Effective and Efficient Similarity Search in Scientific Workflow Repositories

Johannes Starlinger, Sarah Cohen-Boulakia, Sanjeev Khanna, Susan Davidson, Ulf Leser

Effective and Efficient Similarity Search in Scientific Workflow Repositories

Johannes Starlinger[a,*], Sarah Cohen-Boulakia[b], Sanjeev Khanna[c],
Susan B. Davidson[c], Ulf Leser[a]

[a] *Humboldt-Universität zu Berlin, Institut für Informatik,*
Unter den Linden 6, 10099 Berlin, Germany
[b] *Université Paris-Sud, Laboratoire de Recherche en Informatique,*
CNRS UMR 8623, INRIA, LIRMM, France
[c] *University of Pennsylvania, Department of Computer and Information Science,*
3330 Walnut Street, Philadelphia, PA 19104-6389, USA

Abstract

Scientific workflows have become a valuable tool for large-scale data processing and analysis. This has led to the creation of specialized online repositories to facilitate workflow sharing and reuse. Over time, these repositories have grown to sizes that call for advanced methods to support workflow discovery, in particular for similarity search. Effective similarity search requires both high quality algorithms for the comparison of scientific workflows and efficient strategies for indexing, searching, and ranking of search results. Yet, the graph structure of scientific workflows poses severe challenges to each of these steps. Here, we present a complete system for effective and efficient similarity search in scientific workflow repositories, based on the Layer Decompositon approach to scientific workflow comparison. Layer Decompositon specifically accounts for the directed dataflow underlying scientific workflows and, compared to other state-of-the-art methods, delivers best results for similarity search at comparably low runtimes. Stacking Layer Decomposition with even faster, structure-agnostic approaches allows us to use proven, off-the-shelf tools for workflow indexing to further reduce runtimes and scale similarity search to sizes of current repositories.

Keywords: scientific workflows, similarity search

1. Introduction

Scientific workflow systems have become an established tool for creating and running reproducible in-silico experiments. With their increasing popularity, online repositories of scientific workflows have emerged as a means of

[*] Corresponding author

Email addresses: starling@informatik.hu-berlin.de (Johannes Starlinger), cohen@lri.fr (Sarah Cohen-Boulakia), sanjeev@cis.penn.edu (Sanjeev Khanna), susan@cis.penn.edu (Susan B. Davidson), leser@informatik.hu-berlin.de (Ulf Leser)

facilitating sharing, reuse, and repurposing. Examples of such repositories include CrowdLabs [18], SHIWA [2], and the Galaxy repository [14]. Probably the largest workflow collection is myExperiment [19], which currently contains more than 2500 workflows from various disciplines, including bioinformatics, astrophysics, and earth sciences. To make the best use of these repositories, users need support to find workflows that match their specific needs [5]. However, currently these repositories only support keyword queries which are matched against textual descriptions, tags, and titles given to the workflows upon upload [2, 14, 18, 19]. Obviously, the quality of such a search critically depends on the quality of the annotations associated with workflows.

Another source of information that can be exploited for search is the definition of a workflow itself [13]: Scientific workflows typically resemble directed acyclic graphs (DAGs) consisting of global input and output ports, data processing modules, and datalinks which define the flow of data from one module to the next. Each module has a set of attributes associated with it, such as a descriptive label, the type of operation to be carried out, or, for instance, the uri of a web-service to be invoked. Two sample workflows from myExperiment are shown in Figure 1. This structure or topology of the workflow, together with the attributes defined on the workflow's modules, is used by structure-based methods of workflow comparison. An obvious advantage of structure-based approaches to workflow similarity search is that they do not require any additional information to be provided by the workflow designer apart from the workflow itself. Structure-based approaches are typically used in a second search phase: First, users identify workflows which roughly match their needs using keyword search. In the second phase, users select one candidate workflow and let the system retrieve functionally similar workflows, i.e., the system performs a workflow similarity search.

Several studies have investigated different techniques for assessing workflow similarity using this attribute-enriched structure [3, 11, 13, 20, 21, 25, 27], but initial results indicate that they perform no better, and sometimes even worse, than annotation-based methods in terms of retrieval quality [6, 11, 25]. However, these comparisons were performed on very small and well-documented workflow sets, and thus the results should not be extrapolated to the large, but shallowly annotated repositories that exist today. To verify this hypothesis, in prior work we performed a large-scale comparative evaluation of workflow similarity search algorithms [22]. Our results indicated that a) structure-based methods are indispensable for some current repositories which lack rich annotations, b) structure-based methods, once properly configured, outperform annotation-based methods even when such rich annotations are available, and c) any such standalone approach is further beaten by ensembles of annotation-based and structure-based methods. We also discovered that both the amount of configuration required and runtime considerations were drawbacks to such methods: Fast workflow comparison using annotations on the workflows' modules provides best results only when ubiquitous, functionally unspecific modules are removed from the workflows in a preprocessing step. The configuration of which modules are to be removed is specific to a given dataset, and is non-trivial.

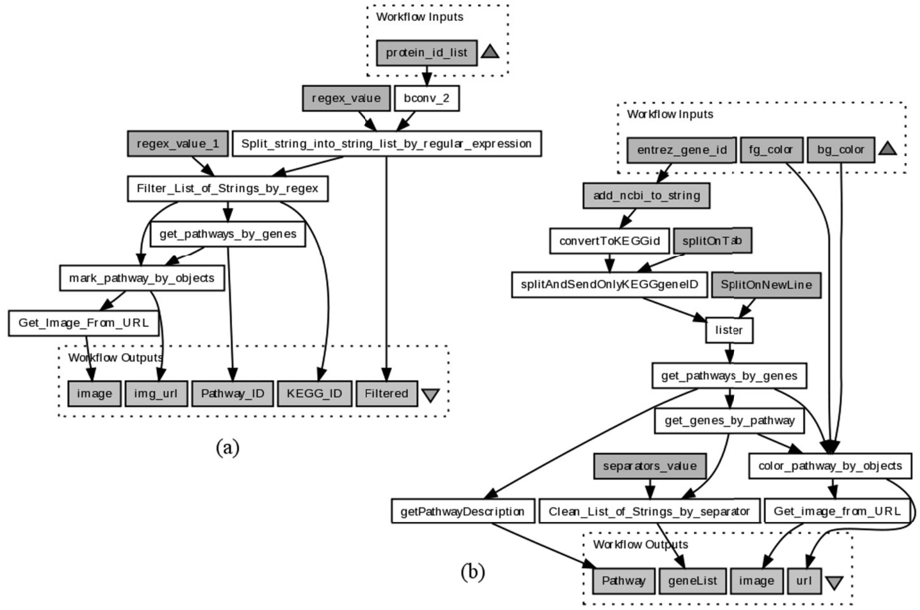


Figure 1: Sample scientific workflows from the myExperiment repository: (a) ID: 1189, Title: *KEGG pathway analysis*, (b) ID: 2805, Title: *Get Pathway-Genes by Entrez gene id*.

Methods based on workflow substructures, on the other hand, provide rather stable results across different configurations, but have prohibitive runtimes.

Based on these findings we presented a novel technique for measuring workflow similarity that accounts for the directed dataflow underlying scientific workflows [23]. The central idea is the derivation of a *Layer Decomposition* for each workflow, which is a compact, ordered representation of its modules, suitable for effective and efficient workflow comparison. Comparatively evaluating this novel technique against previous approaches, we showed that the algorithm a) delivers the best results in terms of retrieval quality when used stand-alone, b) is essentially configuration free which makes it applicable to any workflow repository, regardless of how well its workflows are annotated, c) is faster than other algorithms that account for the workflows' structure, and d) can be stacked and combined with other measures to yield better retrieval at even higher speed.

Extending on these encouraging results we here investigate their transferability into a system for fast similarity search for scientific workflows at repository-scale. While runtime has been a concern in developing the *Layer Decomposition* approach, scaling its quality of scientific workflow comparison to large collections of workflows requires additional considerations as of how to best index workflows for fast retrieval. Especially our previous findings regarding the stackability of Layer Decomposition with other algorithms of even lower structural complexity gives rise to a two-phase architectural design of combining a fast candidate re-

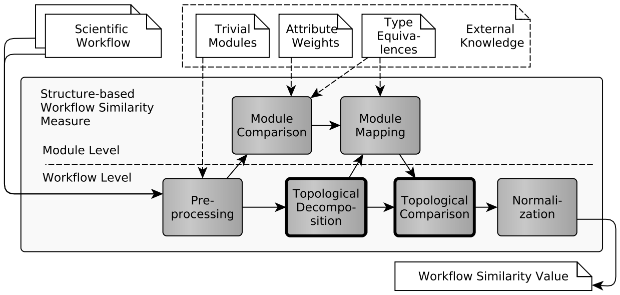


Figure 2: Workflow comparison process, see also [22]

trieval strategy that disregards workflow structure with our best-of-breed Layer Decomposition approach for result ranking: In this paper, we show how our previous findings can be leveraged to a) efficiently index scientific workflows for fast similarity search using off-the-shelf technology, b) improve retrieval precision within the top-ranked results by reranking the initial structure-agnostic search results with the structure-aware Layer Decomposition algorithm, and c) further speed up the overall retrieval process by tweaking specific subtasks of the reranking algorithm.

The remainder of this paper is structured as follows. We first briefly introduce the process of scientific workflow comparison in Section 2 and review the Layer Decomposition algorithm in Section 3. Section 4 describes our similarity search system architecture. In Section 6 we review related work regarding similarity search in scientific workflow repositories. We evaluate our system in Section 5. Section 7 concludes the paper.

2. Scientific Workflow Comparison

In [22], we report on a comprehensive evaluation of previous approaches to workflow similarity search. In addition to comparing retrieval quality quantitatively, we also introduce a framework (shown in Figure 2) for qualitatively comparing different systems. This is an important tool, as the process of workflow comparison entails many steps: First, the similarity of each pair of modules from two workflows is determined using pairwise *module comparison*. Second, using these pairwise module similarities, a *mapping of modules* onto each other is established. This mapping may be influenced by the *topological decomposition* of the workflows imposed by the third step of *topological comparison*, which in turn uses the established mapping to assess the similarity of the two workflows. Finally, *normalization* of the derived similarity value wrt the sizes of the compared workflows is done. This process of scientific workflow comparison is preceded by an (optional) preprocessing step to, for instance, include externally supplied knowledge about the elements the workflow contains (see Section 2.4).

Each of these steps has a notable impact on the concrete similarity values computed. We describe the necessary details in the following. Note that our novel Layer Decomposition method (described in the next section) is a contribution dedicated to the step of *topological decomposition and comparison*.

2.1. Module Comparison

Structure-based approaches to workflow similarity must first measure the similarity of two modules in the to-be-compared workflows using the modules' available attributes. These range from identifiers for the *type* of operation to be carried out, to the descriptive *label* of the module given to it by the workflow's author, to rather specific attributes such as the *url of a web-service* to be invoked. In [22] we compared several combinations of attributes with different weightings and showed that choosing a suitable configuration is most crucial for result quality of structural workflow comparison. Using all of a module's attributes for comparison only yields satisfying results when the weights used on the attributes are set appropriately. Conversely, using only the module's labels for comparison provides equally good results when these are compared by Levenshtein edit distance to cope with minor differences in different authors' naming schemes. While this may seem surprising at first, it does show that the labels provided to the modules by the (heterogeneous) authors of the workflows contained in a repository are often telling of their functionality.

2.2. Module Mapping

Based on the pairwise module similarities determined by module comparison, the *module mapping* defines the optimal set of allowed node associations for further topological comparison. Based on our previous findings [22], we here use two strategies: *maximum weight matching (mw)* chooses the set of one-to-one mappings that maximizes the sum of similarity values for unorderes sets of modules, whereas *maximum weight non-crossing matching (mwnc)* [17] requires an order on each of the two sets to to be given by the workflows' topological decompositions. Given two ordered lists of modules $(m_1, ..m_i, ..m_k)$ and $(m'_1, ..m'_j, ..m'_l)$, a mapping of maximum weight between the sets is computed where no pair of mappings (m_i, m'_j) and (m_{i+x}, m'_{j-y}) may exist with $x, y \geq 1$.

2.3. Normalization

A difficult problem in topological comparison of workflows is how to deal with differences in workflow size. In particular, given a pair V, W of workflows where $V \subset W$ and $|W| >> |V|$, what is their similarity? This decision typically is encoded in a normalization of similarity values wrt the sizes of the two workflows [22]. In this work, we use a variation of the Jaccard similarity coefficient, which measures the similarity of two sets A and B by their relative overlap: $\frac{|A \cap B|}{|A|+|B|-|A \cap B|}$. We modify this formula because the methods for comparing modules do not create binary decisions but instead return a similarity score; details can be found in [22].

2.4. External Knowledge

When comparing two workflows, knowledge derived from the entire workflow repository or even from external sources may be taken into account. Our results in [22] showed that the following two simple and efficiently computable options are already quite effective: *Importance Projection Preprocessing* removes those modules from a workflow prior to its comparison, which provide unspecific functionality such as parameter settings or simple format conversions [24]. The connectivity of the workflow graph is retained by transitive reduction of removed paths between the remaining modules. This method requires external knowledge in the form of a method to assess the contribution of a given module to the workflow's function. The implementation provided here uses manual assignments of importance based on the module's type of operation.

Module Pair Preselection, first classifies modules by their type and then compares modules within the same class, instead of computing all pairwise module similarities for two workflows. This reduces the number of (costly) module comparisons and may even improve mapping quality due to the removal of false mappings across types. Here, external knowledge must assign a predefined class to each module.

3. Layer Decomposition Workflow Similarity

In this section, we describe the *Layer Decomposition* (LD) approach for structurally comparing two workflows, initially presented in [23]. In a comparative evaluation against a number of other approaches to topological workflow comparison under various settings for the different steps of the workflow comparison process described in the previous section, this algorithm provides results of highest quality in similarity search, and of highest stability across different such settings. The fundamental idea behind LD is to focus on the dependency-constrained order in which modules are executed in both workflows by only permitting mappings of modules to be used for similarity assessment which respect this order (in a sense to be explained below). Two observations led us to consider execution order as a fundamental ingredient to workflow similarity. Firstly, it is intuitive: The function of a workflow obviously critically depends on the execution order of its tasks as determined by the direction of data links; even two workflows consisting of exactly the same modules might compute very different things if these modules are executed in a different order. Nevertheless, most structural comparison methods downplay execution order. For instance, a few graph edits can already lead to workflows with very different execution orders (like swapping the first and last of a long sequence of modules). Secondly, we observed in our previous evaluation [22] that approaches to topological workflow comparison which consider execution order are much more stable across different configurations of the remaining steps of the workflow comparison process. In particular, comparing two graphs using their path sets, i.e., the set of all paths from a source to a sink, produced remarkably stable results both with and without the use of external knowledge. Inclusion of such knowledge

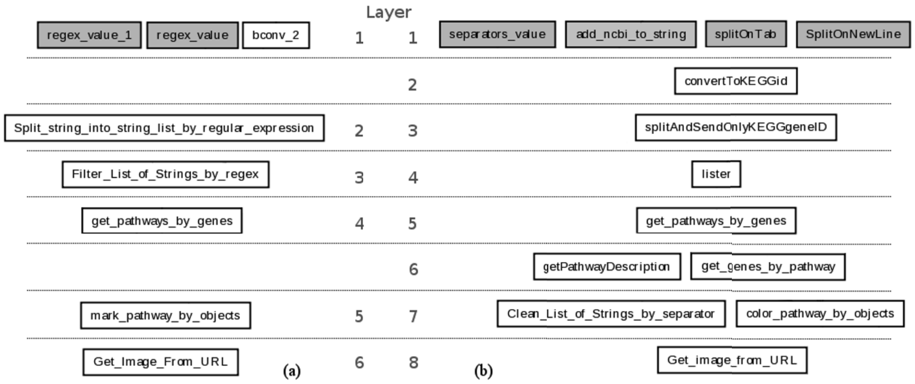


Figure 3: Sample layer decompositions and layer mapping of scientific workflows (a) 1189 and (b) 2805 from Figure 1.

in workflow comparison had among the largest impact on the overall performance of methods, but requires corpus-specific expert intervention. Based on these findings, developing methods that achieve retrieval results of high quality without requiring external knowledge seemed like a promising next step.

In the following, we first explain how LD extracts an ordering of workflow modules from the workflow DAG. We then show in Section 3.2 how two workflows can be effectively compared using this partial ordering. Finally, we explain in Section 3.3 how normalization is performed.

3.1. Topological Decomposition

The *linearization* (or *topological sort*) of a DAG is an ordering of its nodes V such that node u precedes node v in the ordering, if an edge (u, v) exists. Obviously, a DAGs linearization can be computed in linear time using topological sorting; however, it is generally not unique. As the quality of the subsequent mapping (see below) depends on the concrete linearizations chosen for the two workflows under consideration, it is important to find a good pair of linearizations, i.e., linearizations such that highly similar modules will later get mapped onto each other. Since the number of possible linearizations is $\Omega(n!)$ (where n is the number of modules in a workflow), assessing all possible pairs is theoretically infeasible; it is also infeasible in practice, as many real life workflows have many different linearizations (for instance, 23.5% of the 1485 Taverna workflows in our evaluation set have more than 100 different linearizations).

We tackle this problem by representing all possible linearizations of a given workflow in a single, concise data structure. Observe that a DAG has more than one linearization iff between two consecutive nodes in one of its linearizations no direct datalink exists, because in this case swapping the two nodes creates another linearization. In all such cases, we tie the two nodes in question into a single position in the ordering. We call such a tie at position i a *layer* L_i. Compacting all sequences of two or more swappable nodes of a linearization in this

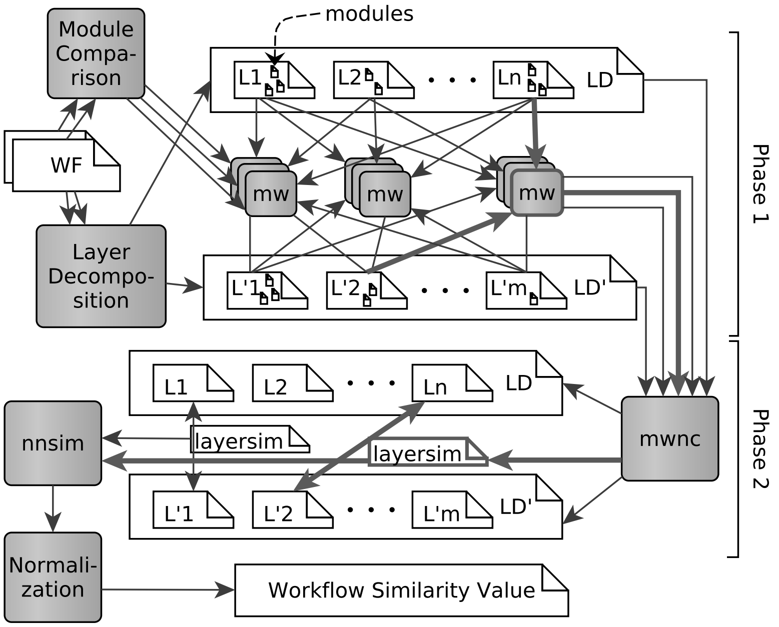


Figure 4: Overview of workflow comparison applied by LD. See text for description of overall process. Bold blue lines indicate pairwise comparison of layers Ln and L'2 by maximum weight matching (mw). The resulting similarity value is passed to maximum weight non-crossing matching (mwnc) to determine best layer mapping and subsequently used to compute non-normalized overall similarity score (nnsim).

way yields a layered ordering of the DAG which we call its *layer decomposition* $LD = (L_1, .., L_i, .., L_k)$. Note that the layer decomposition of a DAG is unique, as a) layers themselves are orderless sets of modules, and b) following from the definition of the underlying *linearization*, for any L_i and L_j such that $i < j$, for every $v \in L_j$, there is some $u \in L_i$ which precedes v in every linearization, which c) uniquely defines the positions of L_i and L_j within the decomposition. To compute a workflow's layer decomposition, we use a simple iterative algorithm. First, all modules with in-degree 0 (the DAGs source nodes) form the top layer L_1. These modules and all their outbound data links are removed from the workflow; this process is repeated until no more modules remain. Figure 3 shows the layer decompositions of the sample workflows introduced in Figure 1. In Figure 3, the different layers are visually aligned to reflect their mapping as derived in the following step.

3.2. Topological Comparison

The layer decomposition of a workflow partitions its module set by execution order creating an ordered list of module subsets. To compare the layer decompositions LD and LD' of two workflows wf and wf', respectively, we take a two-phase approach, sketched in Figure 4. First, pairwise similarity scores for each pair of layers $(L, L') \in LD \times LD'$ are computed from the modules they contain using the maximum weight matching (mw), based on the similarity values $p(m, m')$ derived by a given module comparison scheme as introduced in Section 2.2:

$$layersim(L, L') = \sum p(m, m') \mid (m, m') \in mw(L, L')$$

In the second phase, the ordering of the layers - and thus of the modules they are comprised of - is exploited to compute the decompositions' maximum weight non-crossing matching ($mwnc$) with the pairwise similarities of layers from phase one. The resulting layer-mapping serves as the basis for the overall (yet non-normalized) similarity score of the compared workflows using LD:

$$nnsim_{LD}(wf, wf') =$$

$$\sum layersim(L, L') \mid (L, L') \in mwnc(LD, LD')$$

3.3. Normalization

As done for all other methods we shall compare to, we normalize the similarity values computed by LD using the Jaccard variation described in Section 2.3. Thus, the final, normalized LD-similarity is computed as:

$$sim_{LD}(wf, wf') = \frac{nnsim_{LD}}{|LD| + |LD'| - nnsim_{LD}}.$$

We analogously normalize $layersim(L, L')$ by $|L|$ and $|L'|$. This way, if two workflows are identical, each layer has a mapping with a similarity value of 1. Then $nnsim_{LD} = |mwnc(LD, LD')| = |LD| = |LD'|$, and $sim_{LD} = 1$.

4. Similarity Search Architecture

Thorough evaluation [23] of the Layer Decomposition showed that it provides best results in the task of similarity search, and that it is considerably faster at comparing workflows than its structure-aware competitors. Yet, for similarity search at repository scale, its average runtime is still prohibitive due to the complexity of the involved graph algorithms. An appropriate method for achieving faster retrieval is indexing of the repository. Such indexing of workflows is straightforward when considering only their modules, but requires more sophisticated methods when also topology should be indexed. Encouragingly, our evaluation showed that stacking the Layer Decomposition (LD) algorithm as a reranking step on top of initial candidate preselection by a structure-agnostic similarity measure (that only compares the workflows' modules), provides equally good results as the standalone application of LD.

Here we present and evaluate a two-phased approach consisting of an initial, structure-agnostic retrieval step using indexing, and a subsequent step of result

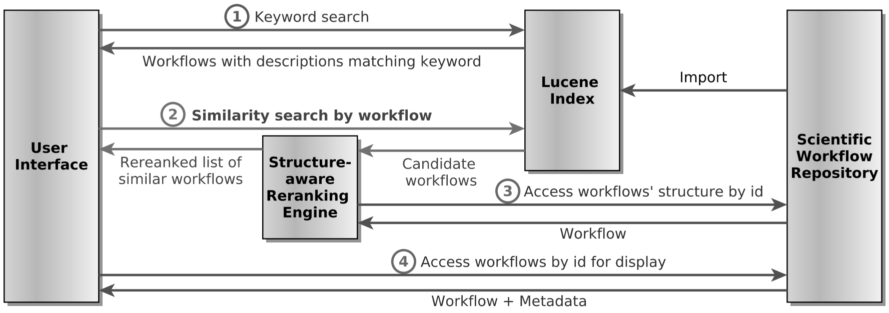

1 Keyword search
Workflows with descriptions matching keyword
Lucene Index
Import
2 Similarity search by workflow
User Interface
Scientific Workflow Repository
Rereanked list of similar workflows
Structure-aware Reranking Engine
Candidate workflows
3 Access workflows' structure by id
Workflow
4 Access workflows by id for display
Workflow + Metadata

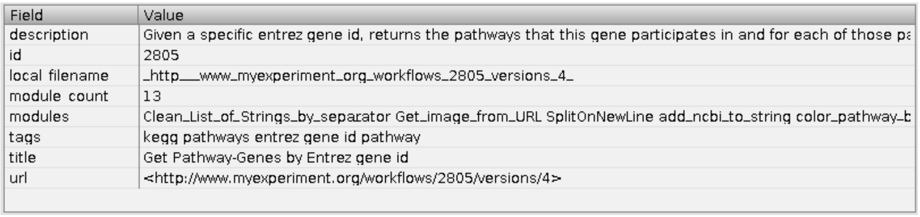
Figure 5: Schematic overview of scientific workflow similarity search using structure-based reranking.

reranking using more complex structure-based workflow comparison. When implementing this method, we tried to use off-the-shelf components as much as possible to reduce maintenance cost in concrete installations - and facilitate applicability to such installations in the first place. In particular, we were eager to build on existing indexing technology currently employed by scientific workflow repositories to provide their search capabilities (based on workflow annotations). A popular such indexing tool is Lucene [15], which is, for instance, used by the myExperiment repository [12]. Directly integrating with this technology reduces maintenance cost to only one (proven) index and provides an inherent benefit to our overall purpose of serving similarity search over scientific workflow repositories: Such a search often starts with (1) an initial keyword query that roughly specifies the user's needs in terms of workflow functionality. Users then select one candidate workflow and (2) let the system retrieve functionally similar workflows, i.e., the system performs a workflow similarity search.

An overview of our system as a whole, reflecting these user-driven steps of the retrieval process, is given in Figure 5. In the following, we describe the index, and the phases of candidate retrieval and reranking (employed by the similarity search of step (2)) in detail.

4.1. Workflow Indexing and Retrieval

The first step is to index workflow properties and modules for fast initial retrieval. As we have shown that module-based retrieval provides a very good preselection of search results, it is a natural target for indexing. Thus, we hypothesize that it is not necessary to apply more complex graph indexing databases and datastructures such as neo4j [1], or the subgraph indexing system used in [13], but sufficient to use fast and simple approaches that capture the necessary details of the workflows we want to index: their modules. A search over such an index would specify a number of modules the query workflow contains, which would then be matched onto the sets of modules stored with each workflow in the index. The fundamental operation required from such an index is the ability to compare modules. Comparison can be effectively done



Field	Value
description	Given a specific entrez gene id, returns the pathways that this gene participates in and for each of those pa
id	2805
local filename	_http___www_myexperiment_org_workflows_2805_versions_4_
module count	13
modules	Clean_List_of_Strings_by_separator Get_image_from_URL SplitOnNewLine add_ncbi_to_string color_pathway_b
tags	kegg pathways entrez gene id pathway
title	Get Pathway-Genes by Entrez gene id
url	<http://www.myexperiment.org/workflows/2805/versions/4>

Figure 6: Document representation of myExperiment workflow 2805 (see Fig. 1) in Lucene. The tabular view of fields and values is taken from the index analysis tool Luke[1].

based on the edit-distance of the modules' labels [22]. Both operations are supported by the open source system Apache Lucene [15].

4.1.1. Indexing Workflows in Lucene

Lucene is a Java-based document indexing and search engine that includes options for analyzing the documents prior to indexing (e.g., stop word removal), and for ranking of search results by their (document-centric) relevance to the respective query [15]. In Lucene, *documents* fed to the index are composed of *fields*, where each field contains a sequence of *terms*. For full text documents, the words in the text naturally represent these terms; the text as a whole makes up one field; and the document as a whole may, next to the field representing its full text body, contain additional fields to hold, for instance, its title or the date of publication. From the terms stored, Lucene creates an inverted index linking each term to the documents and fields it is contained in. For searching, a given query (i.e., one or more search terms) is matched in the index to find the corresponding documents. Lucene then ranks the matching documents by their relevance to the query using various relevance metrics. Next to strict matching of terms in the index, *fuzzy* searching is also supported. When performing a fuzzy search, Lucene uses the edit distance between terms in the query and terms in the index to find matching documents.

Following Lucene's document structure, we represent a workflow as a set of fields. One field holds the set of modules the workflow contains, with the modules' labels being the respective terms. This approach is the most straightforward setting for both indexing and search and, as we will see, provides surprisingly satisfying results (see Section 5.1). We discuss other options for representing workflows in Lucene in Section 7.

Figure 6 shows how a workflow is stored in the index using these fields. We use fields for the workflow's modules, the workflow's id and url in the original repository (here myExperiment), the filename of the workflow definition file in our system, and the number of modules the workflow contains. For the practical reasons of integrating with exising repositories' search capabilities outlined

[1] http://code.google.com/p/luke/

Listing 1: Example fuzzy query to the Lucene index constructed from the modules of myExperiment workflow 1189 (see also Fig. 1).

```
(modules:Filter_list_of_strings_by_regex~0.7)
(modules:Get_image_from_url~0.7)
(modules:Split_string_into_string_list_by_regular_expression~0.7)
(modules:bconv_2~0.7)
(modules:get_pathways_by_genes~0.7)
(modules:mark_pathway_by_objects~0.7)
(modules:regex_value~0.7)
(modules:regex_value_1~0.7)
```

above, we also index the workflow's title, description, and associated tags - in which a user's initial keyword queries would be (additionally) matched. Creating the index for our repository of 1483 Taverna workflows takes approximately 4 minutes on a desktop machine. The index uses 1.7MB on disk.

4.1.2. Searching the Index

Lucene provides its own query syntax for searching the index. Most important to us is the ability to specify over which fields a search is to be carried out, and what the maximum edit distance is at which Lucene will consider terms to match. In Lucene's terminology this distance is given by the minimum similarity at which its *fuzzy* query processing may consider a pair of terms to match. It takes values between 0 and 1, and corresponds to the number of allowed edit operations wrt terms' length. In the following, we refer to this minimum similarity as *fuzzyness.*

To perform a workflow similarity search by modules, we construct a query from the labels of the modules of a given query workflow. An example query is shown in Listing 1. For each module the *modules* field is specified to be searched in, and the desired fuzzyness is appended to the module's label (separated by a ~). This example query for workflows with similar sets of modules to workflow 1189 from Figure 1 returns a total of 289 results.

4.2. Structure-based Result Reranking

After fast structure-agnostic retrieval of candidate results, we rerank these results by the structure-aware Layer Decomposition algorithm. In our previous evaluation [23], this algorithm has not only shown to provide best results in the task of workflow ranking, but to also be comparatively fast.

4.2.1. Layer Decomposition Configurations

In our previous evalutation of the Layer Decomposition algorithm we were especially interested in the performance those configurations of the workflow comparison process (see Section 2) that require as little background knowledge about the repository as possible. When creating a search engine for one specific repository of scientific workflows, on the other hand, extensive background information and tuning will be used to deliver a better user experience. To

Table 1: Configurations of the Layer Decomposition algorithm used for reranking, and the impact the corresponding tweaks have on algorithm quality (Q) and speed (S). (+: improved, o: unchanged)

Comparison step	LD_np_ta_pll	LDpml25_ip_te_pll	Benefit	
			Q	S
Workflow Preprocessing	np: no projection	ip: importance projection filtering out unspecific modules.	+	+
Module Comparison	ta: no preselection of pairs for comparison, all module pairs are compared. pll: only the *labels* are compared by edit distance	te: preselection of modules pairs for detailed comparison by *type equivalence*. pll: only the *labels* are compared by edit distance	o	+
Module Mapping	maximum weight matching	maximum weight matching		
Topological Comparison	LD: Layer Decomposition	LDpml25: Layer Decomposition, penalizing layer mismatch exceeding 25% of the layers in the larger workflow	+	o
Normalization	Jaccard variation	Jaccard variation		

contrast these perspectives of off-the-shelf applicability and customizability, we will evaluate the Layer Decomposition approach in two settings:

- *LD_np_ta_pll* does not use any knowledge of the repository and its workflows, apart from the assumption that the modules' labels are telling of their functionality - a reasonable assumption in our setting given the way our index is constructed.

- *LDpml25_ip_te_pll* uses the maximum amount of knowledge we have, including *importance projection (ip)* to filter out unspecific modules, *type equivalence (te)* module pair preselection to reduce the number of detailed module comparisons made (see Section 2), and *penalties for mismatching layers (pml25)* in the Layer Decomposition topological comparison itself. These penalties measure the amount of layers not matched in LD's maximum weight non crossing matching step, when the number of such mismatched layers exceeds a configurable, repository-dependent percentage of the layers in the larger of the compared workflows (i.e., the maximum number of layers that could possibly be matched). Setting this allowance to 25% notably improves retrieval performance of Layer Decomposition [23].

Table 1 lists each step of the workflow comparison process and how it is treated in these two configurations (and decodes the intricate notation). It also shows how tuning at each step affects quality and speed of the algorithms [22, 23]. We will see how this translates to concrete runtimes of workflow comparison subtasks in Section 5.2.2.

4.2.2. Ensembles of Structure and Annotation

Next to measuring workflow similarity using single algorithms, multiple algorithms can be combined into *ensembles*. These ensembles allow to integrate different perspectives on workflow similarity, especially those provided by structure-based and annotation-based comparison. How applicable such ensembles are to any given repository depends on the amount of annotations it provides. Current repositories greatly differ in this respect, and lack of textual descriptions of workflows greatly affects the applicability of annotation-based measures. When annotations are available, on the other hand, we have shown the use of ensembles to greatly benefit result quality in similarity search [22, 23]. We thus also evaluate the ensemble of *Bag of Words* [6, 22] similarity over the workflows' titles and descriptions and Layer Decomposition in either of the configurations listed above. Ensemble similarity values are derived as mean average of the values computed by its constituting algorithms.

5. Evaluation

We evaluate our system for scientific workflow similarity search on the corpus of scientific workflows introduced in [22]. The corpus consists of 1483 Taverna workflows from the myExperiment repository. For 485 pairs of scientific workflows from this corpus, a total of 2424 similarity ratings were provided by 15 human experts[2]. Ratings were given along a four step Likert scale [16] with the options *very similar*, *similar*, *related*, and *dissimilar* plus an additional option *unsure*. This includes a set of 8 (query) workflows, for which similarity ratings are available covering all workflows returned by a similarity search over the whole corpus by a selection of similarity algorithms.

In the following, we first comparatively investigate how different settings and ensembles of algorithms affect the quality of the retrieved results, both confirming our previous findings [23], and sometimes even improving on them. In Section 5.1 we measure how runtime of similarity search over the whole dataset (i.e., a whole repository) is affected by indexing and different settings discussed in Section 4.

5.1. Retrieval Quality

We measure retrieval quality as the precision at k over the top 10 results retrieved by a similarity search over the whole repository. Relevance of a search result is determined by the median expert similarity rating assigned to each pair of query and result workflow in the corpus (*very similar, similar, related, dissimilar, or unsure*). We thus evaluate precision at k with the relevance thresholds of *similar* and *related*. *very similar* results are found by all presented algorithms at near equal quality (data not shown).

In our system, retrieval quality is influenced by three factors: (i) the minimum similarity of labels (fuzzyness) set for module label matching in the index, (ii) the number of candidate results retrieved and forwarded to the reranking step, and (iii) the algorithm(s) and their configurations used for reranking.

5.1.1. Minimum Similarity of Module Labels

Figure 7 shows precision at k for the top 10 results retrieved by Lucene with various settings of fuzzyness, in direct comparison to the Module Set algorithm

[2] myExperiment: http://www.myexperiment.org
Corpus: https://www.informatik.hu-berlin.de/forschung/gebiete/wbi/resources/flowalike

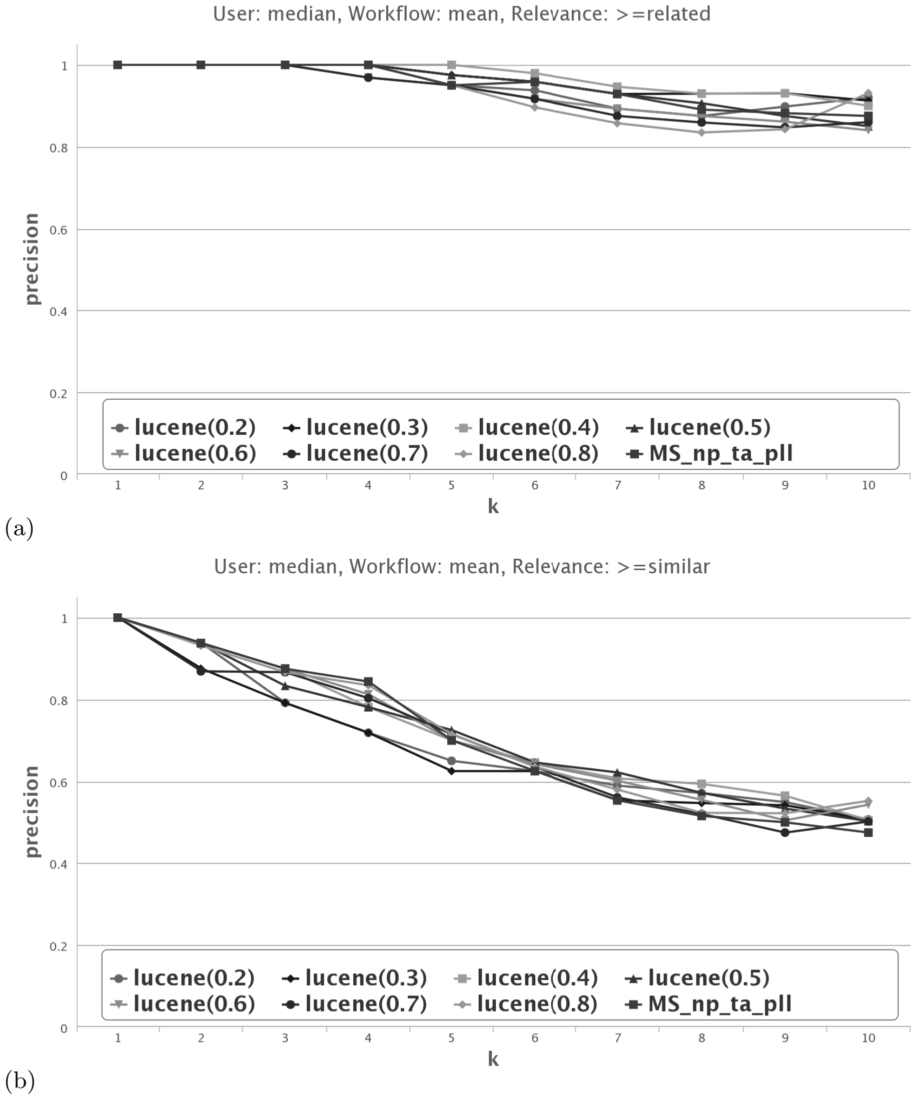


Figure 7: Mean retrieval precision at k against the median expert rating for Lucene with various settings for the minimum similarity of module labels, and Module Sets workflow comparison for relevance threshold (a) related, and (b) similar. Module Sets used with module similarity by edit distance of labels (*pll*), without *ip* and *te*.

Table 2: Numbers of query workflows out of 1483 for which less than 10, 25, and 50 results could be retrieved using the respective *minimum similarity* values.

Lucene minimum similarity	Number of query workflows yielding < 10 results	 < 25 results	 < 50 results
0.2	1	12	27
0.3	21	59	114
0.4	90	185	313
0.5	208	356	556
0.6	298	506	687
0.7	382	624	809
0.8	472	683	891
0.9	498	713	908

used in a comparable configuration: all of the workflows' modules are used for similarity assessment (no *ip* or *te*) and module labels are compared by edit distance (*pll*). Retrieval quality of Lucene is clearly on par with Module Sets. Most obviously, different fuzzyness values greatly influence the results returned. Observe that for a relevance threshold of *similar* (Fig. 7b) there is an apparent trend of higher values to deliver better results: While values of 0.2 and 0.3 are clear outliers for the negative, values of 0.4 and above provide comparable results over the first 5 positions. For the second half of the top 10, values above 0.7 appear too strict, and are outperformed by more fuzzy matching of lower minimum similarity values. This observation of strict versus fuzzy matching is confirmed at a relevance threshold of *related* (Fig. 7a), where especially values of 0.3 and 0.4 provide convincing results.

As we are ultimately interested in reranking the results retrieved by Lucene with any such setting, another aspect to consider is the number of results Lucene retrieves for each value. We searched our index with each of its 1483 workflows in turn, using different settings for fuzzyness of label matching. Table 2 reveals that values above 0.4 result in difficulties in retrieving sufficient numbers of workflows. As a consequence, we focus further evaluation on minimum similarity values of 0.2, 0.3 and 0.4.

5.1.2. Limiting reranking candidates

While for some workflows only limited lists of candidates are available, the majority of query workflows yields large lists of candidate results. For instance, with a minimum similarity value of 0.3, about half of the workflows in the repository have more than 300 candidates (one third for 0.4). We are thus interested in reducing the number of candidates for the reranking phase to only the top-x, i.e., a specific ranking cutoff. For a fuzzyness of 0.3, Figure 8 shows retrieval precision at k for reranking at different such cutoffs. We use the reranking algorithm which has, so far, proven to provide best results (to be comparatively evaluated in the next section). While all cutoffs provide rather similar performance, reranking of the 25 topmost candidates seems to deliver slightly higher result quality. First and foremost, these findings show that quality of reranked retrieval does not necessarily increase with the number of candidate results used for reranking. Apparently there is a synergetic effect of the candidate preselection done by Lucene (i.e., workflow comparison by modules only) and the more

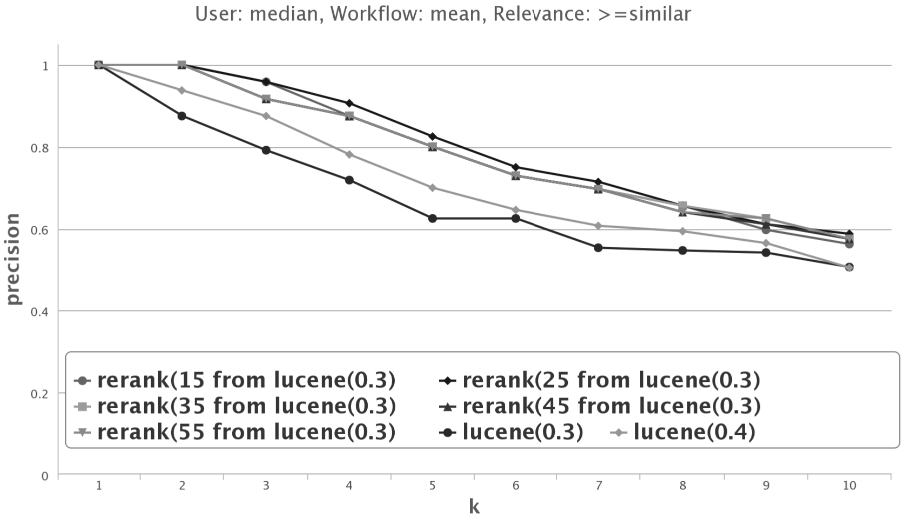


Figure 8: Mean retrieval precision at k against the median expert rating for reranking of different numbers of top-x candidates retried by Lucene with fuzzyness 0.3. Lucene standalone retrieval included as baseline. Relevance threshold similar.

complex reranking, at least on our data set.

For fuzzyness values of 0.2 and 0.4 results are equivalent, only that their peaks in performance are not at cutoffs of 25, but 35 and 15, respectively (data not shown). This shows that the differences in retrieval quality observed for different fuzzyness values in Lucene-only retrieval in the previous section, are evened out by the applied reranking. Together with the observations on the numbers of results returned by Lucene in the first place (see Table 2), it also indicates a trade-off between result quality, the amount of reranking required to achieve it, and the overall number of results available to the user: While with less fuzzy retrieval only the top 15 results have to be reranked to provide best results amongst the top-10, a significant portion of query workflows will not see 10 results at all. The more fuzzy initial retrieval is, on the other hand, the more query workflows will yield larger result sets, but the more reranking has to be applied. Which settings to use depends on the resources available for reranking.

Based on these considerations, all further evaluations use reranking of the top 24 results retrieved using Lucene with a fuzzyness of 0.3, best matching the resources used for runtime evaluation in Section 5.2.

5.1.3. Reranking algorithms

Figure 9 shows how result quality is affected by the two contrary configurations of Layer Decomposition introduced in Section 4.2. The figure also includes the ensembles of these two configurations with the annotation-based measure of *Bag of Words (BW)*. Results correlate at both relevance thresholds of related

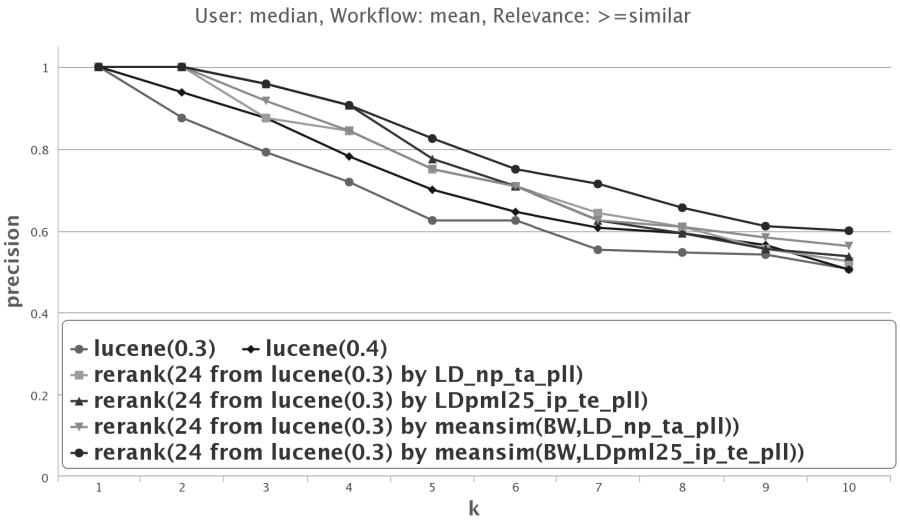


Figure 9: Mean retrieval precision at k against the median expert rating for Lucene's top 24 results reranked by different (ensembles of) algorithms for relevance threshold (a) related, and (b) similar.

and similar (see Fig. A.14, Appendix A for threshold of related). Clearly, both the amount of repository knowledge included and the use of ensembles benefit overall retrieval precision: The best reranking approach is the ensemble of fully tuned Layer Decomposition and Bag of Words. The ensemble including the naive configuration of Layer Decomposition is still outperformed by the standalone, tuned Layer Decomposition when it comes to the very top of the result list, and performs equally well as the standalone naive version. Most importantly, all reranking methods deliver better results than non-reranked retrieval by Lucene only.

5.2. Runtime

We measure the time taken for retrieval of the top 10 most similar workflows from the whole repository using a given query workflow. For Lucene, we use the default configuration of single threaded search over its index. For structure-based workflow comparison, e.g., retrieval by Module Set comparison (see below) or reranking by Layer Decomposition, a setup of 24 parallel processes is used, each handling the comparison of one pair of query workflow and workflow from the repository at a time. While this is a reasonable setup in a server-based environment, it has to be kept in mind that fully sequential comparison of all pairs of workflows would take substantially longer.

5.2.1. Search Phase

Figure 10 plots runtimes of similarity search over the whole repository for the Module Sets similarity algorithm and Lucene against the number of modules

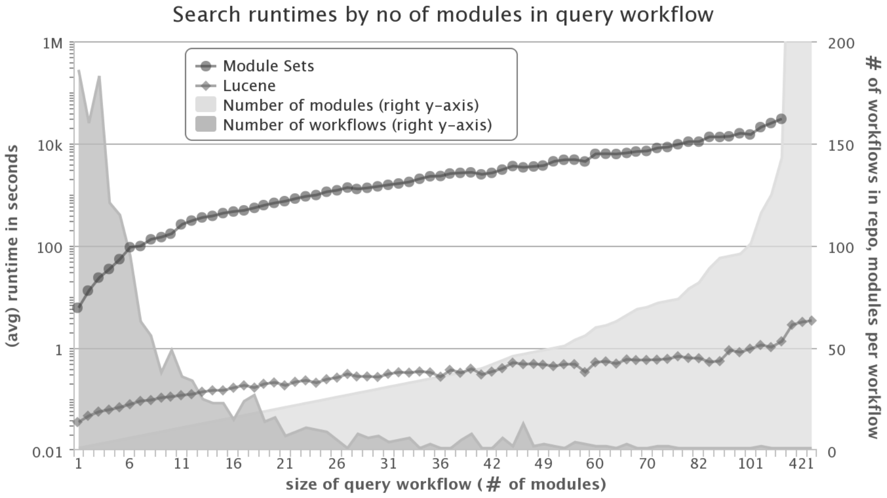


Figure 10: Average runtime (left logarithmic y-axis) of similarity search over repository of 1483 workflows using Module Set workflow comparison (MS) or Lucene, by number of modules in query workflow. Areas (right, linear y-axis): No of workflows in repository with the corresponding number of modules; No of modules (as per value on x-asis)

in the query workflow (again, note that Module Set comparisons are parallelized 24-fold). For small workflows, Module Sets is respectably fast, given the fact that it does not make use of any index structures but has to compare each pair of query workflow and repository workflow separately. With only 5 modules in the query workflow, search times average at one minute already, and double once more at 8 modules. Lucene is faster by several orders of magnitude due to its indexing. Only for very large workflows do retrieval times reach or exceed one second. Observe that workflow sizes on the x-axis don't increase linearly - for a better visual grasp, we also plot the x-values themselves, showing a steep increase of workflow sizes towards the right end of the plot. Furthermore, for a better feel of how the retrieval times translate to the repository's contents, the darker area graphs the number of workflows contained in the repository which have the corresponding number of modules. Clearly, most workflows in this specific repository are rather small, which corresponds to previously observed [26, 22] average workflow sizes of app. 11 modules.

5.2.2. Reranking Phase

After retrieval of candidates from the index we rerank the top 24 results. Figure 11 shows runtimes of both configurations of Layer Decomposition. The tuned configuration is faster by an order of magnitude. Note that we don't show runtimes of ensembles - including Bag of Words in reranking has practically no effect on runtimes.

The advantage of the tuned algorithm over the naive one is a consequence of the speedups listed in Table 1. While these speedups already lead to overall

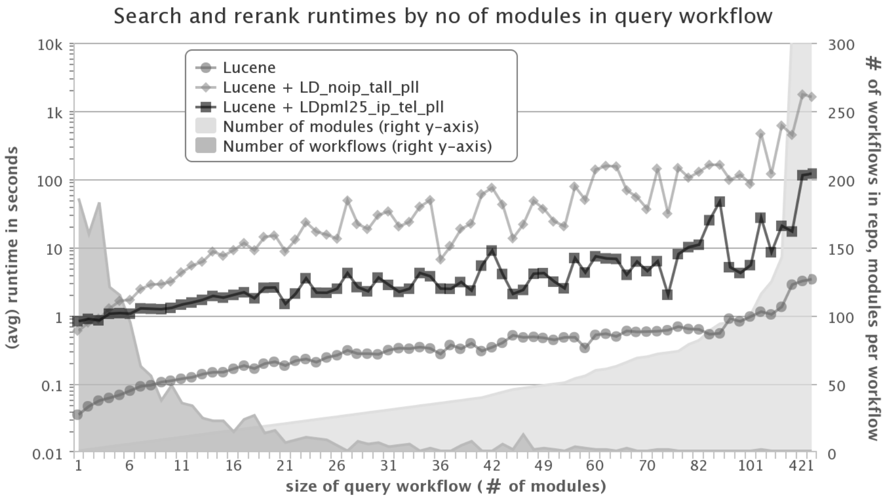


Figure 11: Average runtime (left logarithmic y-axis) of similarity search over repository of 1483 workflows using Lucene and reranking of top 24 candidates by either of two configurations of Layer Decomposition (see text); plotted by number of modules in query workflow. Areas (right, linear y-axis): No of workflows in repository with the corresponding number of modules; No of modules (as per value on x-asis)

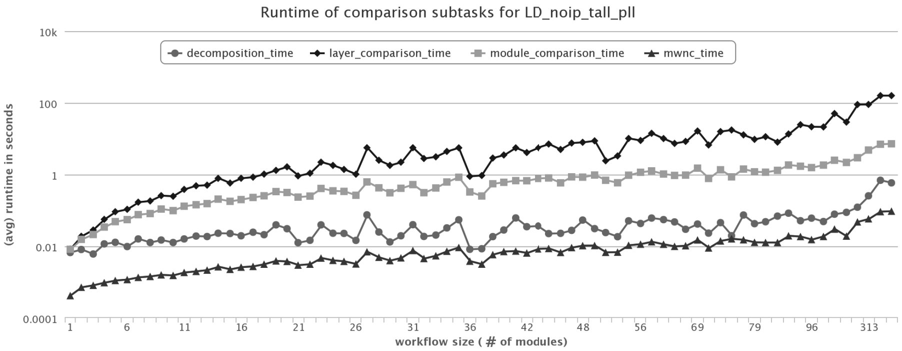


Figure 12: Runtimes of subtasks of Layer Decomposition workflow comparison in dependence of workflow size using maximum weight matching of modules for assessing similarity of single pairs of layers.

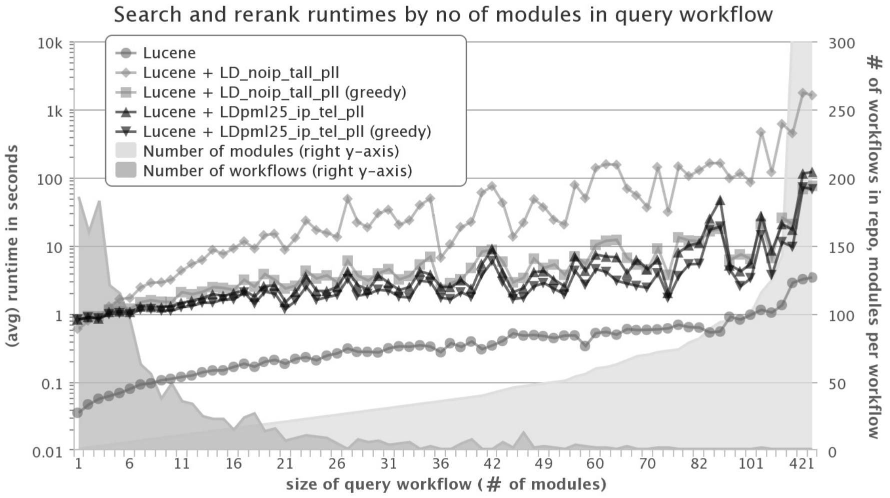


Figure 13: Average runtime (left logarithmic y-axis) of similarity search over repository of 1483 workflows using Lucene and reranking of top 24 candidates by either of two configurations of Layer Decomposition (see text); plotted by number of modules in query workflow. Areas (right, linear y-axis): No of workflows in repository with the corresponding number of modules; No of modules (as per value on x-axis)

similarity search times of less than 10 seconds for the vast majority of workflows in the repository (reranking only the top 24 results), our interest is to see exactly which portion of the Layer Decomposition algorithm is taking how long, and if further speedup is possible that does not include repository knowledge. Dissecting the workflow comparison of Layer Decomposition, Figure 12 reveals that the major part of time is spent comparing all pairs of modules (per edit distance), and comparing all pairs of layers. The times taken for decomposing a workflow into its layers in the first place, and for computing the maximum weight non-crossing matching of those layers after their pairwise comparison are much lower.

Recall from Section 3 that Layer Decomposition uses the *maximum weight matching* of the sets of modules two layers are composed of to determine layerwise similarity. The algorithmic complexity of computing such a matching of two layers L_1 and L_2 is $O((L_1 \cup L_2))^2 L_1 L_2)$. Most interestingly in this respect, we have shown in [22] that for workflow comparison by Module Sets, using greedy mapping of modules instead of maximum weight matching provides equivalent results. Applying this finding to the layer comparison step of Layer Decomposition yields a significant speedup both of the time taken for this specific step, leveling it with the time taken for decomposition (see Figure A.15, Appendix A), and of the time taken for similarity search as a whole: Figure 13 reiterates our previous illustration with additional runtimes plotted for the greedy versions of the reranking algorithms. Especially for the zero-knowledge configuration, the improvement is substantial. And, while less accentuated, the fully tuned version of Layer Decomposition benefits from the shift in complexity as well, now yielding overall search times under 5 seconds for most workflow sizes and an average runtime of 1.4 seconds over all 1483 query workflows. Most importantly, retrieval quality remains unchanged (data not shown).

6. Related Work

Current search capabilities offered by scientific workflow repositories focus on keyword queries [2, 14, 18, 19]. In the myExperiement scientific workflow repository [19], for instance, these queries are matched against a Lucene index on the annotations supplied by the workflows' authors and the labels of the workflows' modules [12]. Matching uses exact string matching and workflow structure is not considered. While previous work has investigated several options to determine similarity of scientific workflows based on workflow structure, e.g, [21, 20, 25, 3, 13, 11, 27], only litte work exists that targets their efficiency of similarity search over whole repositories. A system using subgraph matching to search a repository of scientific workflows using a given query workflow is proposed in [13]. The authors report runtimes of 15 seconds over a repository of 89 workflows and manually evaluate retrieval quality on an example workflow (explicitly rejecting to derive generic claims). In [7] workflows are represented as context free bag grammars. This allows effective matching of keywords specified in an user's query against the (possible) executions of a workflow and the corresponding execution traces of its modules. In [4], a system for similarity search is proposed that uses manually added semantic anntations on workflows and their components. The architecture of the proposed system is analogous to the two-phased approach introduced in the previous section: An initial set of candidate results is retrieved by a fast search using an index over the semantic annotations the workflows contain. These candidates are then reranked by a graph-based approach that determines workflow similarity as the maximum aggregate similarity of the nodes (i.e., modules) contained in a respective mapping between two workflows. Runtime is shown to clearly benefit from the two-step approach, but retrieval quality of the compound system is evaluated against the standalone graph-based approach only. For business workflows, [28] target a similar, two-step approach, specifically addressing the first step of candidate retrieval: A selection of features derived from the graph structure of the workflows is used to index the workflows in a repository. This index is used to assess whether a workflow from the repository is 'irrelevant' to a search or 'potentially relevant'. The candidates from the latter class can then be processed by more complex similarity measures, e.g., [10, 8, 9]. In contrast to any of these approaches, the system proposed here does not rely on graph indexing but is based on simple indexing and candidate retrieval using only the workflows' modules. Our evaluation uses the largest collection of real-world workflows and systematically considers both retrieval speed and quality.

7. Conclusion

We introduced a system for structure-aware similarity search in scientific workflow repositories. Relying on off-the-shelf indexing technology in form of Lucene and using a two-phase approach of candidate retrieval and reranking, we were able to apply the high quality structure-based Layer Decomposition workflow similarity measure at repository-scale with acceptable speed. We distinguished two cases of application, where knowledge about the repository and its workflows is available for fine-tuning of structure-based comparison - or not. For both of these cases we showed that reranking of results clearly improves retrieval quality of similarity search, and closely investigated their runtime properties. The speedup achieved by greedy assessment of layer similarity in workflow comparison even for the naive version of Layer Decomposition is promising for the application of this method to repositories where external knowledge such as for importance projection or layer mismatch penalty are not available.

On the more conceptual level, regarding the three parameters of fuzzyness of initial candidate retrieval, top-x cutoff of candidates fed to the reranking step, and reranking method to set for optimizing retrieval quality, we found that for the first two, no single best setting exists for best result quality, but the most appropriate setting depends on available computational resources. In this respect, another interesting option may be to not select a fixed number of candidates to rerank, but to use a dynamic cutoff based on the similarity values found during reranking. For the last parameter, the reranking method to apply, a clear trend can be observed: The more knowledge about a repository and its workflows is available, and the more different perspectives on workflow similarity can be applied, the better. Especially the latter point of different perspectives puts a hard eye on future research to access other sources of workflow meta-data for similarity assessment, one such source being provenance.

8. Acknowledgments

This work was partly funded by DFG grant GRK1651, DAAD grants D1240894 and 55988515, the EU FP7 Grant 317871, and PHC Procope grant. Work of SCB partly done in the context of the Institut de Biologie Computationnelle, Montpellier, France.

Appendix A. Additional Figures

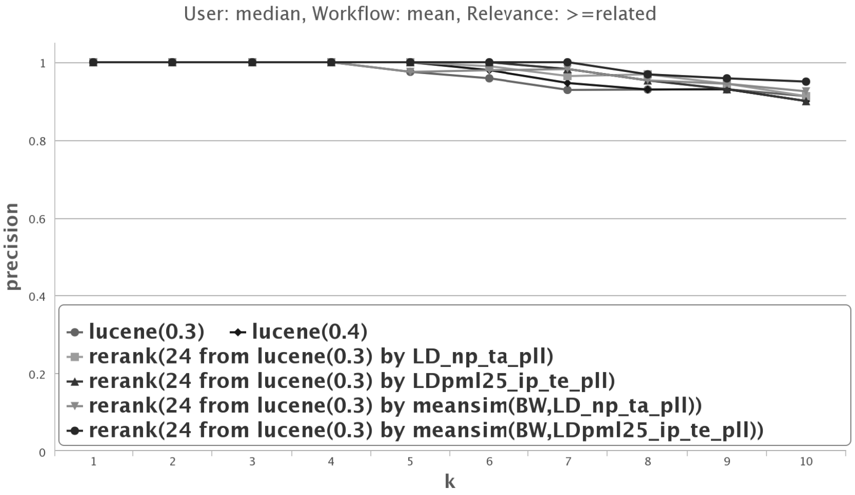


Figure A.14: Mean retrieval precision at k against the median expert rating for Lucene's top 24 results reranked by different (ensembles of) algorithms for relevance threshold related.

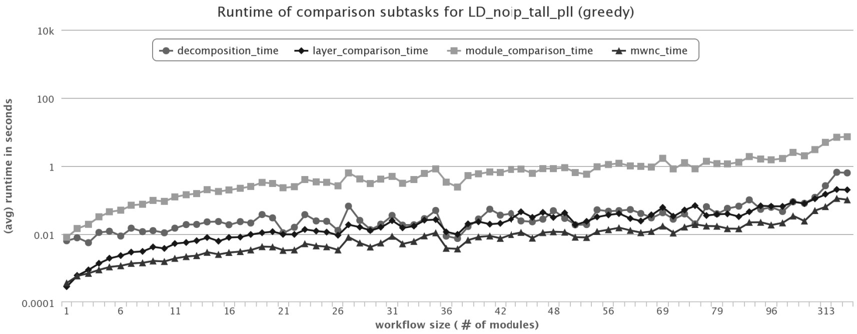


Figure A.15: Runtimes of subtasks of Layer Decomposition workflow comparison in dependence of workflow size using greedy matching of modules for assessing similarity of single pairs of layers.

[1] neo4j graph database. *http://neo4j.com/*.

[2] SHIWA workflow repository. *http://shiwa-repo.cpc.wmin.ac.uk*.

[3] R. Bergmann and Y. Gil. Similarity assessment and efficient retrieval of semantic workflows. *Information Systems*, 40:115–127, 2012.

[4] R. Bergmann, M. Minor, S. Islam, P. Schumacher, and A. Stromer. Scaling similarity-based retrieval of semantic workflows. In *ICCBR-Workshop on Process-oriented Case-Based Reasoning, Lyon*, pages 15–24. Citeseer, 2012.

[5] S. Cohen-Boulakia and U. Leser. Search, Adapt, and Reuse: The Future of Scientific Workflow Management Systems. *SIGMOD Record*, 40(2):6–16, 2011.

[6] F. Costa, D. d. Oliveira, E. Ogasawara, A. Lima, and M. Mattoso. Athena: Text Mining Based Discovery of Scientific Workflows in Disperse Repositories. *RED*, pages 104–121, 2010.

[7] S. B. Davidson, X. Huang, J. Stoyanovich, and X. Yuan. Search and Result Presentation in Scientific Workflow Repositories. *SSDBM*, 2013.

[8] R. Dijkman, M. Dumas, B. V. Dongen, R. Käärik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *Information Systems*, 2010.

[9] R. Dijkman, M. Dumas, and L. García-Bañuelos. Graph matching algorithms for business process model similarity search. *Business Process Management*, 2009.

[10] M. Dumas, L. García-Bañuelos, and R. Dijkman. Similarity search of business process models. *Data Engineering Bull.*, 2009.

[11] N. Friesen and S. Rüping. Workflow Analysis Using Graph Kernels. *SoKD*, 2010.

[12] A. Goderis, D. De Roure, C. Goble, J. Bhagat, D. Cruickshank, P. Fisher, D. Michaelides, and F. Tanoh. Discovering scientific workflows: The myexperiment benchmarks. *IEEE Transactions on Automation Science and Engineering*, 2008.

[13] A. Goderis, P. Li, and C. Goble. Workflow discovery: the problem, a case study from e-Science and a graph-based solution. *ICWS*, pages 312–319, 2006.

[14] J. Goecks, A. Nekrutenko, and J. Taylor. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010.

[15] O. Gospodnetic and E. Hatcher. *Lucene*. Manning, 2005.

[16] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 1932.

[17] F. Malucelli, T. Ottmann, and D. Pretolani. Efficient labelling algorithms for the maximum noncrossing matching problem. *Discrete Applied Mathematics*, 47(2):175–179, 1993.

[18] P. Mates, E. Santos, J. Freire, and C. Silva. Crowdlabs: Social analysis and visualization for the sciences. *SSDBM*, pages 555–564, 2011.

[19] D. Roure, C. Goble, and R. Stevens. The design and realisation of the myexperiment virtual research environment for social sharing of workflows. *Future Generation Computer Systems*, 25(5):561–567, 2009.

[20] E. Santos, L. Lins, J. Ahrens, J. Freire, and C. Silva. A First Study on Clustering Collections of Workflow Graphs. *IPAW*, pages 160–173, 2008.

[21] V. Silva, F. Chirigati, K. Maia, E. Ogasawara, D. Oliveira, V. Braganholo, L. Murta, and M. Mattoso. Similarity-based Workflow Clustering. *CCIS*, 2(1):23–35, 2010.

[22] J. Starlinger, B. Brancotte, S. Cohen-Boulakia, and U. Leser. Similarity Search for Scientific Workflows. *PVLDB*, 7(12), 2014.

[23] J. Starlinger, S. Cohen-Boulakia, S. Khanna, S. B. Davidson, and U. Leser. Layer decomposition: An effective structure-based approach for scientific workflow similarity. *IEEE 10th International Conference on eScience*, 2014.

[24] J. Starlinger, S. Cohen-Boulakia, and U. Leser. (Re)Use in Public Scientific Workflow Repositories. *SSDBM*, pages 361–378, 2012.

[25] J. Stoyanovich, B. Taskar, and S. Davidson. Exploring repositories of scientific workflows. *WANDS*, pages 7:1–7:10, 2010.

[26] I. Wassink, P. Vet, K. Wolstencroft, P. Neerincx, M. Roos, H. Rauwerda, and B. T.M. Analysing Scientific Workflows: Why Workflows Not Only Connect Web Services. *Services*, pages 314–321, 2009.

[27] X. Xiang and G. Madey. Improving the Reuse of Scientific Workflows and Their By-products. *ICWS*, pages 792–799, 2007.

[28] Z. Yan, R. Dijkman, and P. Grefen. Fast business process similarity search with feature-based similarity estimation. In *On the Move to Meaningful Internet Systems: OTM 2010*, pages 60–77. Springer, 2010.