceived precision, recall and F-measure; the efficiency of the search algorithms measured by processing time and scalability over data size.

In the efficiency test, we additionally test the efficiency of WISE in handling synonyms of keywords. Specifically, we use WordNet [8] to find a set of synonyms for each keyword, then take the union of the matches to all synonyms as the matches to this keyword.

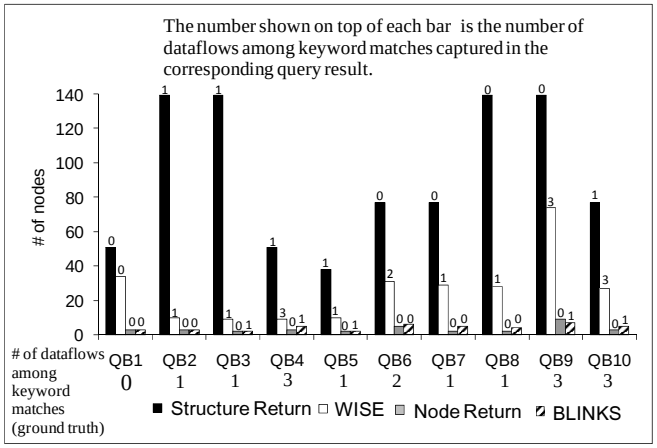


Figure 3: Number of Nodes vs. Number of Dataflows among Keyword Matches in the Query Results

5.1 Experimental Setup

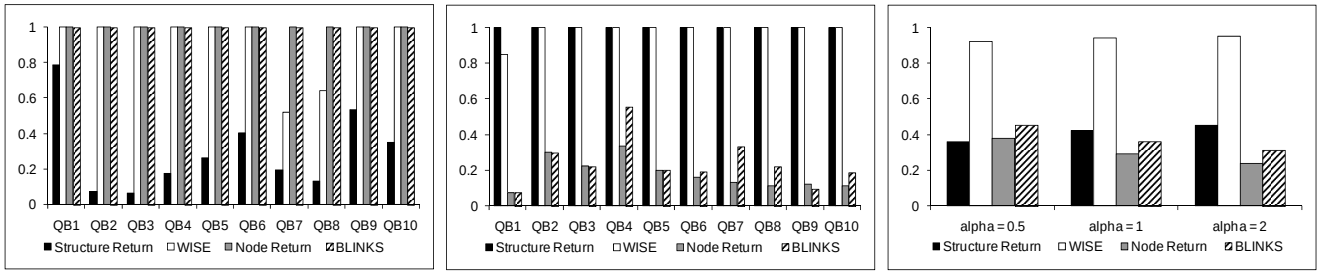
The experiments were performed on a 3.6GHz Pentium 4 machine. The systems were implemented in Java using a commercial database as the backend. All experiments were repeated 5 times independently with cold cache, and we report the average processing time discarding the maximum and minimum values.

Data Set. The data are obtained from the Kepler system[2] in MoML (Modeling Markup Language) format [3], which is the standard file format for specifying workflow hierarchies, widely used in many workflow systems such as Kepler [2], SEEK [5], GEON [1], etc.

Query Set. We have tested thirty queries for workflows in three sample application domains: Biology ($QB_1 - QB_{10}$), Geology ($QG_1 - QG_{10}$), and Ecology ($QE_1 - QE_{10}$). Queries QB_1 to QB_{10} are set by a biologist from the Biology department in Arizona State University. For example, QB_1 intends to find out the usage of *GenBank*, and QB_2 intends to find out how to filter sequences. The queries include single keyword queries, queries whose keywords appear in the same cluster, same layer, or different layers. All queries can be found in the Appendix. .

5.2 Search Quality

Analysis of Information in Query Results. Fig. 3 shows the total number of nodes in the query result, the ground truth of the number of dataflows between keyword matches in the data, and the number of dataflow paths that are captured in the result, of each approach on queries QB_1 to QB_{10}. We can see that these approaches in the decreasing order of the amount of nodes output are: Structure Return, WISE, BLINKS, and Node Return. Each result of each approach contains all keywords of the corresponding query. However, the numbers of dataflow paths among keywords captured by these approaches are quite different, as shown above the bars in Fig. 3. The total numbers of dataflows between all pairs of distinct keywords in the data are considered as the *ground truth*, which are listed below the x-axis. Node Return has zero dataflows returned as no pair of the nodes in a query result are connected. BLINKS is unaware of the difference of dataflow edges and expansion edges and thus often fails to capture the dataflow paths between keywords. Structure Return outputs the entire minimal workflow hierarchies as query results, which only explicitly capture the dataflow paths among keyword matches that are leaf nodes. Even though it outputs much more data nodes than WISE, the amount of relevant information is



Precision Recall F-Measure

Figure 4: Search Quality

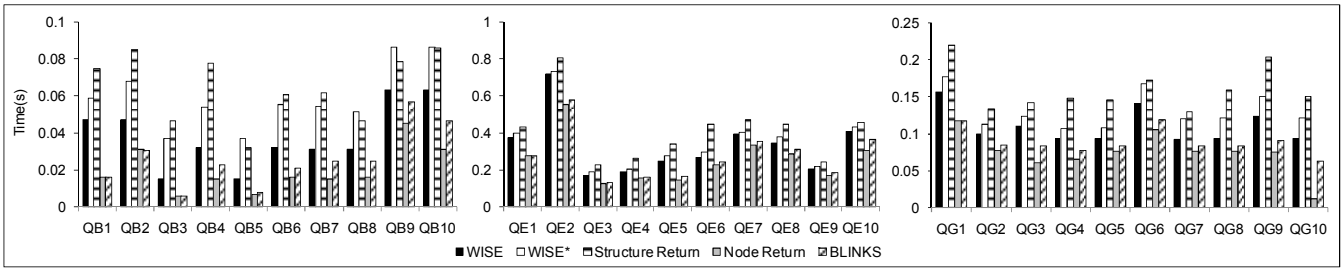


Figure 5: Query Processing Time

limited. On the other hand, WISE can capture all the dataflows by explicitly displaying the dataflow paths connecting keywords in the query results, and thus return *informative* results.

We also evaluate the number of self-contained results generated from all four methods for QB_1 - QB_{10}. All results generated by WISE and Structure Return are self-contained, as they generated minimal views / minimal workflow hierarchies as results. Node Return does not generate any self-contained results. BLINKS generates self-contained results only when it happens to return exactly the nodes and edges in a cluster as a result. For QB_1 - QB_{10}, BLINKS does not generate self-contained results.

User Evaluation. To further verify the rationale of WISE's semantics and the acceptance of WISE's query results by users, we also performed a user study on QB_1 to QB_{10} to measure the *precision*, *recall*, and *F-measure* of these four approaches.

Ten students who are not aware of this project were invited for the survey. For each query, we provided the users with the search results generated by each of the four systems, as well as an option for them to specify their own query results if none of them are satisfactory, by circling all the nodes they wish to be returned (Recall that dataflow information is analyzed in Fig. 6). The ground truth is set based on the majority of agreements by the users. We then calculate the precisions and recalls of each approach on the ten queries based on the ground truth, which are shown in Fig. 4.

As we can see, Structure Return usually has a perfect recall as the entire minimal workflow hierarchies are returned for each query. However, it suffers a low precision as not all the nodes returned are relevant. Take QB_2 as an example, the users are only interested in the information about *Get Sequence* and *Filter*, which comprises only a small portion of the minimal workflow hierarchy. On the other hand, Node Return has a perfect precision on all queries, but suffers a very low recall. Consider QB_4, no information about how to use *Align*, *Blast* and *Get Promoters* together is returned. Similarly, BLINKS has a perfect precision, but low recall, since the results that are returned are generally not self-contained workflows.

WISE has both high precision and recall in general. There are a few queries on which WISE's search quality can be further improved. For query QB_1, WISE has a low recall because WISE outputs the cluster containing *GenBank*, and task *GenBank* is clickable but not expanded. However, the users prefer having the detailed information of *GenBank* directly. The reason WISE has a low precision for QB_8 is that the occurrences of the keywords *Array Merge* and *Align* in the data are far away from the start tasks of the minimal workflow hierarchy. In this case, the users prefer omitting some portion of the data from the start tasks to the keyword matches in the result for conciseness reason. QB_7 has the similar reason as QB_8.

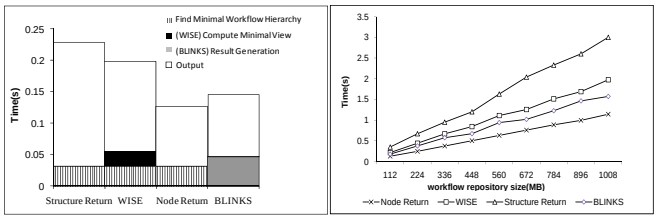
We compute the F-measure of each approach according to the average precision and recall across all the test queries, with parameter α= 0.5, 1 and 2, as presented in Fig. 4. WISE significantly outperforms Structure Return, Node Return and BLINKS.

5.3 Efficiency

Processing Time. The processing times of WISE, Structure Return, Node Return and BLINKS on the test data and query sets are shown in Fig. 5. WISE* denotes the approach that incorporates WordNet for finding synonyms of keywords. All approaches are efficient. Node Return is the fastest, followed by BLINKS and WISE, and Structure Return is the slowest. WISE* takes additional processing time for finding the synonyms of each keyword, as well as finding the matches to them. It can be seen that the additional time WISE* takes to handle synonyms is fractional.

Fig. 6(a) shows the breakdown of the average processing time of each approach over all 30 queries. Structure Return, Node Return and WISE start with finding the minimal workflow hierarchies; WISE further finds the minimal views; BLINKS has specific result generation algorithms; and all consume time for outputting results.

As we can see, the output time is dominant in the overall cost, which is determined by the amount of information output. Fig. 3 shows the number of nodes that are output by each approach for



(a) Processing Time Breakdown

(b) Scalability

Figure 6: Processing Time Breakdown, Scalability and Number of Dataflows Captured

queries on Biology workflow repository. Structure Return, which outputs the whole minimal workflow hierarchies, has the largest output sizes and thus is the slowest. Its output size is followed by that of WISE, and then BLINKS. On the other hand, Node Return, which only outputs individual match nodes, has the smallest number of nodes returned and therefore is fastest.

We also observe that the algorithm of WISE for generating minimal view is very efficient, consuming a very small portion of its processing time. The result generation time of WISE is similar to those of BLINKS and Node Return. The processing overhead of WISE is mainly due to the additional information output, which captures more dataflows between keyword matches (shown in Fig. 3) and results in best search quality among all (shown in Fig. 4).

Scalability. We test the scalability of all four approaches over increasing data sizes by replicating the workflows in the repository multiple times. The processing times of QB_1 are shown in Fig. 6(b). All four approaches increase linearly when the data size increases. They scale similarly on other queries and the figures are omitted.

In summary, WISE achieves significantly better search quality compared with Structure Return, Node Return and BLINKS and returns self-contained, informative yet concise query results. It is efficient and scales well.

6. CONCLUSIONS

We present WISE, a keyword search engine for workflow hierarchies, which are modeled as hierarchies of multiple layers. We identify the minimal views of minimal workflow hierarchies as query results, which are self-contained, informative and concise. Experimental evaluations have shown the effectiveness and efficiency of WISE.

In the future, we will study effective ranking schemes and top-k algorithms such that search results will be output in the order of their relevances. We will also study the generation of query results that are meaningful with respect to the execution of a workflow, e.g., a workflow in which two query keywords only have OR or XOR relationships should not be retrieved as a result for the query, as they can not both be executed during an execution. Another interesting topic to explore is to re-use parts of a workflow hierarchy when constructing a new workflow hierarchy, exploring the trade-off between the efficiency of construction, storage and retrieval, as well as the re-use of node/edge labels and indexes.

7. REFERENCES

[1] GEON. http://www.geongrid.org.

[2] Kepler. http://kepler-project.org/.

[3] MOML. http://ptolemy.eecs.berkeley.edu/papers/05/ptIIdesign1-intro/ptIIdesign1-intro.pdf.

[4] myExperiment. http://www.myexperiment.org/.

[5] Seek. http://seek.ecoinformatics.org.

[6] Taverna Project. http://taverna.sourceforge.net/.

[7] Triana. http://www.trianacode.org/collaborations/index.html.

[8] WordNet: A Lexical Database for English. http://wordnet.princeton.edu/.

[9] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock. Kepler: An Extensible System for Design and Execution of Scientific Workflows. In *SSDBM*, 2004.

[10] Z. Bao, S. C. Boulakia, S. B. Davidson, A. Eyal, and S. Khanna. Differencing Provenance in Scientific Workflows. In *ICDE*, 2009.

[11] Z. Bao, T. W. Ling, B. Chen, and J. Lu. Effective XML Keyword Search with Relevance Oriented Ranking. In *ICDE*, 2009.

[12] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying Business Processes. In *VLDB*, 2006.

[13] O. Biton, S. C. Boulakia, S. B. Davidson, and C. S. Hara. Querying and Managing Provenance through User Views in Scientific Workflows. In *ICDE*, 2008.

[14] O. Biton, S. Cohen-Boulakia, and S. B. Davidson. Zoom*UserViews: Querying Relevant Provenance in Workflow Systems. In *VLDB*, 2007.

[15] A. Chebotko, S. Chang, S. Lu, F. Fotouhi, and P. Yang. Scientific Workflow Provenance Querying with Security Views. In *WAIM*, 2008.

[16] I.-M. A. Chen and V. M. Markowitz. Modeling Scientific Experiments with an Object Data Model. In *ICDE*, 1995.

[17] S. Cohen, S. C. Boulakia, and S. B. Davidson. Towards a Model of Provenance and User Views in Scientific Workflows. In *DILS*, 2006.

[18] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T. H. Jordan, C. Kesselman, P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi, and L. Zhao. Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance tracking: The CyberShake Example. In *e-Science*, 2006.

[19] K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword Proximity Search in Complex Data Graphs. In *SIGMOD*, 2008.

[20] H. He, H. Wang, J. Yang, and P. Yu. BLINKS: Ranked Keyword Searches on Graphs. In *SIGMOD*, 2007.

[21] T. Heinis and G. Alonso. Efficient Lineage Tracking for Scientific Workflows. In *SIGMOD*, 2008.

[22] A. K. Joshi and Y. Schabes. Tree-Adjoining Grammars and Lexicalized Grammars. In *Tree Automata and Languages*, pages 409–432. 1992.

[23] K. Lee, N. W. Paton, R. Sakellariou, and A. A. A. Fernandes. Utility Driven Adaptive Workflow Execution. In *CCGRID*, 2009.

[24] G. Li, V. Muthusamy, H.-A. Jacobsen, and S. Mankovski. Decentralized Execution of Event-Driven ScientificWorkflows. In *SCW*, 2006.

[25] D. T. Liu and M. J. Franklin. The Design of GridDB: A Data-Centric Overlay for the Scientific Grid. In *VLDB*, 2004.

[26] Z. Liu and Y. Chen. Identifying Meaningful Return Information for XML Keyword Search. In *SIGMOD*, 2007.

[27] Z. Liu and Y. Chen. Reasoning and Identifying Relevant Matches for XML Keyword Search. In *VLDB*, 2008.

[28] Z. Liu and Y. Chen. Return Specification Inference and Result Clustering for Keyword Search on XML. *ACM Trans. Database Syst.*, 35(2), 2010.

[29] Z. Liu, Q. Shao, and Y. Chen. WISE: Searching Workflow Hierarchies. Technical report, Arizona State University, 2010.

[30] Y. Luo, X. Lin, W. Wang, and X. Zhou. SPARK: Top-k Keyword Query in Relational Databases. In *SIGMOD*, 2007.

[31] C. B. Medeiros, J. de Jesús Pérez Alcázar, L. A. Digiampietri, G. Z. P. Jr., A. Santanchè, R. da Silva Torres, E. R. M. Madeira, and E. Bacarin. WOODSS and the Web: Annotating and Reusing Scientific Workflows. *SIGMOD Record*, 34(3):18–23, 2005.

[32] M. A. Nieto-Santisteban, J. Gray, A. S. Szalay, J. Annis, A. R. Thakar, and W. O'Mullane. When Database Systems Meet the Grid. In *CIDR*, 2005.

[33] T. M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[34] C. E. Scheidegger, H. T. Vo, D. Koop, J. Freire, and C. T. Silva. Querying and Re-Using Workflows with VisTrails. In *SIGMOD*, 2008.

[35] S. Shankar, A. Kini, D. J. DeWitt, and J. F. Naughton. Integrating Databases and Workflow Systems. *SIGMOD Record*, 34(3):5–11, 2005.

[36] Q. Shao, P. Sun, and Y. Chen. WISE: A Workflow Information Search Engine. In *ICDE*, 2009.

[37] P. Sun, Z. Liu, S. B. Davidson, and Y. Chen. Detecting and Resolving Unsound Workflow Views for Correct Provenance Analysis. In *SIGMOD*, 2009.

[38] M. Vrhovnik, H. Schwarz, S. Radeschütz, and B. Mitschang. An Overview of SQL Support in Workflow Products. In *ICDE*, 2008.

[39] D. L. Wang, C. S. Zender, and S. F. Jenks. Clustered Workflow Execution of Retargeted Data Analysis Scripts. In *CCGRID*, 2008.

[40] Y. Xu and Y. Papakonstantinou. Efficient Keyword Search for Smallest LCAs in XML Databases. In *SIGMOD*, 2005.

APPENDIX

A. Pseudo Code of WISE

Algorithm 1 Keyword Search on Workflow Hierarchies

keywordSearch ($keyword[n]$**, indexes)**
1: **for** i = 1 to n **do**
2: $matches[i] = word2NID(keyword[i])$
3: $roots[r] = findMWHs(matches)$ [40]
4: $matchGroup[r] = grouping(matches, roots)$
5: **for** i = 1 to r **do**
6: $result[i] = genMV(roots[i], matchgroup[i], indexes)$

grouping ($matches[n][p], roots[r]$)
1: $mergedList[m]$ = merge-sort $matches[n][p]$ into a sorted list
2: $i = j = 1$
3: $matchgroup[i] = \emptyset$ for all $1 \leq i \leq r$
4: **while** $i \leq r$ and $j \leq m$ **do**
5: **if** $ancestor - or - self(roots[i], mergedList[j])$ **then**
6: $matchgroup[i] = matchgroup[i] \cup mergedlist[j]$
7: $j++$
8: **else if** $matchgroup[i]! = \emptyset$ **then**
9: $i++$
10: **else**
11: $j++$

genMV ($root, matchgroup[g]$, **indexes)**
1: cn = root; $cm = 1$; $mv = \emptyset$
2: $EdgeHash$ = a hash table initialized as empty
3: **while** $cn \neq$ null and $matchgroup[cm] \neq$ null **do**
4: **if** $cn = matchgroup[cm]$ **then**
5: **if** $ancestor(cn, matchgroup[cm + 1])$ **then**
6: {cn is a match node and has descendant matches}
7: $cm++$
8: continue
9: **else**
10: {cn is a match node and has no descendant match}
11: $outputnode(mv, cn, EdgeHash)$
12: $cn = cn$'s next node in NID order, which is not a descendant of cn
13: **else if** $ancestor(cn, matchgroup[cm])$ **then**
14: {cn is not a keyword match and has descendant matches}
15: $cn = cn$'s first child
16: **else**
17: {cn is not a keyword match and has no descendant match}
18: $outputnode(mv, cn, EdgeHash)$
19: $cn = cn$'s next node in NID order, which is not a descendant of cn
20: return mv

outputnode ($mv, cn, EdgeHash$)
1: add cn into mv
2: **for** each edge entry $eid(incoming/outgoing, d) \in LAL(cn)$ **do**
3: **if** there is an entry (eid, u) in the $EdgeHash$ **then**
4: {u is the other endpoint of eid that has been output}
5: add an edge from u to cn (or from cn to u') into mv with data item d
6: **else**
7: insert an entry (eid, cn) into $EdgeHash$

B. Proof of Theorem 4.1

We adopt the Indexed Lookup Eager Algorithm [40] to compute minimal workflow hierarchies. In the following, we prove that a query result $QR = (V, E)$ generated by Algorithm 1 is the minimal view of a minimal workflow hierarchy that contains all keyword matches which do not have descendant matches.

First, we prove that QR is a view of $H(R, Q)$, i.e., it satisfies the three conditions of view in Section 2.

1. $\forall u \in H$, if $u \in QR$, then none of u's children is output in QR according to Algorithm 1 (recall that after we output a node cn, we move to the next node that is not a descendant-or-self of cn). Therefore, QR satisfies condition 1 in Definition 3.2.
2. $\forall u$, u is a leaf of H and $u \notin QR$, according to Algorithm 1, during the depth-first traversal of H, u must have an ancestor u' which is visited and output in QR (recall that we do not output a node if and only if it is expanded, or an ancestor node is output). In other word, u' is a node and has no descendant matching keywords. Therefore, QR satisfies condition 2 in Definition 3.2.
3. $\forall u, v \in V$, according to Algorithm 1, there is an edge from u to v, (u, v, d), if and only if the ID of the edge (e.g., e_i) is recorded in both $LAL(u)$ and $LAL(v)$. According to the design of leaf adjacency list, this happens if and only if there is an edge (u', v', d) between a descendant u' of u and a descendant v' of v labeled e_i. Therefore, an edge (u, v, d) between u and v is output in the view if and only if there exists an edge (u', v', d) between u' and v'. Thus QR satisfies condition 3 in Definition 3.2.

Next we prove that QR contains all keyword matches in the minimal workflow hierarchy that have no descendant keyword match. For any keyword match node m, each of m's ancestors will be visited and expanded during the traversal of the minimal workflow hierarchy, as it has a descendant match. Therefore, m will be visited and output, as all children of an expanded node will be visited.

At last we prove that QR generated by Algorithm 1 is minimal, i.e., it has the smallest number of nodes among all views of H that contain all keyword matches. Suppose there is another view of H, $QR' = (V', E')$ which has a smaller number of nodes than QR and contains all keywords. Since both views are from the same workflow hierarchy, there must be at least one composite node u' in H which is expanded in QR but not in QR', i.e., $\exists u \in V$ and $u' \in V'$, such that u is a descendant of u'. According to Algorithm 1, u' is expanded in QR only if it has descendant keyword matches. But since u' is output but not expanded in QR', none of its descendants can be output, thus QR' can not contain all the keyword matches of H, which is a contradiction. Therefore, QR is the minimal view of H, and is a qualified query result.

A minimal workflow hierarchy has exactly one minimal view for a query. Thus Algorithm 1 finds exactly the set of query results.

C. Proof of Theorem 4.2

In Algorithm 1, finding the matches using the inverted index takes $O(M)$ time, where M is the number of keyword matches. Procedure $findMWHs$ adopts the Indexed Lookup Eager Algorithm, which takes $O(M_{min}kd\log M_{max})$ time [40]. The $grouping$ procedure traverses the roots of the minimal workflow hierarchies and the keyword match list, whose complexity is bounded by $O(M)$.

$genMV$ scans each minimal workflow hierarchy, but only visits the ancestor-or-self of the nodes in the minimal views. Since each non-leaf node in the minimal workflow hierarchy has at least 2 children, the total number of nodes visited is no more than $2N$. For each edge in the minimal view, $genMV$ has two operations: insert it into a hash table when the first endpoint is output, and retrieve it from the hash table when the second endpoint is output. Thus each edge takes $O(1)$ time to process. Therefore, the time complexity of $genMV$ is $O(N + E)$, which is the best possible complexity as it is equal to the output size. The overall time complexity of the algorithm is $O(M_{min}kd\log M_{max} + M + N + E)$.

D. Queries for Evaluation

Biology	
QB_1	GenBank
QB_2	Get Sequence, Filter
QB_3	Get Promoters, Align
QB_4	Align, Blast, Get Promoters
QB_5	Filter, Synchronizer
QB_6	Record Updater, Filter
QB_7	Blast, Get Sequence, Merge
QB_8	Array Merge, Align
QB_9	Record Updater, Record Disassembler
QB_{10}	Align, Filter, Synchronizer
Geography	
QG_1	SVG Concatenate
QG_2	ExtractURL, ExtractShpURL
QG_3	WebService, RecoradDisassembler
QG_4	ClassifySample, extractAge
QG_5	ClassifySample, AssertPoint
QG_6	ClassifyBody, SVG To Polygon Converter
QG_7	Composite Actor, SVG Concatenate
QG_8	Add Point To SVG, QueryBodyAge
QG_9	Get 2D Point, Record Disassembler, Render Mapler
QG_{10}	Record Disassembler, SVG To Polygon Converter
Ecology	
QE_1	Add Grids
QE_2	GarpPrediction, GarpAlgorithm
QE_3	Future-Climate-Model, IJMacro
QE_4	LocationFile, I - DataPoints
QE_5	Create ASC Maps , Garp Prediction
QE_6	Calculate Best Rulesets , CV Hull to RasterMask
QE_7	IJMacro, II - EnvLayerSet
QE_8	Add Grids , GarpAlgorithm
QE_9	Future-Climate-Model,ConvexHull, GarpPrediction
QE_{10}	Create ASC Maps, Add Grids, I - DataPoints

E. Related Work

Keyword Search on Graphs/Trees. Keyword search on relational data/graph models [20, 19, 30] and on XML data/tree models [11, 26, 28] has been well studied. However, as explained in Section 1, these techniques are not applicable for searching scientific workflows for several reasons. First, a workflow hierarchy is a three dimensional structure consisting of nested graphs, and thus is more expressive than graphs/trees. Second, there are two types of edges in a workflow hierarchy, dataflow edges and expansion edges, where the latter carry different semantics as the edges in graphs/trees. Third, while the keyword search results on graphs/trees are substructures extracted from the original data, search results on workflow hierarchies often should contain synthesized dataflow edges that are not explicitly present in the original data, in order to explicitly capture the dataflows among nodes matching keywords in the view. Furthermore, query results of workflow hierarchies should be self-contained with respect to its composite tasks and expansion edges, which is a unique requirement compared with keyword search on graphs/trees. We propose novel techniques and algorithms in WISE to address the unique challenges of the problem of keyword search on workflow hierarchies.

Workflow Models. WOODSS [31] models scientific workflow designs with multiple abstraction layers in order to facilitate workflow composition and reuse. [17] proposes a formal and general model of data provenance for scientific workflows having multiple abstraction layers. [13] provides a method for automatically constructing views from the bottom layer of a scientific workflow according to user specified important tasks. All of them focus on how to model and construct the hierarchial structures of workflows. On the other hand, the focus of this paper is effectively retrieving information from hierarchical workflows and presenting it to the user according to specified keywords. There is also research on how to effectively execute a specific type of workflow in various circumstances [39, 24, 18, 23]. These are orthogonal problems of defining query results on workflow hierarchies, as individual execution methods of a workflow hierarchy do not affect the specification of views on the workflow hierarchy, hence do not affect the definition of query results and the algorithms to generate query results.

Querying Workflows. Studies have been performed on querying scientific workflows. One option is to map workflows into relational/object-oriented databases, or XML, and express the search using standard database query languages [16]. Recent efforts propose an integration of scientific workflows and relational databases [32, 25, 35], which associate each task in a workflow with a relational table recording the input and output data. However, it can be difficult for scientific users to search workflows using database query languages since they need to learn the query languages, complex data schemas as well as complicated mappings between schemas and workflows. A recent work [12] proposes BPEL, a visualized query language for business processes, whose interface is similar to the one for workflow construction.

Kepler [2], Triana [7] and Taverna [6] allow users to search workflow tasks using keywords or regular expressions. Individual tasks that match the input keywords/expressions will be returned, without the dataflow information among the tasks. On the other hand, myExperiment [4] returns the entire workflow hierarchies containing the keywords, delivers an overwhelming volume of information.

WISE is the first keyword search engine on workflows hierarchies, which is desirable for users not familiar with database query languages, such as scientific users. WISE returns minimal views (concise) of workflow hierarchies that have unique goals/names (self-contained) and preserves/synthesizes the dataflows among nodes matching keywords (informative), so that users can easily understand the relationships of the query keywords in the results. Furthermore, the results generated by WISE satisfy desirable properties proposed in the literature, including soundness, monotonicity and consistency.