Graph *n*-grams for Scientific Workflow Similarity Search

David Luis Wiegandt, Johannes Starlinger, and Ulf Leser

Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
{wigandtd,starling,leser}@informatik.hu-berlin.de
https://www.informatik.hu-berlin.de

Abstract. As scientific workflows increasingly gain popularity as a means of automated data analysis, the repositories such workflows are shared in have grown to sizes that require advanced methods for managing the workflows they contain. To facilitate clustering of similar workflows as well as reuse of existing components, a similarity measure for workflows is required. We explore a new structure-based approach to scientific workflow similarity assessment that measures similarity as the overlap in local structure patterns represented as *n*-grams. Our evaluation shows that this approach reaches state-of-the-art quality in scientific workflow comparison and outperforms some established scientific workflow similarity measures.

Keywords: Scientific Workflows, DAGs, n-grams, graph theory

1 Introduction

Over the last years, *scientific workflows (SWFs)* have emerged as a useful means of data analysis, particularly in the life sciences. SWFs are shared in (online) repositories to optimally support non-experts who cannot develop SWFs by themselves. A problem that arises with the growth of such repositories is the occurrence of (partial) duplicates. To avoid duplicates in the first place and enforce the reuse of existing components instead, and to allow for grouping of workflows that fulfil similar tasks (e.g. for result presentation), a similarity measure for SWFs is required. The problem of defining adequate similarity measures has been attacked in various ways, either based on the SWFs' graph structure or on their associated description and annotations in repositories. In this work, we focus on graph-based approaches as they are in principal applicable to arbitrary workflows, in contrast to approaches that rely on the presence of annotations.

Graphs can be compared using either global or local measures. A systematic approach to categorising and evaluating various established measures [14] yields that algorithms that consider a graph's full structure are too strict in the context of SWF similarity assessment. On the contrary, algorithms that fully ignore any structural information and only perceive graphs as sets of vertices are too loose, as two workflows that share a high amount of common tasks but with very different interconnections, should not be declared as similar. Instead, approaches that retain substructures only to a certain degree yield promising results [1,15,17].

Since it is our aim to measure the similarity of SWFs that are stored in repositories, we particularly focus on algorithms that decompose graphs into smaller units. These units, encoding a certain amount of structure, can be computed once when the workflow is added to the repository, and be reused in subsequent similarity assessments, making the computational complexity of the decomposition process negligible. In information retrieval, one way to represent such units are *n*-grams, a concept which has been applied to graphs as well.

For instance, in [17], workflows are perceived as finite state automata and the automata's states are encoded into fixed-size words called *n*-grams. Two workflows' similarity is measured by the overlap in their *n*-gram-sets. This approach was proven to outperform other similarity measures in the more general field of workflow similarity assessment, however, it cannot easily be transferred to SWFs, as, in general, SWFs lack transition labels that are required to convert them to FSAs. A similar approach that aims at similarity assessment of trees is presented in [1]. In [19], an *n*-gram-based approach to measuring two graphs' similarity in the context of graph similarity joins was presented as an alternative to the computationally expensive graph edit distance. It is among the state-of-the-art algorithms in this field [3].

Since *n*-gram-based approaches have proven to be successful in various domains, in this work, we transfer the concept to SWF similarity assessment. To this end, we particularly focus on *n*-grams that represent sub-paths of length *n*. For our evaluation, we make use of the gold standard corpus of manually retrieved similarity ratings introduced in [14].

In the following, we start with formal definitions of graphs and workflow graphs in Section 2, that we refer to in the subsequent review of existing solutions in Section 3. In Section 4, we introduce our novel approach which we evaluate in Section 5. In Section 6, we make specific propositions on how to improve our approach in the context of possible future work and close with some concluding remarks.

2 Preliminaries

2.1 Graphs

A *directed graph* $G = (V,E)$ is a tuple consisting of a vertex set V and an edge set $E \subseteq V \times V$. With $[n] := \{i \in \mathbb{N} : 1 \leq i \leq n\}$ we usually assume $V = [n]$ for some $n \in \mathbb{N}$. A *path* of length n is a sequence of vertices $v_1,\ldots,v_n \in V$ that are connected by edges, i.e. $(v_{i-1},v_i) \in E, i = 2\ldots n$. If $v_1 = v_n$, the path is called a *cycle*. Directed graphs without cycles are called directed acyclic graphs *(DAGs)*. For the rest of this work, when talking about graphs we always mean DAGs if not explicitly stated otherwise.

With $\hat{E} := \{\{v,w\} : (v,w) \in E\}$, we call $\hat{G} := (V,\hat{E})$ the *underlying undirected graph* of G. A graph G is *connected* if there exists a path between any two vertices in V and *weakly connected* if only $\hat{G}$ is connected. A *vertex-labelled* graph is a quadruple $G = (V,E,\Sigma,l)$ with labels over a finite alphabet Σ and a function $l : V \to \Sigma^{\star}$, although we usually do not explicitly state the alphabet and the labelling function. An *edge-labelled* graph is defined analogously. The ratio of $|E|$ and the graph's maximum possible amount of edges is called its *density*.

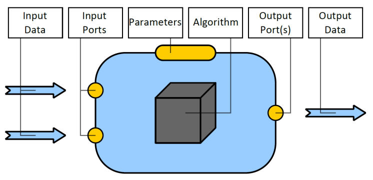
Given any $v \in V$, we call $suc(v) = \{w \in V : (v,w) \in E\}$ its set of *successors* and $pre(v) = \{w \in V : (w,v) \in E\}$ its set of *predecessors*. Furthermore, we denote a vertices' *in-degree* by $\deg^-(v) = |pre(v)|$ and its *out-degree* by $\deg^+(v) = |suc(v)|$. We refer to $b(G) = \frac{|E|}{|V|}$ as the *average branching factor* of G, being the average out-degree (and in-degree) of any vertex $v \in V$. For some $S \subset V$ we call $G[S] = (S, E[S])$ with $E[S] = \{(v,w) \in E : v,w \in S\}$ the vertex-induced subgraph of G.

2.2 Workflow Graphs

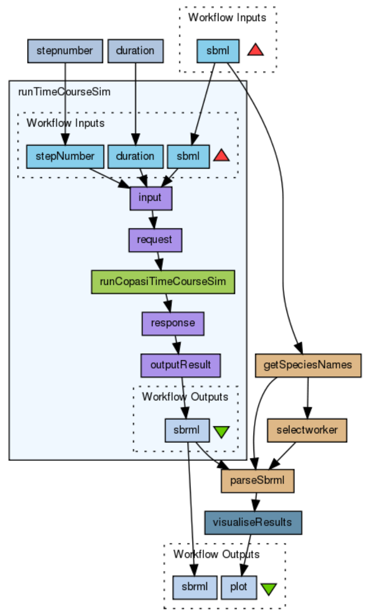
Prodan et al define an SWF recursively as a tuple $G = (V, E_c, E_d, IN, OUT)$ [10, 204 ff.] with IN being the workflow's set of input ports, OUT being its set of output ports and V being its set of *tasks*. Each task $v \in V$ computes a function $v : IN^{(v)} \to OUT^{(v)}$. By connecting one task's output ports to another task's input ports, *control* and *data* flow dependencies $E_c, E_d \subseteq V \times V$ are introduced.

Following *Prodan et al*, we distinguish *atomic* and *composite* tasks, the latter being a set of nested sequential and parallel tasks that can, in turn, be composite and atomic tasks. Figure 1a shows an atomic task, whereas Figure 1b shows a complete SWF (itself a composite task) consisting of multiple atomic tasks and a composite task `runTimeCourseSim` we call a *sub-workflow*. Relations $\prec_c, \prec_d \subseteq V \times V$ are defined with $\prec_c$ being the control flow precedence and $\prec_d$ being the data flow precedence. Given two activities $v_1, v_2 \in V$, we write $v_1 \prec_c v_2$ iff v_1 can not be executed before v_2 finishes. Similarly, we write $v_1 \prec_d v_2$ iff v_1 can not be executed without being provided with data by v_2. In contrast to *Prodan et al*, in this study we do not require $v_1 \prec_d v_2 \implies v_1 \prec_c v_2$ as we focus on the *Apache Taverna* Workflow Management System, which allows pipelining and streaming of data between tasks [16].

The set V of a workflow can be further subdivided into disjoint sets A (activities), J_{AND} (AND-Joins), J_{OR} (XOR-Joins), S_{AND} (AND-Splits) and S_{OR} (XOR-Splits) [5], allowing for conditional branching and parallel processing. This workflow model can be further extended, e.g. by adding a labelling function.



(a) A labelled task of an SWF [2]



(b) An SWF with a sub-workflow [7]

Fig. 1: SWF components

We make use of the fact that most *scientific workflow management systems* (such as Apache Taverna) prohibit looping, i.e. cycles in the workflow graph. This allows us to represent SWFs as DAGs [16]: We set $E = E_c \cup E_d$ and omit the explicit mentioning of IN and OUT, because we are only interested in the underlying DAG's structure, which can be described by its set of vertices and edges as introduced in the previous section.

3 Related Work

The problem of workflow similarity assessment has been attacked in various different ways. As we focus on graph-based approaches, we group the approaches outlined in this section wrt. their preservation of topology and structure and distinguish (1) *local* approaches that completely disregard the graphs' structure, (2) *global* approaches that preserve the graphs' structure (almost) entirely and (3) *alternative approaches* that aim to compromise between local and global by preserving *sub*structures.

Local Measures. Considering only the workflows' tasks, in [13], the Vector Space Model (VSM) as known from information retrieval [12] is adapted for workflow comparison. Given two vertex-labelled workflow graphs $G_i = (V_i, E_i, \Sigma_i, l_i)$ for $i = 1,2$, each workflow is represented as a k-dimensional vector $D_i = (d_{i,1}, \ldots, d_{i,k})$ with $T = l_1[V_1] \cup l_2[V_2] = \{t_1, \ldots, t_k\}$ being the set of all vertex labels in both workflows and $d_{i,j} = |\{v \in V_i : l(v) = t_j\}|, j = 1..k$ being the amount of vertices in the i-th workflow that are labelled with t_j. In a last step, the similarity is computed as the cosine similarity between both workflows' vectors.

Another approach called *Module Sets (MS)*, that is similar to the VSM in that it completely disregards substructures by perceiving a workflow graph only as a set of tasks (called *modules* in the referenced work), proved to be comparably reliable and fast [14]: Given SWF graphs $G_i = (V_i, E_i)$ for $i = 1,2$, all pairs of tasks in $V_1 \times V_2$ are compared (e.g. with regard to their label's string edit distances).

$$sim_M : V_1 \times V_2 \to [0,1] \subset \mathbb{R} \tag{1}$$

Based on sim_M, a *maximum weight* matching $mw \subseteq V_1 \times V_2$ is computed and the similarity score is given by eq. (2).

$$nnsim_{MS}(V_1, V_2) = \sum_{(v_1, v_2) \in mw} sim_M(v_1, v_2) \tag{2}$$

In a last step, the similarity score is normalised over the workflows' size using a modified Jaccard index:

$$sim_{MS}(V_1, V_2) = \frac{nnsim_{MS}(V_1, V_2)}{|V_1| + |V_2| - nnsim_{MS}(V_1, V_2)} \tag{3}$$

Global Measures. An alternative examined in the same work [14] that turned out to provide more stable results than module sets is to retain substructures by comparing *path sets*. Given a DAG $G = (V, E)$, we denote the set of all paths of length n that start at a node with no inbound edges and end in a node with no outbound edges by $P_n(G)$

and the set of all such paths of arbitrary length by $PS(G)$:

$$P_n(G) = \{(v_1, \dots, v_n) \in V^n : (v_{i-1}, v_i) \in E, i = 2 \dots n, \deg^-(v_1) = \deg^+(v_n) = 0\} \quad (4)$$

$$PS(G) = \bigcup_{n=1}^{|V|} P_n(G) \quad (5)$$

Given SWF graphs G_1, G_2 and their path sets $PS_1 := PS(G_1), PS_2 := PS(G_2)$, the computation of the pairwise similarity score between paths (sim_P), the non-normalised ($nnsim_{PS}$) and the normalised similarity score (sim_{PS}) are defined analogous to the module sets approach by using paths instead of modules. To compute the similarity of two paths $P_1 = v_1, \dots, v_n, P_2 = w_1, \dots, w_m$, a *maximum weight non-crossing (mwnc) matching* is computed on their vertices, i.e. for all $(v_i, w_j), (v_k, w_l) \in mwnc \subseteq P_1 \times P_2$ it holds that $k > i \implies l > j$ and $k < i \implies l < j$ [8].

Another common graph distance measure is the graph edit distance. Given graphs G_1, G_2 and the three edit operations *insert*, *delete* and *substitute* (all applicable to edges and nodes), each with a certain cost assigned, the graph edit distance is defined as the minimum cost sequence of edit operations to transform G_1 into G_2 or vice versa [11]. Despite being optimal in terms of accuracy in finding structural differences between graphs, the graph edit distance's time complexity is exponential, making it inappropriate for large graphs. Further, it turned out to be too strict for SWF similarity assessment and yields comparably bad results [14].

Alternative Approaches. In [14], it was shown that graph comparison algorithms that consider the graphs full structure appear to be too strict for fine grained assessment of SWF similarity, whereas the comparison of SWFs by their module sets turned out to provide convincing results. Comparing substructures and focusing on the execution order of the workflows' tasks, on the other hand, turned out to lead to more reliable results, but is computationally expensive.

In [15], a new approach to SWF similarity search, that aims to compromise between local and global approaches, is introduced. *Layer Decomposition (LD)* performs a topological decomposition of a given SWF graph, i.e. it decomposes the graph into disjoint sets of vertices depending on the vertices relative position within their data flow strand. This is done by repeatedly removing all vertices with no inbound edges and their associated outbound edges. Given a graph $G = (V, E)$, the i-th layer L_i is defined recursively:

$$L_1(G) = \{v \in V : \deg^-(v) = 0\} \quad (6)$$

$$L_i(G) = \{v \in G[L_{i-1}(G)] : \deg^-(v) = 0\} \quad (7)$$

The ordered set $LD(G)$ that contains all layers is called the graph's layer decomposition. To compare two graphs G_1, G_2 by their layer decompositions $LD_1 := LD(G_1), LD_2 := LD(G_2)$, the approach of computing $nnsim_{LD}$ and sim_{LD} is analogous to the path and module sets, except for the use of a maximum weight non-crossing matching between layers and a maximum weight matching between two layers' vertices. LD is able to outperform other algorithms for SWF similarity assessment, including path sets, module sets and the graph edit distance [15].

The distinctive feature of layer decomposition and path set comparison is that the matching between two workflows' tasks within the retrieved substructures is performed as a part of the similarity assessment, while the other approaches assume a global matching to be computed beforehand. Two workflows' tasks are matched with respect to their execution order, because tasks in one workflow can only be mapped to tasks that are within the matched layer or path in the other workflow. Since the matching is non-crossing, the overall similarity depends on the extent to which both workflows overlap in their module sets but also on the similarity of the order the tasks appear in. This is important as two workflows might share a great amount of common tasks (i.e. they have similar module sets) but due to their differing execution order, the workflows might implement different functionalities.

In the related area of automata theory, an n-gram-based approach was introduced [17] in which a workflow with transition labels in $\Sigma^\star$ is perceived as a *finite state automaton (FSA)* $M = (Q, \Sigma, \delta, q_0, F)$ with states Q, an alphabet Σ, a transition function $\delta : Q \times \Sigma \to Q$, an initial state $q_0 \in Q$ and final states $F \subseteq Q$. We define $\hat{\delta}(q, w)$ as the state of M after reading a word w, starting in state q:

$$\hat{\delta} : Q \times \Sigma^\star \to Q \tag{8}$$

An n-gram is defined as a word $w \in \Sigma^n$ with $w = w_1 w_2 \dots w_n$, s.t. there exists a state from which on the word w is recognized, ending in a state $q \in Q$:

$$S_n(q) = \{w \in \Sigma^n : \exists r \in Q : \hat{\delta}(r, w) = q\} \tag{9}$$

The idea is to represent an automaton by the union of all states' n-grams and to measure two FSAs distance by the minimum sum of edit distances (the minimum cost sequence of insertions, deletions and substitutions for transforming one given string into another [18]) across all pairs of n-grams.

It was shown in [18] that this approach empirically outperforms certain structure- and language-based algorithms.

In summary, it can be observed that it is important to partially retain a workflow's structural topology and thus the execution order, but the increase in computational complexity should not be too high. Module and path set comparison, as well as the graph edit distance and layer decomposition have all been evaluated in the context of SWF comparison. In the following, we investigate the adaption of the concept of n-grams for SWFs.

4 Novel Approach

We propose a new approach that takes local graph patterns into account to compare graphs by measuring their commonalities regarding certain patterns. First, we introduce the notion of n-grams as used in our approach (similar to [17]). We assume $\mathbb{S}$ to be the set of all SWF graphs.

Definition 1 (n-gram). *Given a graph $G = (V, E) \in \mathbb{S}$, we define the function $NGS_n : \mathbb{S} \to \mathbb{P}(V^n)$ to compute a set of all paths of length n as*

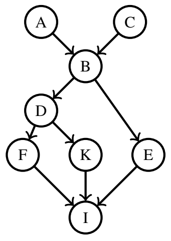
$$NGS_n(G) = \{(v_1, v_2, \dots, v_n) \in V^n | (v_i, v_{i+1}) \in E, i = 1 \dots n-1\} \tag{10}$$

with $NGS_n(G)$ being a set of n-grams of vertices.

Algorithm 2: Path n-gram

```
input  : Graph $G = (V, E)$, $n$
output: $n$-grams $\subseteq V^n$
procedure compute_paths($G = (V, E)$, $v \in V$, $depth$)
    if $depth = 0$ then
        return $\{(v)\}$;
    else
        $paths \leftarrow \emptyset$;
        foreach $w \in$ suc($v$) do
            $tmp \leftarrow$ compute_paths($G$, $w$, $depth - 1$);
            foreach $(u_1, \ldots, u_{depth}) \in tmp$ do
                $paths \leftarrow paths \cup \{(v, u_1, \ldots, u_{depth})\}$;
            end
        end
        return $paths$;
    end
end
$n$-grams $\leftarrow \emptyset$;
foreach $v \in V$ do
    $n$-grams $\leftarrow$ $n$-grams $\cup$ compute_paths($G$, $v$, $n - 1$);
end
return $n$-grams;
```

Example 1 (n-grams). We illustrate the concept of n-grams by calculating all 3-grams for the graph on the right. To simplify the notation, we denote the tuples by concatenating their vertices. We obtain the following 3-gram set:



$$NGS_3(G) = \{\texttt{ABD}, \texttt{ABE}, \texttt{BDF}, \texttt{BDK}, \texttt{BEI}, \texttt{CBD}, \texttt{CBE}, \texttt{DFI}, \texttt{DKI}\}$$

For the computation of a graph's set of n-grams we propose algorithm 2, based on a modified *iterative deepening depth-first search (IDDFS)* [6]. The decision to propose an IDDFS-based algorithm instead of a Floyd-Warshall-based algorithm was made because the Floyd-Warshall algorithm is less efficient for graphs with a low density, which is, in turn, correlated with the graph's average branching factor. It has been shown in [9], that most workflows have a rather low average branching factor, which also implies a low density. Our observations regarding our evaluation corpus, where the median density is approximately 0.3, support these findings. Our evaluation corpus (introduced in [14]) contains 1483 of the 2123 SWFs in the *myExperiment* repository, the largest public SWF repository to date.

To derive a measure from two graphs' sets of n-grams, we first introduce a method to compare two n-grams. For the remainder of this section, we assume two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ and refer to their accompanying n-gram sets by $NGS_1 := f(G_1) \subseteq V_1^n, NGS_2 := f(G_2) \subseteq V_2^n$ with $f \in \bigcup_{n \in \mathbb{N}} NGS_n$.

Since an n-gram $a = (a_1, \ldots, a_n) \in NGS_n(G)$ for some graph $G = (V, E)$ is a vector in the n-dimensional vector space V^n, we refer to the i-th component of an n-gram by a_i.

Definition 2 **(n-gram similarity).** *Given two graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$, a similarity function $sim : V_1 \times V_2 \to [0,1]$ and n-gram sets NGS_1, NGS_2, we define the similarity of two n-grams $a \in NGS_1, b \in NGS_2$ as*

$$sim_{NG}(a,b) = \frac{1}{n} \sum_{i=1}^{n} sim(a_i, b_i) \tag{11}$$

In our case, as well as in [15], the *sim*-function is derived from the normalised string edit distance between the tasks' labels a_i, b_i. To retrieve a set of mappings of n-grams that are present in both sets wrt. the global maximum weight matching, we introduce the n-gram mapping operator:

Definition 3 **(n-gram mapping operator).** *Given two n-gram sets NGS_1, NGS_2 and a global maximum weight matching $mw \subseteq V_1 \times V_2$ between both workflows' tasks, we define the set of common n-grams as*

$$NGS_1 \otimes^{mw} NGS_2 := \{(a,b) \in NGS_1 \times NGS_2 : (a_i, b_i) \in mw, i = 1..n\} \tag{12}$$

The similarity of two n-gram sets NGS_1, NGS_2 is calculated as the sum of the similarities of all n-gram mappings in $NGS_1 \otimes^{mw} NGS_2$:

Definition 4 **(n-gram set similarity measure).** *Given two n-gram sets NGS_1, NGS_2, we define the non-normalised similarity as*

$$nnsim_{NGS}(NGS_1, NGS_2) := \sum_{(a,b) \in NGS_1 \otimes^{mw} NGS_2} sim_{NG}(a,b) \tag{13}$$

To normalise this similarity value, we use a modified Jaccard index as described in [14] and already used for layer decomposition [15]:

Definition 5 **(Normalisation).** *We define the normalised similarity of two n-gram sets NGS_1, NGS_2 as*

$$sim_{NGS}(NGS_1, NGS_2) := \frac{nnsim_{NGS}(NGS_1, NGS_2)}{|NGS_1| + |NGS_2| - nnsim_{NGS}(NGS_1, NGS_2)} \tag{14}$$

We also considered another type of n-grams that encode each vertices' neighbourhood (i.e. its predecessors, successors and itself) similar to [1]. Due to the low average branching factor in SWFs, this type of pattern turned out to be too restrictive and its detailed analysis is omitted here.

5 Evaluation

5.1 Setup

For the evaluation of our novel approach, we make use of the gold standard of similarity ratings utilised in [15]: From a corpus of 1485 Apache Taverna workflows, 24 *query* *workflows* were chosen, each having another 10 workflows associated with it. 15 SWF experts from 6 different institutions were shown a query workflow along with one of its 10 associated workflows and had to decide whether they are *very similar*, *similar*, *related* or *dissimilar*. A fifth option was to choose *unsure*, though these ratings were not further considered in the evaluation. The result were multiple rankings of 10 workflows, all ordered by their similarity to the respective query workflow.

In a last step, a *consensus* ranking was computed from these rankings to aggregate the single experts' ratings. We use this consensus to compare it to the rankings retrieved by our algorithm using the measures introduced in [4]: We represent a consensus ranking and a ranking obtained by running our algorithm as partial orders $\sqsupset_\star$ and $\sqsupset$, respectively, and call a pair of workflows W_1, W_2 *concordant* if it appears in the same order in both rankings, i.e.:

$$C = \{(W_1, W_2) : (W_1 \sqsupset W_2 \wedge W_1 \sqsupset_\star W_2) \vee (W_2 \sqsupset W_1 \wedge W_2 \sqsupset_\star W_1)\} \quad (15)$$

If the order in our predicted ranking differs from the consensus ranking, we call it *discordant*, i.e.:

$$D = \{(W_1, W_2) : (W_1 \sqsupset W_2 \wedge W_2 \sqsupset_\star W_1) \vee (W_2 \sqsupset W_1 \wedge W_1 \sqsupset_\star W_2)\} \quad (16)$$

To compare a ranking obtained using our algorithm to the respective consensus ranking, we count the amount of concordant and discordant pairs and use the definition of ranking *correctness* (eq. (17)) and *completeness* (eq. (18)):

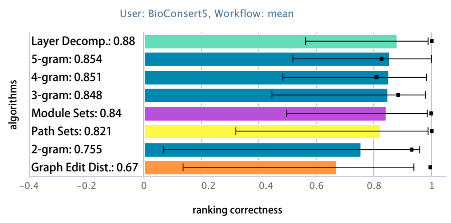
$$CR(\sqsupset, \sqsupset_\star) = \frac{|C| - |D|}{|C| + |D|} \quad (17) \qquad\qquad CP(\sqsupset) = \frac{|C| + |D|}{|\sqsupset_\star|} \quad (18)$$

The value of correctness is in $[-1, 1]$, with -1 indicating a perfectly negative correlation between $\sqsupset_\star$ and $\sqsupset$ as in this case $|C| = 0$. Conversely, $+1$ indicates a perfectly positive correlation. On the other hand, completeness measures the number of pairs whose elements can be distinguished by the experts but not by our algorithm.

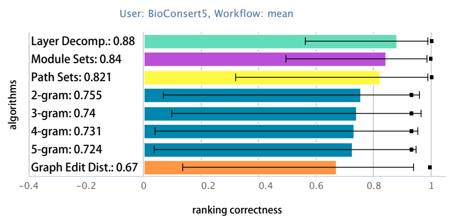
5.2 Results

Figure 2a shows the results: The coloured bars show the algorithms' ranking correctness, whereas the small black squares indicate ranking completeness. The black error bars visualise the standard deviation of ranking correctness. All approaches present in the diagrams use matchings of tasks based on the edit distance of the tasks' labels.

For $n \in \{3, 4, 5\}$, n-grams increasingly outperform path sets in terms of correctness, for $n = 5$ even with a slightly lower variance as can be seen in Figure 2a. Contrariwise, there is an overall decrease in completeness as n grows, making n-grams the only approach with a completeness far below 1. This is due to our definition of an n-gram: As opposed to e.g. path sets, we require both graphs to contain at least one path of length n, whereas in path sets, where graphs are compared by paths starting at a source and ending in a sink without any constraints on the paths' lengths, even a single vertex per graph is sufficient to conduct a similarity assessment. The same applies to module sets, graph edit distance and layer decomposition, as neither of them imposes any requirements on the graphs' size.



(a) Ranking Correctness



(b) Ranking Correctness for cumulative n-grams

Fig. 2: Mean Ranking Correctness for Different Algorithms and Setups

To analyse the impact of this design specificity, we first define *coverage* as the ratio of a graph's vertices, that are part of at least one n-gram. The histogram in Figure 3 shows the share of workflows in our corpus whose coverage is within different ranges for n-grams of size $n = 2..5$. For instance, in 45% of the workflows in our corpus, between 80% and 100% of the vertices are covered by 4-grams, while 48% do not even contain a single 4-gram, as can be seen in the green and light blue bars of p4. Further, only 67% contain at least one 3-gram, which implies that only $\approx 45\%$ of all possible pair-wise comparisons can be conducted using 3-grams. For 4-grams, the share further decreases to $\approx 27\%$.

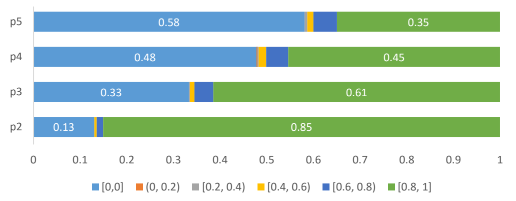


Fig. 3: A histogram depicting the coverage of the graphs in our corpus by n-grams of different sizes.

Observations regarding our corpus of 1483 SWFs yield that the median number of (a) vertices is 5, (b) edges is 4 and (c) average branches per vertex is 0.8, i.e. the majority of SWF graphs are comparably small, prohibiting a further increase in correctness by the use of larger n-grams. This is the major problem that is inherent to our approach, as we have to *reject* a comparison whenever at least one of the workflows that are to be compared does not contain an n-gram of the required size, leading to a lower completeness.

An obvious remedy could be to let the value of n depend on the involved graphs' size, instead of requiring both graphs to contain at least one path of a fixed length n. To this end, we define the maximum n that a graph contains at least one n-gram for as:

$$maxn(G) = \max\{n \in \mathbb{N} : NGS_n(G) \neq \emptyset\} \tag{19}$$

Given workflow graphs G_1, G_2, we let $n_1 = maxn(G_1), n_2 = maxn(G_2)$ and compute the similarity as the (weighted) sum of similarity scores for all $2 \leq n \leq n_\star$ with $n_\star = \min(n_1, n_2)$:

$$csim(G_1, G_2) := \frac{2}{n_\star(n_\star + 1) - 2} \sum_{i=2}^{n_\star} i \cdot sim_{NGS}(NGS_i(G_1), NGS_i(G_2)) \tag{20}$$

In eq. (20), the weighted sum is normalised by dividing it by the sum of all integers from 2 to $n_\star$, with $\frac{2}{n_\star(n_\star+1)-2}$ being derived from the Gaussian summation formula.

Unfortunately, the great increase in completeness (≈ 0.92 for all $n \in \{2,3,4,5\}$) comes at the cost of bad correctness and variance as can be seen in Figure 2b. In fact, as noted in [4], there exists a negative correlation between correctness and completeness, that is due to the definitions of the evaluation measures, i.e. the correctness can be increased at the cost of more rejections and, in turn, a lower completeness. As a result, our initial approach performs better as it simply rejects workflow pairs it can not compare, instead of falling back to a lower n which would yield a less reliable similarity value.

6 Conclusion

In this work, we introduced n-grams in the context of SWF comparison as fixed-size local graph patterns that cover a vertices context to different extents, depending on the value of n. To compare two SWFs by their sets of n-grams, we adapted the similarity measure from [14,15] and were able to show that n-grams for $n \in \{3,4,5\}$ outperform path and module sets and are close to layer decomposition in terms of the mean correctness. However, we still see potential for various optimisations based on these first results on n-gram based SWF similarity.

First of all, we see the evaluation of other local graph patterns as a possible direction of further research. For instance, another interesting approach could be to compare only data-flow splits or to retrieve similarity values for multiple patterns and to assign a weight to each one. Regarding the approach presented in this work, we propose optimisations specifically dedicated to either correctness or completeness:

Correctness. To further increase correctness, the matching strategy could be adjusted. In particular, a more local matching such as a maximum weight non-crossing matching *within* n-grams combined with a maximum weight matching *across* n-grams should lead to an increase in correctness as the similarity measure would become more sensitive to partly similar n-grams and thus would allow for a more fine-grained similarity assessment.

Also, some modules in an SWF perform trivial tasks with a low specificity for a certain context [14], which is particularly a problem for higher n, which yield smaller n-gram sets. To this end, an optimisation the correctness can be expected to benefit from is *importance projection* introduced in [14], where modules performing trivial tasks are removed from the graph prior to the similarity assessment. As the workflow graphs are possibly reduced in size by the use of importance projection, the n-gram retrieval algorithm (Algorithm 2) would also benefit in terms of run time, as its run time complexity has an exponential correlation with n.

Completeness. The most obvious way to increase the completeness would be to relax the definition of the n-gram similarity measure from Definition 2 to instead compare n-grams of possibly different sizes. Formally, we compute the n-gram set for some $n_1 = \min(maxn(G_1), n)$ (with $maxn$ as defined in eq. (19)) for workflow G_1 and some n_2 for G_2 respectively. The overall comparison process becomes similar to the path set comparison as presented in Section 3: To preserve the execution order within the compared n-grams, a maximum weight non crossing matching is computed within the *n*-grams and the overall similarity is the maximum weight matching of both workflows' *n*-gram sets as described before regarding the correctness.

Altogether, we have shown *n*-grams to be a powerful technique for SWF similarity assessment, that still leaves room for further refinements to take on in future work.

References

1. Augsten, N., Böhlen, M., Gamper, J.: Approximate matching of hierarchical data using pq-grams. In: PVLDB 2005. pp. 301–312. VLDB Endowment (2005)
2. Bux, M., Leser, U.: Parallelization in scientific workflow management systems. arXiv preprint arXiv:1303.7195 (2013)
3. Chen, Y., Zhao, X., Xiao, C., Zhang, W., Tang, J.: Efficient and scalable graph similarity joins in mapreduce. TheScientificWorldJournal 2014, 749028 (2014)
4. Cheng, W., Rademaker, M., De Baets, B., Hüllermeier, E.: Predicting partial orders: ranking with abstention. In: ECML-PKDD. pp. 215–230. Springer (2010)
5. Kiepuszewski, B., ter Hofstede, A., van der Aalst, W.: Fundamentals of control flow in workflows. Acta Informatica 39(3), 143–209 (2003)
6. Korf, R.E.: Depth-first iterative-deepening: An optimal admissible tree search. Artificial intelligence 27(1), 97–109 (1985)
7. Li, P.: Copasi time simulation of sbml model. `http://www.myexperiment.org/workflows/1202/versions/1/previews/full`, accessed: 05.03.2016
8. Malucelli, F., Ottmann, T., Pretolani, D.: Efficient labelling algorithms for the maximum noncrossing matching problem. Discrete Applied Mathematics 47(2), 175–179 (1993)
9. Ostermann, S., Prodan, R., Fahringer, T., Iosup, R., Epema, D.: On the characteristics of grid workflows. In: CoreGRID Symposium-Euro-Par. vol. 2008, pp. 1–12 (2008)
10. Prodan, R., Fahringer, T.: Grid Computing, vol. 4340. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
11. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. Image and Vision Computing 27(7), 950–959 (2009)
12. Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. Communications of the ACM 18(11), 613–620 (1975)
13. Santos, E., Lins, L., Ahrens, J.P., Freire, J., Silva, C.T.: A first study on clustering collections of workflow graphs. In: Provenance and Annotation of Data and Processes, pp. 160–173. Springer (2008)
14. Starlinger, J., Brancotte, B., Cohen-Boulakia, S., Leser, U.: Similarity search for scientific workflows. Proceedings of the VLDB Endowment 7(12), 1143–1154 (2014)
15. Starlinger, J., Cohen-Boulakia, S., Khanna, S., Davidson, S.B., Leser, U.: Layer decomposition: An effective structure-based approach for scientific workflow similarity. In: 10th International Conference on e-Science. vol. 1, pp. 169–176. IEEE (2014)
16. Talia, D.: Workflow systems for science: concepts and tools. ISRN Software Engineering 2013 (2013)
17. Wombacher, A.: Evaluation of technical measures for workflow similarity based on a pilot study. In: On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, pp. 255–272. Springer (2006)
18. Wombacher, A., Li, C.: Alternative approaches for workflow similarity. In: Services Computing (SCC), 2010 IEEE International Conference on. pp. 337–345. IEEE (2010)
19. Zhao, X., Xiao, C., Lin, X., Wang, W.: Efficient graph similarity joins with edit distance constraints. In: 2012 IEEE International Conference on Data Engineering (ICDE 2012). pp. 834–845 (2012)