Detecting common scientific workflow fragments using templates and execution provenance

Daniel Garijo
Ontology Engineering Group
Depto. Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
dgarijo@fi.upm.es

Oscar Corcho
Ontology Engineering Group
Depto. Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
ocorcho@fi.upm.es

Yolanda Gil
Information Sciences Institute and Department of Computer Science
University of Southern California
gil@isi.edu

ABSTRACT

Provenance plays a major role when understanding and reusing the methods applied in a scientific experiment, as it provides a record of inputs, the processes carried out and the use and generation of intermediate and final results. In the specific case of in-silico scientific experiments, a large variety of scientific workflow systems (e.g., Wings, Taverna, Galaxy, Vistrails) have been created to support scientists. All of these systems produce some sort of provenance about the executions of the workflows that encode scientific experiments. However, provenance is normally recorded at a very low level of detail, which complicates the understanding of what happened during execution. In this paper we propose an approach to automatically obtain abstractions from low-level provenance data by finding common workflow fragments on workflow execution provenance and relating them to templates. We have tested our approach with a dataset of workflows published by the Wings workflow system. Our results show that by using these kinds of abstractions we can highlight the most common abstract methods used in the executions of a repository, relating different runs and workflow templates with each other.

Categories and Subject Descriptors

I.2.6 [**Learning**]; I.2 [**Artificial Intelligence**]

General Terms

EXPERIMENTATION

Keywords

Scientific workflow, provenance, abstraction, Wings

1. INTRODUCTION

A scientific workflow can be seen as a digital instrument that allows scientists to encode a scientific experiment in the form of a set of computational or data manipulation steps. Scientific workflows play an important role in the reproducibility and replicability of scientific experiments, as well as in repurposing and reusing results from previous experiments [13]. Given their importance in the research lifecycle, scientific workflows are beginning to be included in scientific publications, together with datasets and other elements used in the context of an experiment. At the same time, repositories of workflows like myExperiment [24], Crowdlabs [19] or Galaxy [10] facilitate workflow publication, exchange and reuse. These repositories currently store thousands[1] [2] of workflows (referred to as workflow templates), which have been uploaded by scientists in many different domains (ranging from life sciences to text analytics or astronomy [8]).

Given that support for reproducibility and justification of the obtained results are some of the main roles of scientific workflows, recording the provenance of workflow executions has become crucial to gain insight about what happened when executing a certain workflow, and consequently to explain what happened in a scientific experiment. Hence workflow systems like Wings [11], Taverna [20], Vistrails [5] or Kepler [18] allow recording and exporting the provenance of workflow executions (according to different provenance models) along with their workflow templates.

These workflow templates and the provenance associated to their executions are used for different purposes: detection of the source of an error in a particular execution, determining workflow similarity among workflows [12] [1], automatic workflow mining for helping in workflow design [17], etc.

However, execution provenance traces are often provided at a too low granularity, what makes the execution difficult to understand (especially for users that were not involved in the workflow development process). As a first step to address this issue, we have previously built a catalog of typical abstractions that generalize the computational steps followed by the workflow (called workflow motifs[3] [8]). Sample workflow motifs that we manually identified in our catalog include data preparation, data cleaning, data moving, data retrieval and workflow overloading. Similarly to earlier work in problem solving methods [22], workflow motifs add a layer of abstraction that generalizes the functionality of each step or set of steps, helping scientists to understand the main functionality of the workflow. Motifs also help relating a

[1]`http://www.myexperiment.org/workflows`
[2]`http://www.crowdlabs.org/vistrails/workflows/`
[3]`http://purl.org/net/wf-motifs`

workflow to other workflows, for instance those in a repository.

This paper describes our work towards the automatic detection of these types of motifs from the provenance traces generated by the executions of a workflow or of a set of workflows in a repository. In particular, we are interested in finding common workflow fragments within a workflow (Internal Macro motif), as well as identifying the most common workflow fragments in a repository (Composite Workflow motif), which are two of the motifs that we identified in previous work [8].

Detecting these motifs has several benefits, which are strongly related to the lifecycle of scientific research:

1. *Workflow/experiment reuse and discovery*, by making explicit the overlapping parts of a workflow with other workflows of the catalog. This is useful for scientists interested in exploring workflows with overlapping fragments of similar functionality.

2. *Workflow/experiment understandability*, by grouping several specific workflow templates or executions within a single abstract workflow fragment, which describes them in a more generic way. This is useful for scientists to find out the different ways of performing an abstract method.

3. *Workflow/experiment design*: by proposing as new workflow templates the most popular fragments obtained from workflow templates or workflow execution provenance traces. This is particularly useful in repositories of complex workflow execution provenance traces with barely any workflow templates to relate them.

A distinctive feature of our approach is that we use semantic representations to infer generalizations of the same template or execution from different workflows. This is achieved by exploiting taxonomies of workflow components used for designing the workflow. We have tested our method with workflow templates and provenance traces of their executions in a text analytics domain [15]. These workflows have been developed and executed with the Wings workflow system [11], and the templates and provenance traces are exposed as Linked Data[4], using the Open Provenance Model for Workflows (OPMW) [9], a model that captures provenance and plans of scientific workflows, aligned with the W3C standard for Provenance[5] (PROV).

This paper is structured as follows. Section 2 introduces the main terminology used in the document, along with the descriptions of the main abstractions we aim to identify in the workflows. Section 3 describes the approach that we follow to detect commonalities between workflows, and how we have used inference in this process. Section 4 discusses the results of our approach, comparing them to a manual detection approach. Section 5 describes the related work on pattern recognition and workflow discovery, and in Section 6 we discuss our planned lines of work in order to improve the current results.

2. WORKFLOW ABSTRACTIONS

This section introduces relevant distinctions about workflows. Section 2.1 introduces the terminology used in our work, using examples from abstractions in a text analytics domain described in [15]. Section 2.2 explains the different types of abstractions we aim to identify based on our work on workflow motifs [8].

[4]http://www.opmw.org/sparql
[5]http://www.w3.org/TR/prov-o/

2.1 Terminology

A *workflow template* connects the steps of the workflow together, its inputs, intermediate results and expected outputs, and defines their types and dependencies. There are two types of workflow templates, as illustrated in Figure 1:

1. *Abstract workflow template*: Template where some steps of the workflow are not bound to a specific component (i.e., a particular implementation of an algorithm, web service, etc.). The abstract workflow template aims to capture the scientific method used in an experiment, independently from a specific tool or service. For example, Figure 1 shows an abstract template on the left, where the "Stemmer" and "TermWeighting" components are methods that can be specialized with different algorithms shown in the component taxonomy of the right of the figure.

2. *Specialized workflow template*: Template in which all the steps are bound to a specific service, tool or code. It is more specific than the abstract workflow template. In Figure 1, two specialized workflow templates are shown in the middle: every step specializes a step of the same abstract template (left) according to the taxonomy of components (on the right of the figure).

A *workflow execution provenance trace* is a structured log of the workflow execution results. It contains details on how every intermediate result and output was generated by each of the steps and their dependencies (i.e., the complete provenance of the execution).

We assume for this work that both workflow templates and execution provenance traces can be represented as directed acyclic graphs (DAGs), which is the most common representation in data-oriented scientific workflows. Therefore templates with conditionals and loops are out of the scope of this paper.

2.2 Types of workflow abstractions

Catalogs of patterns (or of problem solving methods) relevant for the creation of complex methods have been extensively studied in the literature for a large number of scientific and non-scientific domains [25, 8, 22]. Given a set of workflow templates and/or provenance execution traces we aim to detect the following two types of patterns:

1. *Internal Macros*: This kind of abstraction refers to groups of steps in a workflow that correspond to repetitive patterns of combined tasks. An example can be seen in Figure 2, showing a workflow for document classification. Part of the branches of the workflow perform the same tasks (StopWords, SmallWords, Stemmer, TF_IDF, Multi2Single, FormatArff), so we can consider them as an internal macro. Although this pattern might be easy to spot manually in simpler templates, it gets increasingly hard to detect when workflows grow in size.

2. *Composite workflows*: Abstraction referring to a workflow composed by one or more sub-workflows. In this

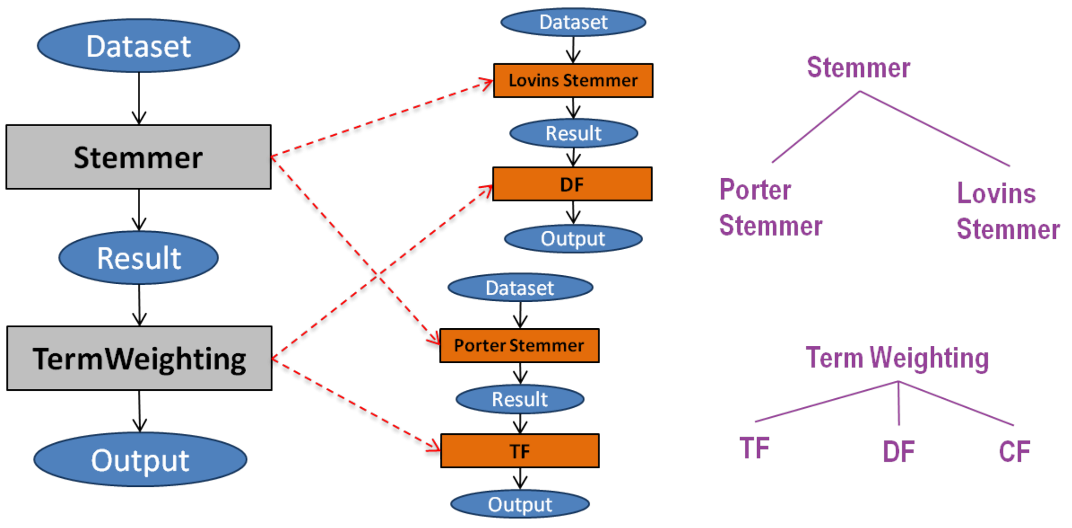

Figure 1: Example of an abstract workflow template (left), two different specialized workflow templates (center) and the taxonomy of workflow components (right). Computational steps are represented in rectangles, while inputs, intermediate results and outputs are represented with ovals.

work we have expanded the original definition in [8], considering under this category those workflows with overlapping fragments. Figure 3 shows an example of a composite workflow, where a template for stemming a document appears in another template for feature selection. Note that the Stemmer step is specialized in the larger workflow, so it is not an exact match.

3. AUTOMATIC DETECTION AND GENERALIZATION OF WORKFLOW ABSTRACTIONS

Given a dataset that contains both workflow templates and workflow execution provenance traces, our goal is to detect Internal Macros and Composite Workflows. We face two major challenges in our work:

1. *Detection of common workflow fragments in the workflow dataset.* It is important to note that we are not interested in retrieving just the largest common sub-workflows, but also the most frequent smaller workflows.

2. *Generalization of workflow fragments to derive abstract workflow templates.* For instance, two different workflow templates may be in fact specializations of the same abstract workflow template. By noting explicitly their common abstract workflow fragments, we make them comparable and easier to understand (both are alternative implementations of the same method).

We describe next how we have approached each of these of challenges.

3.1 Common workflow fragment detection

If we consider a workflow as a dataflow graph, we can use graph algorithms to detect common fragments as common subgraphs. We represent the workflow templates and the workflow execution provenance traces as labeled graphs. Nodes are labeled with the types of the workflow components of the taxonomy (e.g., TF or LovinsStemmer in Figure 1) and the types of the data being used and produced in the workflow (e.g., Dataset in Figure 1); while the edges are labeled with the dependencies of the workflow (*usage* when a computational process consumes an input and *generation* when a computational step produces an output). Our goal is finding the maximum overlapping subgraphs between N different graphs, where N corresponds to the number of workflows or execution provenance traces available in the dataset. Since this problem can be reduced to the subgraph isomorphism problem between two graphs, its complexity is NP-Complete [7].

Trying to make a naive comparison of workflows in a repository (e.g., by comparing them in an all-against-all manner) would lead to inefficient and limited solutions for the problem that we are trying to solve, given its complexity. To address this, we use the SUBDUE algorithm [16], which allows learning the most relevant context-free grammar from a set of labeled graphs. This grammar highlights the most frequent structures found in the graph and evaluates each candidate structure according to the way it compresses the overall collection graph. SUBDUE proposes to use two different metrics for compressing the graph with the production rules of the grammar:

1. *The Minimum Description Length (MDL)*, where the best structure is the one that minimizes the description length of the entire data set (i.e., the bits needed for its encoding) [6].

2. *The size of the graph*, which aims at finding the structure that best reduces the size of the overall collection graph.

We apply SUBDUE independently for each of the workflows to capture Internal Macros, and taking as input the whole collection to detect Composite Workflows. A filtering step is then applied to the results of SUBDUE in order to remove the non complex and repeated structures.

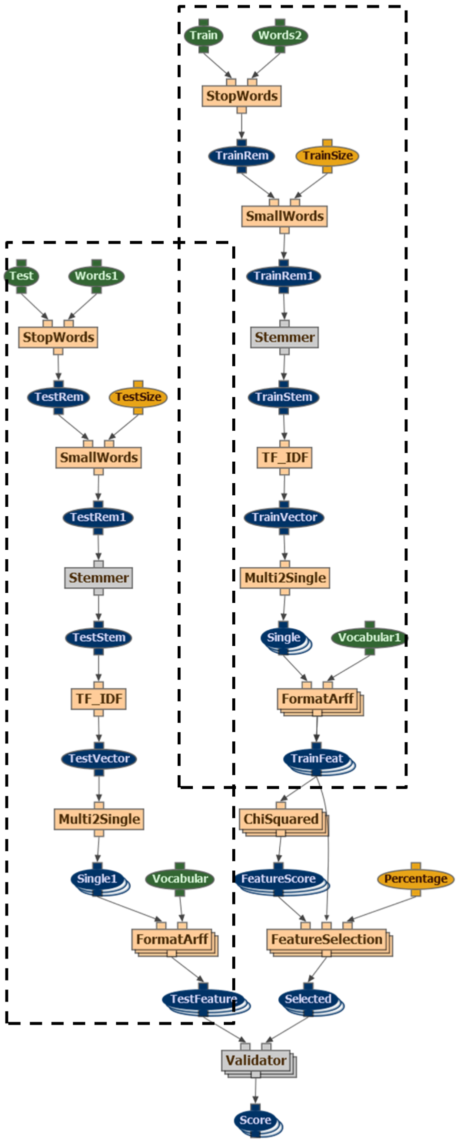

Figure 2: Example of an internal macro within a workflow for feature selection. Two branches have the same sequence of steps.

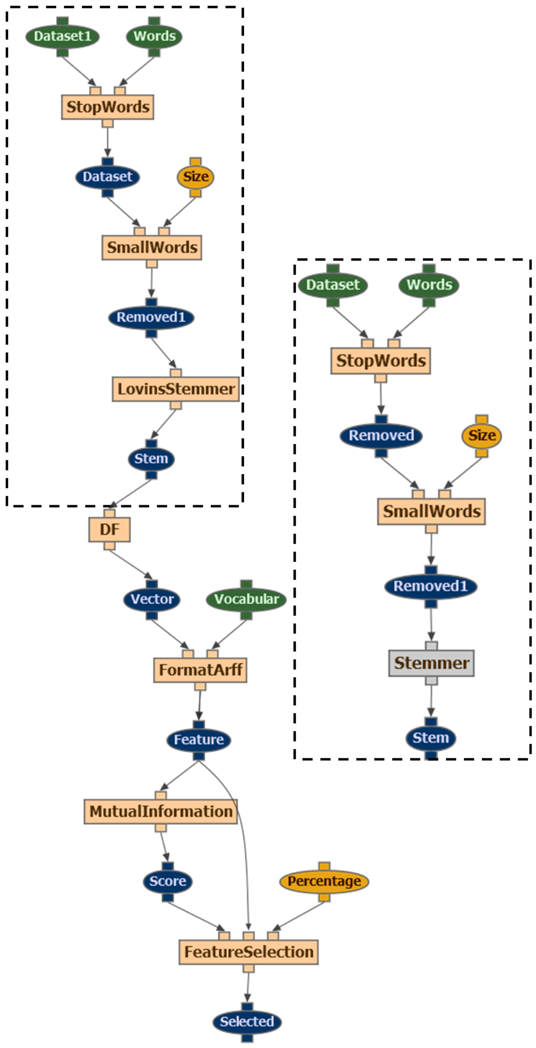

Figure 3: Example of a composite workflow. An abstract workflow template used for stemming (right) is part of another that performs a feature selection of the input data (left).

3.2 Workflow template and workflow execution provenance trace generalization

A key feature of our approach is that we can generalize workflow templates and workflow execution provenance traces in order to derive abstract workflow fragments. The set of abstract workflow fragments is based on a taxonomy of components that can be associated to a catalog of workflow components. An example of this type of abstraction can be seen in Figure 1, where starting from any of the specialized templates in the center plus the taxonomy shown at the right of the figure we can generalize the specialized templates to the abstract template on the left.

This generalization requires an additional step of the input graphs before executing the SUBDUE algorithm. The generalization step replaces the types of the nodes of a given workflow template or workflow execution provenance trace with their appropriate superclass in the taxonomy of components. We abstract all the workflow steps to the most generic class in the taxonomy of workflow components, ensuring that maximum abstraction is provided. The level of abstraction can be changed by modifying the taxonomy (for example, by removing a class if considered too generic).

4. RESULTS

This section presents the results of our analysis, which aim at answering whether the application of our approach allows obtaining the same type of fragments that would be obtained by manual inspection or not. The experiment setup is presented in Section 4.1, while the individual analysis for each type of abstraction is done in Section 4.2 for Internal Macros and Section 4.3 for Composite Workflows.

4.1 Workflow datasets

Table 1: Results for detecting Internal Macros. The analysis is performed on execution provenance traces and templates without applying generalization (no gener) and applying it (gener)

		Executions		Templates		Manual Analysis
		No gener	Gener	No Gener	Gener	
Fragments found	MDL	22	39	15	24	
	size	22	35	15	24	3
Irreducible fragments	MDL	17	31	11	15	
	size	17	26	11	15	3
Multi-step fragments	MDL	8	8	9	9	
	size	8	8	9	9	3
Filtered multi-step fragments	MDL	**2**	**2**	**3**	**3**	
	size	**2**	**2**	**3**	**3**	**3**

Table 2: Results for detecting Composite Workflows. The analysis is performed on execution provenance traces and templates without applying generalization (no gener) and applying it (gener)

		Executions		Templates		Manual Analysis
		No gener	Gener	No Gener	Gener	
Fragments found	MDL	11	13	18	18	
	size	30	22	27	26	9
Irreducible fragments	MDL	8	10	10	14	
	size	21	15	19	19	7
Multi-step fragments	MDL	5	6	7	7	
	size	7	10	6	7	4
Filtered multi-step fragments	MDL	**5**	**6**	**7**	**7**	
	size	**7**	**10**	**6**	**7**	**4**
Fragments found automatically and manually	MDL	3	4	7	5	
	size	3	4	7	5	9
Occurrences of all fragments	MDL	43	42	59	49	
	size	111	78	92	84	22
Occurrences of multi-step fragments	MDL	24	33	27	25	
	size	28	45	25	30	17
Occurrences of filtered multi-step fragments	MDL	**24**	**33**	**27**	**25**	
	size	**28**	**45**	**25**	**30**	**17**

We have selected a dataset that contains 22 workflow templates specified using the Wings workflow system[6] [11]. We also use a dataset of 30 workflow execution provenance traces obtained from the executions of the 22 workflow templates and annotated according to the Open Provenance Model for Workflows (OPMW) [9]. Both datasets are in the domain of text analytics.

This specific domain and datasets have been selected for our experiment for four reasons:

- *They contain abstract and specific templates in the same domain*, which gives many alternatives for abstractions of executions.
- *They contain several workflow executions that correspond to the same template*, allowing to detect different common fragments between workflow templates and workflow execution provenance traces.
- *They contain workflow execution provenance traces that had execution errors when running the workflow*, which provides the means to test our capabilities to deal with incomplete provenance graphs.
- *They have been manually analyzed* (as described in [8]), so as to identify the Internal Macros and Composite Components in it. These annotations are used in our evaluation to assess the applicability and goodness of our approach in this domain.

It is important to note that, as aforementioned, all workflow execution provenance traces and workflow templates are annotated with OPMW, which is a model for representing scientific processes and their execution provenance. The repository from which the datasets have been used is available online[7] following Linked Data principles [2]. OPMW extends the Open Provenance Model (OPM) [21] and the recent W3C Standard for Provenance PROV[8] in order to be interoperable with other representations of provenance. In OPMW, all the inputs, intermediate results and outputs of a given execution are *WorkflowExecutionArtifacts*, while each step executed in the workflow is a *WorkflowExecutionProcess. WorkflowExecutionArtifacts* are generated by *WorkflowExecutionProcesses* and *WorkflowExecutionProcesses* use other *WorkflowExecutionArtifacts.* This way the basic provenance for each execution is captured and preserved, being stored as a DAG. A similar approach is provided to link the data dependencies of the workflow templates.

4.2 Internal macro results

[6]`http://www.wings-workflows.org/`

[7]`http://www.opmw.org/sparql`

[8]`http://www.w3.org/TR/prov-o/`

Table 1 shows a summary and comparison of *the workflow fragments found by our automated approach* with respect to those found by the manual annotation. The analysis was performed on workflow execution provenance traces and workflow templates independently using both of the metrics described in Section 3.1 (MDL and Size). Generalization was applied to the workflows, so as to determine whether more fragments could be found. For each type of analysis, we show several intermediate results: *total number of fragments found*, the *irreducible fragments* (i.e., fragments that are not composed of simpler fragments) and the *number of multi-step fragments* (i.e., fragments that combine at least two of either processing steps or irreducible fragments with a processing step). The *number of filtered multi-step fragments* (i.e., multi-step fragments that already include other smaller fragments with the same number of occurrences as the larger fragment) is highlighted in Table 1 because it represents the final workflow fragments found.

Our goal is to maximize the number of filtered multi-step fragments, since regular multi-step fragments may be discarded. For example, Figure 2 shows a multi-step fragment composed of six steps (StopWords, SmallWords, Stemmer, TF_IDF, Multi2Single and FormatArff) and also a smaller multi-step fragment with just SmallWords and StopWords (included in the larger fragment). The filtering step discards the smaller fragment and keeps the larger one because they appear with the same frequency in the workflow (twice). If the workflow had a third branch with a StopWords step and a SmallWords step, the number of occurrences of the fragment would be three instead of two and therefore it would be considered a filtered multi-step fragment.

The total number of multi-step fragments is higher than the filtered multi-step fragments due to the way the algorithm operates. SUBDUE takes the input graph and expands each of the nodes in every iteration, returning the best fragment compressing the graph according to the metrics described in Section 3.1. Therefore the most general fragment is returned in the latest iterations of the algorithm, using the fragments detected previously. In contrast, in the manual annotation only the most general fragment is considered.

The best results are obtained when running the algorithm to find multi-step fragments and filtering them. These results match almost perfectly the ones found with the manual analysis. Most of the multi-step fragments of the manual analysis have been found with the application of our approach (2 out of 3 filtered multi-step fragments in the workflow execution provenance traces, 3 out of 3 filtered multi-step fragments in the workflow templates). The number of internal macros detected in the workflow execution provenance traces is lower because some executions had failed, and hence the provenance graph corresponding to that part of the workflow was missing in the execution provenance traces.

Another interesting fact is the high number of fragments that are *discarded* from those obtained by SUBDUE (i.e., not filtered multi-step fragments), with a maximum of 31 out of 39 in executions with generalization. These fragments usually contain a single processing step, so they are not relevant for our results. Their presence is motivated in the executions by the instantiation of collections. An example can be seen in Figure 2, where the Multi2Single step produces a collection of Single1 results that are consumed in parallel by a collection of FormatArff steps. In templates, the number of occurrences of discarded fragments is high when a single component is shared among many templates (with different specializations of the same abstract method).

4.3 Composite Workflow Results

Regarding the Composite Workflow detection, Table 2 shows the results obtained after the execution of the algorithm. The first four rows of the Table measure the *number of fragments found*, the *irreducible fragments*, the *multi-step fragments* and the *filtered multi-step fragments* (as in Table 1). It also shows the *fragments found automatically and manually* (i.e., number of fragments that are equal to any of the fragments identified in the manual annotation). The rest of the rows measure the *occurrences of the detected fragments* (in order to know how many times has the detected fragments been found in the workflow dataset), the *occurrences of the multi-step fragments* and the *occurrences of filtered multi-step fragments* in the workflow dataset. As we did with the previous analysis, the algorithm was run with different metrics (MDL and Size) and with generalized and non-generalized workflows. The filtered multi-step fragments are highlighted in Table 2.

The fragments found automatically often overlap with the ones identified manually (with a minimum of 3 and a maximum of 7 overlapping fragments over executions and templates). The occurrences of the filtered multi-step fragments obtained by the automatic results are higher than the number of fragments identified manually (ranging from 25 to a maximum of 45 versus 17). This indicates that the fragments found automatically *highlight better the most frequent subworkflows and provide better coverage of the dataset* than those detected manually.

Manual and automatic analyses show different results because their scope is slightly different. On one hand, the manual analysis identifies the subworkflow relationships between different workflows in the dataset, detecting whether a workflow is included in other workflows or not. On the other hand, the automatic analysis aims at finding the most common fragments between the workflows of the dataset, including subworkflows.

Although most of the results obtained by the algorithm share the main core of common workflow fragments, the *best results are found in the workflow execution provenance traces with generalization and using the size metric* (up to 10 multi-step fragments with 45 occurrences in the workflow dataset). The explanation of this result is related to the number of executions that specialize abstract workflow templates in different ways: when applying generalization, the fragments found are likely to be grouped around the abstract template with most executions. In contrast, generalization does not improve significantly the number of multi-step fragments in templates, as the amount of specialized templates available in the dataset is not high (in most cases just the abstract templates are available). Similarly to what happened in the Internal Macro analysis, the results of the executions differ from those obtained by the templates because of collections and incomplete executions. Therefore, the workflow fragments detected in the executions represent the most common portions of workflows executed successfully.

A limitation of our approach is regarding the selection of overlapping candidate fragments. An example can be seen in Figure 4. Two different workflows, the one in the left side of Figure 4 (1) and the one in the right side of the Figure (2), overlap partially within another workflow represented in the center. The fragment that better reduces its graph is (1), but reducing the graph with it leads to avoid the full detection of (2) (instead, the fragment will be just the ”C” and ”output” resources). This is not ideal, since it reduces the number of filtered multi-step fragments we are looking for (by avoiding the full detection of (2), we may be discarding a valid fragment).

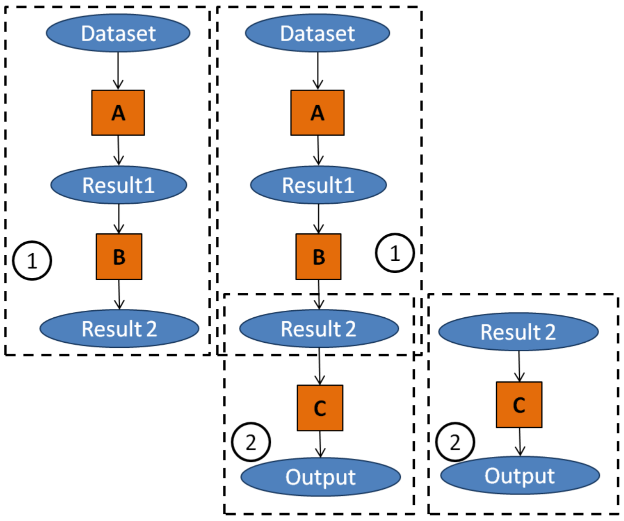

Figure 4: Example of 2 conflicting candidate fragments in three different workflows. The fragment that best compresses the overall graph is (1), leading to (2) not being detected properly.

All the input, results and logs obtained from the execution of our approach can be found online[9].

5. RELATED WORK

Finding commonalities among fragments of workflows is related to the workflow discovery and graph-based comparisons problems. Goderis and colleagues [12] apply sub-graph isomorphism techniques combined with semantic technologies for creating a ranking to find the similarities between a repository of workflow and an input workflow. Similarly, in [1], Bergmann and Gil use graph matching techniques and semantic annotations to find similarities between a template and a repository of existing workflows. Our work is different from these approaches in that we are not interested on the relation of singular templates with other templates of the repository, but common fragments among all of them.

Other work approaches this problem from data mining or case-based reasoning perspectives. Leake and Kendall-Morwick [17] propose to use case-based reasoning over provenance execution traces in order to help users with suggestions for the creation of new workflows. The suggestions are created detecting similarities of the new workflow with previous existing ones, which is another form of sub-graph isomorphism. Yaman et al [26] also propose to mine existing provenance executions, combining the control flow and dataflow of existing workflows in order to approximate to a target workflow. This work is integrated within the POIROT framework [4], used to learn complex hierarchical task models from user-generated traces. The main difference with respect to our work is that we aim at making explicit the relation among the workflows in the dataset rather than learn or suggest similar workflows to a given one. However, while our purpose is not to learn the main method derived from the workflow dataset, the workflow fragments that we detect could be suggested as new templates for users.

Finally, another area of related work is the automatic detection of Problem Solving Methods (PSMs), which describe the reasoning process to achieve the goal of a task in an implementation and domain-independent manner. Gómez-Pérez and Corcho [14] used graph matching techniques to find specific PSMs in provenance logs. This is related to our work since we also look for fragments in a workflow dataset, but in our case the fragments are not defined beforehand, as it is the case for that work. These common fragments could be seen as automatically detected PSMs, since they encode the common method shared among several workflows for achieving a subgoal.

6. CONCLUSIONS AND FUTURE WORK

In this paper we have described an approach for the automatic detection of the most common workflow fragments among a scientific workflow dataset, using as input a repository of workflows that can be either workflow templates or workflow execution provenance traces. We have focused on two types of workflow fragments: those that occur within a workflow (Internal Macros) and those that occur among several workflows (Composite Workflows). We have shown that our proposed approach is able to detect filtered complex fragments successfully and generalize fragments from workflows and provenance traces of their executions.

Our approach uses SUBDUE algorithm, which derives the most relevant context-free grammar of a given graph. Our results demonstrate that this approach finds the workflow fragments with most occurrences, improving the results of manual sub-workflow detection within the fragments of the dataset.

We are currently pursuing different lines of work in order to improve our approach. First, we are analyzing how to detect other relevant abstractions regarding the relations between workflows, like *workflow overloading* (i.e., detection of workflows with the same processing steps but different datatypes). Second, we are looking to extend our approach by adding extra data preparation steps to simplify the graph, or even transformations [3] for reducing the search space. Some of these improvements could help to overcome the overlapping limitation exposed in Section 4. We are also planning to test our approach with traces and templates from other systems with existing taxonomies of components, such as GenePattern [23] or Galaxy [10]; and from systems without them, such as Taverna [20] or Vistrails [5]. Finally, we plan to explore workflows with additional constructs such as conditionals and iterations. We have been using SUBDUE for DAGs, but the algorithm is capable of discovering any fragment as long as it can be encoded in a labeled graph (including cyclic graphs).

7. ACKNOWLEDGMENTS

This research was supported in part by the Wf4Ever Eu-

[9]`http://www.oeg-upm.net/files/dgarijo/kcap2013Eval`

ropean project (FP7-270192), an FPU grant (Formación de Profesorado Universitario) from the Spanish Science and Innovation Ministry (MICINN), the MyBigData project from the Spanish Science and Innovation Ministry (TIN2010-17060) and the US Air Force Office of Scientific Research with grant number FA9550-11-1-0104.

8. REFERENCES

[1] R. Bergmann and Y. Gil. Similarity assessment and efficient retrieval of semantic workflows. *To appear in the Information Systems Journal*, 2012.

[2] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.

[3] S. C. Boulakia, C. Froidevaux, and J. Chen. Scientific workflow rewriting while preserving provenance. In *8th IEEE International Conference on eScience 2012*, pages 1–9, Chicago, 2012. IEEE Computer Society Press, USA.

[4] M. H. Burstein, R. Laddaga, D. D. McDonald, M. T. Cox, B. Benyo, P. Robertson, T. S. Hussain, M. Brinn, and D. V. McDermott. Poirot - integrated learning of web service procedures. In *AAAI*, pages 1274–1279, 2008.

[5] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. Vistrails: Visualization meets data management. In *ACM SIGMOD*, pages 745–747. ACM Press, 2006.

[6] D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[7] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[8] D. Garijo, P. Alper, K. Belhajjame, O. Corcho, Y. Gil, and C. Goble. Common motifs in scientific workflows: An empirical analysis. In *8th IEEE International Conference on eScience 2012*, Chicago, 2012. IEEE Computer Society Press, USA.

[9] D. Garijo and Y. Gil. A new approach for publishing workflows: Abstractions, standards, and linked data. In *Proceedings of the 6th Workshop on Workflows in support of large-scale science*, pages 47–56, Seattle, 2011. ACM.

[10] B. Giardine et al. Galaxy: A platform for interactive large-scale genome analysis. *Genome Research*, 15(10):1451–1455, Oct 2005.

[11] Y. Gil, V. Ratnakar, J. Kim, P. A. González-Calero, P. T. Groth, J. Moody, and E. Deelman. Wings: Intelligent workflow-based design of computational experiments. *IEEE Intelligent Systems*, 26(1):62–72, 2011.

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

[13] A. Goderis, U. Sattler, P. W. Lord, and C. A. Goble. Seven bottlenecks to workflow reuse and repurposing. In *International Semantic Web Conference*, pages 323–337. Springer, 2005.

[14] J. M. Gómez-Pérez and O. Corcho. Problem-solving methods for understanding process executions. *Computing in Science and Engineering*, 10(3):47–52, May 2008.

[15] M. Hauder, Y. Gil, and Y. Liu. A framework for efficient data analytics through automatic configuration and customization of scientific workflows. In *Proceedings of the 2011 IEEE Seventh International Conference on eScience*, ESCIENCE '11, pages 379–386, Washington, DC, USA, 2011. IEEE Computer Society.

[16] L. B. Holder, D. J. Cook, and S. Djoko. Substructure Discovery in the SUBDUE System. *AAAI Workshop on Knowledge Discovery*, pages 169–180, 1994.

[17] D. Leake and J. Kendall-Morwick. Towards case-based support for e-science workflow generation by mining provenance. In *Proceedings of the 9th European conference on Advances in Case-Based Reasoning*, ECCBR '08, pages 269–283, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[19] P. Mates, E. Santos, J. Freire, and C. T. Silva. Crowdlabs: Social analysis and visualization for the sciences. In *23rd International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 555–564. Springer, 2011.

[20] P. Missier, S. Soiland-Reyes, S. Owen, W. Tan, A. Nenadic, I. Dunlop, A. Williams, T. Oinn, and C. Goble. Taverna, reloaded. In *22nd International Conference on Scientific and Statistical Database Management (SSDBM)*, Heidelberg, Germany, 2010.

[21] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers, B. Plale, Y. Simmhan, E. Stephan, and J. Van den Bussche. The Open Provenance Model core specification (v1.1). *Future Generation Computer Systems*, July 2010.

[22] A. G. Pérez and R. Benjamins. Applications of ontologies and problem-solving methods. *AI Magazine*, 20(1), 1999.

[23] M. Reich, T. Liefeld, J. Gould, J. Lerner, P. Tamayo, and J. P. Mesirov. Genepattern 2.0. *Nature Genetics*, 38:500 – 501, 2006.

[24] D. D. Roure, C. A. Goble, and R. Stevens. The design and realisation of the myExperiment virtual research environment for social sharing of workflows. *Future Generation Comp. Syst.*, 25(5):561–567, 2009.

[25] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[26] F. Yaman, T. Oates, and M. Burstein. A context driven approach for workflow mining. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1798–1803, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.