Proceedings of the Fifth International Conference on Knowledge Capture (K-CAP), Redondo Beach, CA, September 1-4, 2009.

Workflow Matching Using Semantic Metadata

Yolanda Gil[1], Jihie Kim[1], Gonzalo Florez[2], Varun Ratnakar[1], Pedro A. González-Calero[2]

[1] Information Sciences Institute
University of Southern California
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292, USA
{gil, jihie, varunr}@isi.edu

[2] Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, Spain
gflorez@fdi.ucm.es, pedro@sip.ucm.es

ABSTRACT
Workflows are becoming an increasingly more common paradigm to manage scientific analyses. As workflow repositories start to emerge, workflow retrieval and discovery becomes a challenge. Studies have shown that scientists wish to discover workflows given properties of workflow data inputs, intermediate data products, and data results. However, workflows typically lack this information when contributed to a repository. Our work addresses this issue by augmenting workflow descriptions with constraints derived from properties about the workflow components used to process data as well as the data itself. An important feature of our approach is that it assumes that component and data properties are obtained from catalogs that are external to the workflow system, consistent with current architectures for computational science.

Categories and Subject Descriptors
I.2.11 Distributed Artificial Intelligence; I.2.8 Problem Solving, Control Methods, and Search; H.4 Information Systems Applications; I.2.4 Knowledge Representation Formalisms and Methods.

General Terms
Algorithms, Languages.

Keywords
scientific workflows, workflow matching, workflow discovery, workflow retrieval, workflow catalogs, semantic workflows, semantic matchmaking.

INTRODUCTION
Workflows represent complex applications assembled from distributed steps implemented as remote services or remote job submissions [3,19]. Workflows are becoming an increasingly more common paradigm to manage scientific analyses. Workflows represent data analysis routines as workflow *components*. Workflows also contain *links* that express the dataflow among these components and reflect the interdependencies that must be managed during their execution.

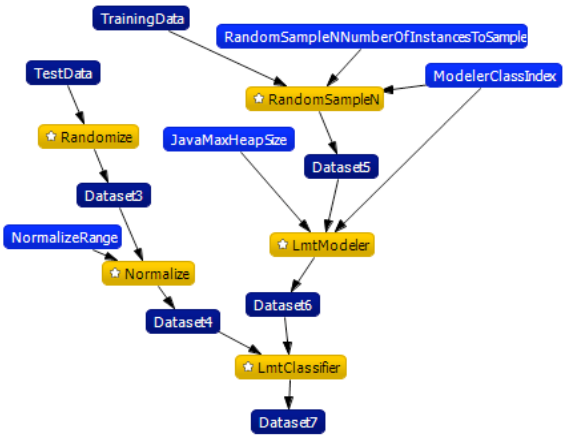
As scientific workflows become more commonplace, workflow repositories are emerging with contributions from a variety of scientists. Provenance systems record the details of the execution of workflows so they can be retrieved later [17,13]. Since workflow executions contain a lot of details that make it harder to reuse them, scientists also share *workflow templates* that describe a general kind of analysis that can be more easily reused [19]. Workflow repositories can also contain best practices for common types of scientific analyses [18].

Workflow matching and discovery from these repositories becomes a challenge. A scientist may need a workflow appropriate to analyze some dataset he or she has, or need a workflow that does a certain kind of analysis, or a workflow fragment that produces a certain type of result. A series of studies regarding the requirements for scientific workflow matching and discovery [5-8] found the need to support:

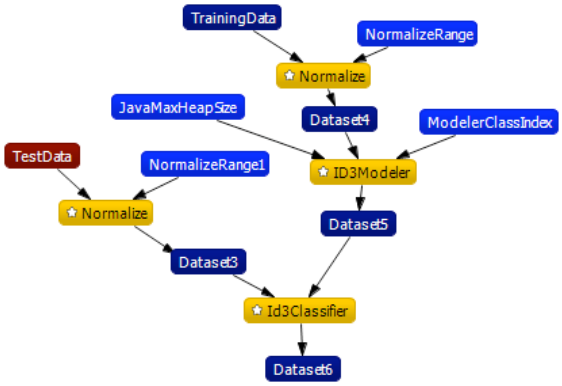
I. Queries based on the types of data used by a workflow,
II. Queries based on the types of intermediate or final data that the workflow produces,
III. Queries specifying ordered data points that must appear in the dataflow,
IV. Queries that specify what components (algorithms) must appear in the workflow and their relative order,
V. Queries that specify properties of the workflow such as authors, creation time, derived variants, or popularity,
VI. Queries that contain any combination of the above.

Recent work on workflow discovery has only investigated retrieval based on component orderings [6], retrieval based on (social) tags and textual descriptions [7], and retrieval of workflow execution traces to be reused during workflow creation [14]. Matching based on types of data has been done for the discovery of individual services or software components [10,15], but not for the more complex structures that workflows represent. Queries I to III above have not been investigated to date and are the most challenging, as they represent data-centered properties of the workflow that are often not specified in the workflow.

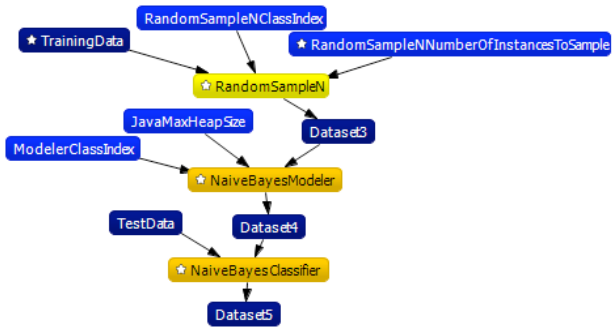
K-CAP'09, September 1-4, 2009, Redondo Beach, California, USA.



T1: Sample the training data to build an LMT model, randomize and normalize the test data and use an LMT classifier.

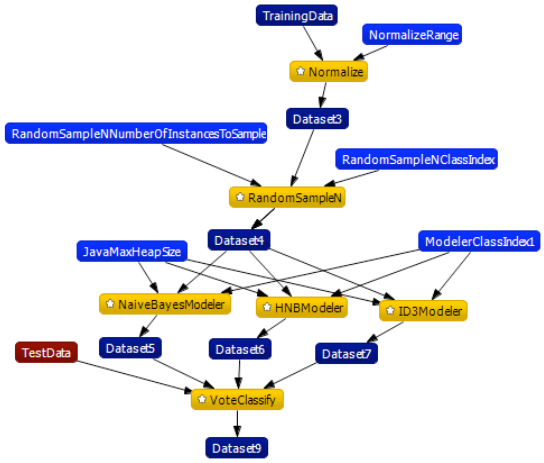


T2: Normalize training and test data, then use ID3 to model and classify the data.



TrainingData	wflow:hasDataBinding	ddlb:weather-nominal-2007-07-31-155627
RandomSampleNNumberOfInstancesToSample	wflow:hasParameterValue	10000^^xsd:int

T3: Use NaiveBayes with sampled discrete weather 155627 for training.



T4: Normalize and sample training data, then use a voting scheme to combine a NaiveBayes model, an HNB model, and an ID3 model.

Figure 1. Example workflow templates in a workflow catalog, showing the diagram and constraints specified by users.

The main contribution of this work is to support workflow retrieval given data-centered queries and their combination with other constraints on components and workflow structure. One important challenge is that workflow catalogs typically specify only a limited amount of information that is insufficient for data-centered queries. That is, although semantic annotations of workflows have been explored in prior work, the presence of any semantic information in the workflow is assumed to be manually provided. However, when users create a workflow they rarely add such information. We believe that many semantic annotations can be extracted from component catalogs that describe individual components reused in different workflows. Our approach is to use such component descriptions to automatically enrich the workflows created by users into semantic workflows that contain inferred properties that are needed for supporting data-centered queries. An important consideration in our work is that scientific applications are developed using data and component catalogs that are independent of workflow catalogs (e.g., www.nvo.org, www.earthsystemgrid.org, cabig.cancer.gov). Therefore, a contribution of our work is that our workflow matching algorithms identify reasoning tasks that are specific to datasets and components, and submit requests to external data and component catalog services to carry them out.

The paper begins with examples of the kinds of queries that our approach can address. After describing our workflow enrichment and matching algorithms, we present our implemented system and performance results on a modest but realistic library of workflows.

MOTIVATING EXAMPLES

We use a small set of queries to illustrate the capabilities of our approach and implemented system. Here, we use machine learning workflows with components from the well-known Weka repository [21] and Irvine datasets [1]. These workflows typically consist of a few data preparation steps, then build a model with training data that is used to classify test data.

Figure 1 shows in detail four workflow templates that could be in a workflow catalog. For each template, we show the dataflow diagram as the creator would specify it by selecting *components* (which correspond to data analysis routines used, shown in yellow) and connecting them. The dataflow graph shows stubs for data, which are called *workflow data variables*, in dark blue. *Parameters* of components are shown in light blue.

TrainingData	dcdom:isDiscrete	true^^xsd:boolean
TrainingData	rdf:type	dcdom:Instance
TrainingData	rdf:type	dcdom:DiscreteInstance
TestData	dcdom:isDiscrete	true^^xsd:boolean
TestData	rdf:type	dcdom:Instance
TestData	rdf:type	dcdom:DiscreteInstance
Dataset9	rdf:type	dcdom:Classification
Dataset7	dcdom:isDiscrete	true^^xsd:boolean
Dataset7	dcdom:isNormalized	true^^xsd:boolean
Dataset7	rdf:type	dcdom:Model
Dataset7	rdf:type	dcdom:DecisionTreeModel
Dataset6	dcdom:isDiscrete	true^^xsd:boolean
Dataset6	dcdom:isNormalized	true^^xsd:boolean
Dataset6	rdf:type	dcdom:Model
Dataset6	rdf:type	dcdom:BayesModel
Dataset5	dcdom:isDiscrete	true^^xsd:boolean
Dataset5	dcdom:isNormalized	true^^xsd:boolean
Dataset5	rdf:type	dcdom:Model
Dataset5	rdf:type	dcdom:BayesModel
Dataset4	dcdom:isDiscrete	true^^xsd:boolean
Dataset4	dcdom:isNormalized	true^^xsd:boolean
Dataset4	dcdom:isSampled	true^^xsd:boolean
Dataset4	rdf:type	dcdom:Instance
Dataset4	rdf:type	dcdom:DiscreteInstance
Dataset3	dcdom:isDiscrete	true^^xsd:boolean
Dataset3	dcdom:isNormalized	true^^xsd:boolean
Dataset3	rdf:type	dcdom:Instance
Dataset3	rdf:type	dcdom:DiscreteInstance

Figure 2. Workflow constraints for T4 after the workflow is augmented with propagated constraints.

When users create workflow templates, they specify workflow components and dataflow links. Users will rarely specify any additional constraints of any of the datasets in the workflow. For example, we expect that a user creating T1 will not specify that the model generated by a logistic model tree (LMT) modeler is of type LMTmodel. In the user's mind, this would fall from the definition of an LMT modeler, and does not need to be specified in each workflow. Therefore, workflows contributed to the catalog will mostly consist of the dataflow graph and have little or no semantics about the datasets involved. In our approach, the original dataflow provided by the workflow creator is augmented with additional inferred constraints based on information coming from the component catalog about how each workflow component behaves. That is, a component catalog would know that a discretizer component does not change the sparseness of a dataset, but will make a continuous dataset into a discrete one. Figure 2 shows examples of such constraints derived by our system as tables of triples of <object property value>, which we call *data object descriptions (DODs)*.

Table 1 shows retrieval results for five diverse example queries. The first column gives a textual description of the query. The second column shows which workflows can be matched if the original workflows are augmented only with type information obtained from a basic component catalog that only represents data type constraints for components. The last column shows the workflows that are retrieved by our system after augmenting the original created workflow with additional inferred properties from a component catalog with richer semantic information about components.

Query	**Matched workflows augmented with types**	**Matched workflows with propagated constraints**
Q1: Find workflows where input data is cpu data, which is continuous and has missing values	T1 T2 T3 T4	- - - -
Q2: Find workflows that generate a classification of Iris data	T1 T2 T3 T4	T1 T2 - T4
Q3: Find workflows that can have cpu-2008-07-09 as input to a classifier (the dataset is continuous and has missing values)	T1 T2 T3 T4	- - - -
Q4: Find workflows that use sampled training data to produce a Bayes model, then use model to generate a classification of Iris test data, which is discrete and has missing values	- - T3 T4	- - - T4
Q5: Find workflows that normalize labor data to create a decision tree model (labor data is continuous and has missing values)	T1 T2 - T4	- - - -

Table 1. Example queries showing workflows retrieved.

T1 illustrates the difference. It uses the LMT algorithm, a decision tree approach that requires that the data does not have missing values (i.e., it is not sparse). When T1 is augmented with the types that are input and output for each component, it would include a constraint that the model created by the LMT modeler is a decision tree model (DT model). Therefore, it would not be retrieved for query Q4, which asks for a Bayes model, but it would be retrieved for Q1, Q2, Q3, and Q5. But the LMT algorithm also requires that the data has no missing values. Therefore, the input to the sampler cannot have missing values in turn. When the system propagates this requirement through the sampler component, it can detect that T1 is not appropriate for Q1, Q4, or Q5 because the input datasets would be sparse. In essence, T1 is only really appropriate to retrieve for Q2. If no constraints are propagated, the system would also retrieve T1 for Q1, Q3, and Q5 resulting in lower precision.

An analogous situation occurs with workflow template T2. It uses the ID3 algorithm, which can only take discrete values, and normalizes the datasets before modeling and classifying. When T2 is augmented with the input and output types for each component, it includes a constraint that the model created by the ID3 modeler is a decision tree. Therefore, it would not be retrieved for query Q4, which asks for a Bayes model, but it would be retrieved for Q1, Q2, Q3, and Q5. When the system propagates the constraint that input datasets cannot have continuous values, it can detect that T2 is not appropriate for Q1, Q3, or Q5 because the query datasets are continuous.

Augmented workflows lead to higher precision with no loss in recall. Higher precision is very important for workflow discovery, since the size of repositories can become very large. For example, the Taverna bioinformatics workflow system includes more than 3,000 services as of 2006 [11].

Construct	Type	Use
<?c1 precedes ?c2>	SP	component ?c1 appears upstream in the dataflow from ?c2
<?d1 datapointPrecedes ?d2>	SP	data variable ?d1 appears upstream in the dataflow from data variable ?d2
<?d1 datapointImmediatelyPrecedes ?d2>	SP	data variable?d1 is input to a component that outputs data variable ?d2
<?c hasInputData ?d>	SP	dataset identifier ?d is an input to the component ?c
<?c Subclass-of ?t>	CP	component ?c is a subclass of component type ?t
<?d hasDataBinding ?i>	DP	data variable ?d in the workflow is bound to dataset identifier ?i
<?d canBeBound ?i>	DP	data variable ?d in the workflow could potentially be bound to dataset identifier ?i
<?d hasType ?t>	DP	Data variable ?d must be of type ?t
<?d ?p ?v>	DP	dataset ?d has property ?p with value ?v
<?w hasInputDataset ?d>	WP (DP)	?d represents an input data variable to the workflow ?w
<?w hasOutputDataset ?d>	WP (DP)	?d represents an output data variable of the workflow ?w
<?w hasIntermediateDataset ?d>	WP (DP)	?d is a dataset of workflow ?w that is not an input or output data variable
<?w hasDataset ?d>	WP (DP)	?d represents a data variable in workflow ?w that can be an input, output, or intermediate data variable
<?w hasComponent ?c>	WP (CP)	?c represents a component in workflow ?w
<?w hasInputParameter ?p>	WP (CP)	?p is a parameter in workflow ?w

Table 2. Overview of query language constructs.

EXPRESSING QUERIES FOR WORKFLOW DISCOVERY

Queries for workflow discovery may be issued in a variety of contexts [5,6]:

- A user with a dataset is looking for ways to analyze it
- A user is creating a workflow and is looking for workflow fragments that have been created by others for specific functions
- A user has created a workflow and wants to find similar ones
- A user is browsing a workflow catalog and wants to search for workflows with general kinds of features
- A system can retrieve workflows to be brought up to the attention of a user, for example during workflow creation or when trying to execute workflows
- A system can retrieve a workflows and use it to generate a desired type of result requested by the user

As we mentioned in the introduction, a series of studies have documented what kinds of queries users wish to express for retrieving workflows [5-8], discussed as categories I-VI above. Users often describe the desired workflow in terms of the output produced and input used, or may indicate some properties of intermediate data. We refer to all these as *data-centered properties (DP)*. Q1 is an example where input data properties are specified, Q2 is an example for output and Q4 for intermediate data.

Another finding of those studies is that users also want to be able to specify the kinds of algorithms (components) or component types that are used in the workflow. We call these *component-centered properties (CP)*. Q3 and Q5 are examples of queries with such properties, though note that they also include data-centered properties.

Other queries would specify constraints on the structure of the workflow, concerning the relative ordering of the data processing steps. Users wanted to specify a partial or total order for the steps. In addition, users wanted to specify the types of data generated and in what order. We refer to both of these as *structural properties (SP)*. Examples are Q4 and Q5, where three types of data to be generated are mentioned and their relative order is specified.

Our work focuses on supporting user queries that specify data-centered properties (DP) and their combination with component-centered (CP) and structural properties (SP). These comprise all but category V discussed earlier.

Table 2 shows the constructs in our query language to represent user requirements for DP, CP, and SP. The language includes constructs to refer to *workflow properties (WP)*, which indicate the role of datasets and components in the workflow. These are the last six constructs shown in the table. The table specifies each construct's type and its use. Each construct is expressed as a triple <object property value>. Disjunction and negation are not allowed.

The semantics of most constructs are straightforward, but there is an important aspect to discuss. We make a particular choice of those proposed by [15] over others that have been proposed for semantic service discovery. In that approach, a matcher should support retrieval with the opposite subclass relations as well intersection of the classes. However, based on our experience we make the following choices. First, for <?c Subclass-of ?t> we assume that the semantics the user intends is that only workflows which contain a component of class ?t or a more specific class should be retrieved. For example, for <?c Subclass-of DecisionTreeclassifier> it is ok to retrieve workflows with ID3classifier and with LMTclassifier, but not workflows that have more general classes such as TreeClassifier. The rationale for this is that if a user found a more general component class acceptable they would not have indicated such a specific class. Second, when the user specifies data properties we assume that the semantics intended is that any workflows appropriate for more general classes should be retrieved. For example, for <?d hasType BayesModel> and <?d hasDomain weather> it is ok to retrieve workflows that generate datasets of type Model and

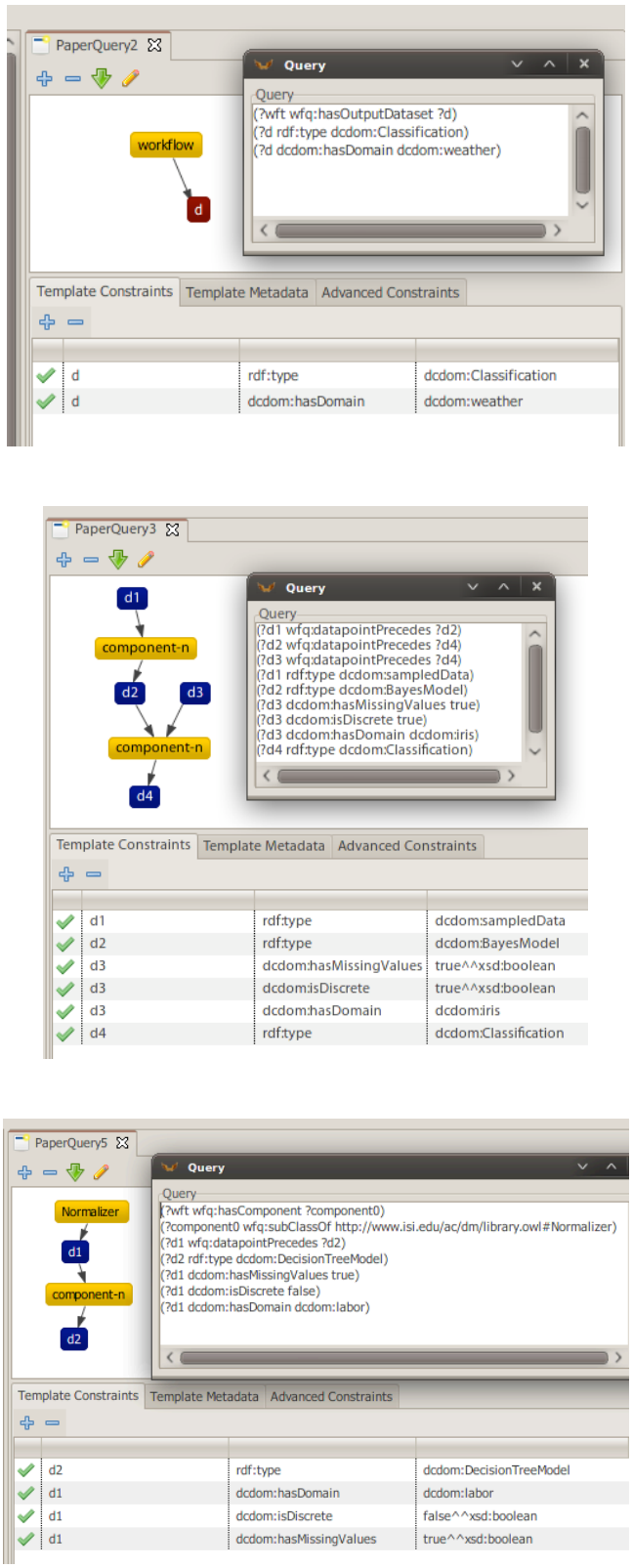


Figure 3. Snapshots of the user interface to specify queries Q1, Q2, Q4, and Q5, showing the formal query expression formulated by the system in the pop-up window.

do not specify the domain. The rationale here is that the user is looking for a workflow that can be used to analyze or to generate data of a specific type, and any workflows that process or generate data of a more general type are appropriate. Therefore, note that the semantics are very different for component classes and for dataset classes.

Figure 3 shows snapshots of the user interface when specifying the example queries, with a pop-up window showing the formal representation of the query. Users would have used a similar interface to create and view workflow templates (such as those shown in Figures 1 and 2). So, like a workflow, a query consists of a dataflow diagram and a table of constraints (triples). Much as they do in the workflow editor, users can create a query by selecting elements and drawing links among them, and by defining constraints on the query data variables. When defining constraints, the system offers a pull down menu of possible properties of data variables. When a property is selected, the system shows possible types and values.

The query editor has a few additional elements in the dataflow diagram pane that do not exist in the workflow template editor. To express that a dataset is a workflow input or output dataset (as in Q1 and Q2), the user can draw a node and label it as "workflow". To express one or more components to process a dataset (as in Q4 and Q5), the user can draw a node and label it as "component-n." This type of node is also used to express a query as a series of data points (as in Q4).

Note that our system formulates the formal query (shown in the popup windows of Figure 3) based on what the user specifies in the query interface.

APPROACH TO WORKFLOW MATCHING

Our approach has three novel contributions, described in this section:

1. An algorithm for enriching workflow catalogs that incorporates in each template relevant properties of data and components and then propagates these properties throughout the workflow
2. An algorithm for workflow matching based on data-centered properties and their combination component-centered and structural properties
3. A separation of reasoning steps that can be called out to external component and data catalogs, concerning individual components and datasets

Data and Component Catalog Requirements

In order to support the reasoning mechanisms for workflow retrieval, we require that the data catalog and the component catalog support the following functions:

- `subsume:` $DOD(d_1) \times DOD(d_2) \rightarrow Boolean$
  Determine whether data object d_1 is compatible with data object d_2 by checking if a set of metadata annotations on d_1 subsume those on d_2.
- `combineDODs:` $DOD(d) \times DOD(d) \rightarrow DOD(d)$
  Combine two sets of metadata annotations on the same data object d.

- `findDataRequirements: DOD(c) → DOD(c)`
  Return a possibly larger set of constraints on the arguments of component *c* given an initial set of constraints on those arguments.
- `subsume: c1 × c2 → Boolean`
  Whether a component class subsumes another.

Note that the findDataRequirements function in the component catalog can be supported in two alternative ways. One is simpler and only returns input and output types. In that case, our matcher only matches based on types. The second way is to return any constraints inferred in addition to types. Those constraints can be propagated through the workflow and used for matching. As we illustrated with our query examples, matching based on propagated constraints achieves higher precision than matching simply based on types.

Enriching Workflow Catalogs: Propagating Semantic Properties of Workflow Templates

We enrich workflows by propagating through the workflow structure constraints about the components and about particular datasets as obtained from the data and component catalog. The propagation of constraints is done in two phases: first, from workflow outputs to workflow inputs (*backward sweep*); and then, from workflow inputs to outputs (*forward sweep*). These two phases are applied iteratively until no changes occur. Table 3 shows the pseudo-code for the backward sweep. The forward sweep uses a similar algorithm.

Workflow Retrieval

Given a query, for every workflow template in the library we generate possible mappings between variables in the query and variables in the template. If a mapping is found the template is added to an initial set of candidate matches together with the possible variable mappings. From that set, we filter out those that violate any of the query constraints. This section describes this algorithm in detail.

A query *q* can be represented as a tuple <*DVq*, *CVq*, *DPq*, *CPq*, *SPq*, *WPq*> containing a set *DVq* of data variables, a set *CVq* of component variables, a set *DPq* of data-centered properties, a set *CPq* of component-centered properties, a set *SPq* of structural properties, and a set *WPq* of workflow properties. A workflow template *wt* can be represented as a tuple <*DVwt*, *Cwt*, DOD(*DVwt*), L> containing a set *DVwt* of data variables, a set *Cwt* of components, a set DOD(*DVwt*) of data object descriptions on the data variables of the workflow, and a set L of links that express the dataflow among components and which data variables correspond to that dataflow link.

We define a mapping *map* between a query *q* and a template *wt* as a function that associates each data variable in *q* to a data variable in *wt* and each component variable in *q* to a component in *wt*:

$$\forall\, dv \in DVq \;\exists\; map(dv) \in DVwt$$
$$\forall\, cv \in CVq \;\exists\; map(cv) \in Cwt$$

BackwardSweep(*wt*)
1. *backward-processed-DVs* ← output-DVs(*wt*)
2. *remaining-components* ← C_{wt}
3. while not empty(*remaining-components*)
4. *c* ← selectBackwardProcessable(*remaining-components*)
5. DOD(*c*) ← { }
6. *remaining-components* ← *remaining-components* - {*c*}
7. forEvery *arg* ∈ output(*c*)
8. let *dv* ≡ destination(*c*, *arg*)
9. DOD(*c*) ← `combineDODs`(DOD(*c*), DOD(*dv*)(*dv* | *arg*))
10. DOD(*c*) ← `findDataRequirements`(DOD(*c*))
11. forEvery *arg* ∈ input(*c*)
12. let *dv* ≡ origin(*c*, *arg*)
13. DOD(*dv*) ← `combineDODs`(DOD(*dv*), $DOD_{arg}(c)$(*arg* | *dv*))
14. if destination(*dv*) ∩ *remaining-components* = ∅ then
15. *backward-processed-DVs* ← *backward-processed-DVs* ∪ {*dv*}

where:
- *wt* is a workflow template
- output-DVs(*wt*) returns the set of output data variables in *wt*
- C_{wt} represents the set of components in *wt*
- selectBackwardProcessable returns a component in the workflow all whose output arguments are linked to data variables in *backward-processed-DVs*
- input(*c*) (output(*c*)) represents input (output) arguments of *c*
- destination(*c*, *arg*) represents the data variable where the result of output argument *arg* of component *c* is stored
- origin(*c*, *arg*) represents the data variable that stores the input for the argument *arg* of component *c*
- $DOD_{arg}(c)$ represents the subset of constraints on arguments of component *c* that only refers to argument *arg*
- (id_1 | id_2) represents the replacement of identifier id_1 with identifier id_2 in a set of constraints

Table 3. Algorithm to augment workflow templates by propagating constraints from data products to data inputs.

Given a query *q* and a workflow template *wt* we first generate all possible mappings from variables in the query to data variables and components in the workflow that fulfill the set *WPq*. The candidate mappings are filtered in three consecutive steps using the rest of the query:

1. Filter out candidate mappings that violate *CPq*.
2. Filter out candidate mappings that violate *SPq*.
3. Filter out candidate mappings that violate data properties *DPq*. That is, for every data variable in the query, *dv* ∈ *DVq*, the data catalog `subsume`(DOD*(m(dv)*), *DPq(dv*)) must return true where DOD*(m(dv)*) represents the subset of constraints in DOD*(DVwt*) that only refers to *m(dv*), and *DPq(dv*) represents the subset of constraints in *DPq* that only refers to *dv*.

There is a tradeoff between precision and performance that is worth mentioning here. The constraint propagation algorithm can be run on each workflow template as it is being added to the library and prior to resolving any queries. Alternatively, it can be run at query matching time and incorporating all the constraints specified in the query into each workflow being matched. To achieve this, we

would simply propagate the constraints in the query through the workflow structure using the same backward and forward sweep algorithms. The first option makes the performance of the matcher much faster that running the constraint propagation algorithms, but the precision is higher with the second option as all the constraints that appear in the query are taken into account.

PERFORMANCE RESULTS

Our query interface and matcher are implemented as part of the WINGS workflow system [4,13]. We represent workflows and their constraints in OWL, and use Jena as the underlying reasoner. The data and component catalogs are implemented as separate services that support the functions described in the prior section.

We tested two different matchers. One matcher (**m-types**) uses only data types, that is the component catalog returns only input and output type information. The other matcher (**m-constr**) propagates not only data types but also all other constraints on datasets.

We looked at queries in five categories. Category QC1 corresponds to queries that have constraints on input data. QC2 are queries with constraints on output data. QC3 are queries with constraints on components. QC4 are queries that have constraints on intermediate data. QC5 are queries that include constraints both on components and on data. Each of the queries Q1-Q5 in Table 1 belongs to one of the QC1-QC5 categories. We created four additional queries for each category.

We used three different workflow catalogs. CAT1 contained 20 workflows, CAT2 contained 100 workflows that could be in principle relevant, and CAT3 included 100 workflows that would not be relevant. We created relevant workflows with several variations of steps and data constraints. We created irrelevant workflows by adding a constraint that made the domain different from the domain used in all the queries.

Table 4 shows the performance for the two matchers and the three workflow libraries across different categories of queries. We show the query response time in seconds, using an Intel Core 2 Duo 2.6GHz with 2GB. We also show the number of workflows matched and the precision.

The **m-constr** matcher that uses propagated constraints shows in these experiments:

- Higher precision: **m-constr** always returns less matches than **m-types**, even when the query does not have DP constraints.
- Same recall: Both matchers return all the correct answers.
- Comparable performance: **m-constr** is rarely slightly slower and often slightly faster, than **m-types**, since it ends up ruling out candidate matches earlier in the matching process. QC4 queries take longer since there are many possible mappings of data variables that need to be considered.
- Comparable scalability: Performance against larger catalogs is not necessarily slower, as it depends on how early the matching process terminates for each template.

Q. category (features used)	CAT1: 20 workflows in library		CAT2: 100 workflows in library that could be relevant		CAT3: 100 workflows in library that would not be relevant	
	m-types: Avg Retr. Time (Avg # retrv'd, prec'n)	**m-constr:** Avg Retr. Time (Avg # retrv'd, prec'n)	**m-types:** Avg Retr. Time (Avg # retrv'd , prec'n)	**m-constr:** Avg Retr. Time (Avg # retrv'd, prec'n)	**m-types:** Avg Retr. Time (Avg # retrv'd)	**m-constr:** Avg Retr. Time (Avg # retrv'd)
QC1 (DP, WP)	0.16 (15.8, 0.67)	0.16 (10.6, 1)	0.40 (39.0, 0.68)	0.41 (26.6, 1)	0.41 (0)	0.40 (0)
QC2 (DP, WP)	0.10 (15.0, 0.93)	0.10 (14.0, 1)	0.25 (35.8, 0.96)	0.25 (34.4, 1)	2.60 (0)	0.25 (0)
QC3 (CP, WP)	0.07 (9.0, 0.84)	0.06 (7.6, 1)	0.16 (22.0, 0.85)	0.17 (18.6, 1)	0.15 (0)	0.01 (0)
QC4 (DP)	0.30 (11.0, 0.38)	0.29 (4.2, 1)	0.80 (30.8, 0.31)	0.81 (13.5, 0.70)	0.81 (0)	0.81 (0)
QC5 (CP, DP, WP)	0.11 (3.0, 0.38)	0.11 (0.8, 1)	0.28 (5.8, 0.21)	0.28 (1.2, 1)	0.07 (0)	0.07 (0)

Table 4. Performance of the workflow retrieval system.

RELATED WORK

As we mentioned earlier, prior work on workflow matching does not address data-centered queries, but rather focuses on queries based on components and their orderings or tags added by users [6,14].

Prior research investigated how to enrich user queries by inferring the value for additional attributes to support matching [9]. However the work was applied to enriching queries rather than workflows, and also used simpler domain models with minimal dependencies between attributes.

The backward and forward sweeps are essentially a form of goal regression [16] and forward projection [2] used in AI planning. There are subtle differences in that initial states in planning contain ground literals while workflow templates include constraints that are not ground. When we consider workflows as composed of steps that change the properties of datasets, we find that workflow matching is related to plan matching [12,20]. Plans are retrieved according to the aspects of the initial state and goals in the query. When the plans are stored, they are augmented with the necessary conditions that justify the choices made during planning. In our work we do not annotate choices, rather we augment the description of the datasets. Our work would be analogous to augmenting the description of a planning state, which the abovementioned research does not need to do since the states are always assumed to be fully described.

CONCLUSION

We have presented a novel approach to workflow discovery that allows users to query based on data-centered properties. The key idea of the approach is to augment

workflow descriptions with semantic information to achieve higher precision in the matching process. We described an algorithm to enrich workflow templates created by users that do not contain many important constraints. These constraints are obtained from component catalogs and incorporated and propagated through the workflow. We also described an algorithm for matching workflows that calls out to external component and data catalogs to do the aspects of the reasoning that concerns individual components and datasets. This is essential to support scientific applications.

Workflow repositories have the potential to transform computational science. Today, much effort is invested in re-implementing scientific analysis methods described in publications. If workflows were routinely shared by scientists, they would be readily reusable and as a result that effort will be saved. This would also facilitate reproducibility of scientific results, a cornerstone of the scientific method. Effective techniques for matching and discovery are key incentives for scientists to share analytic methods as reusable computational workflows.

ACKNOWLEDGMENTS

This research was funded in part by the National Science Foundation under grant number CCF-0725332.

REFERENCES

[1] Asuncion, A. & Newman, D.J. UCI Machine Learning Repository Irvine, CA: University of California, School of Information and Computer Science, 2007. Available from www.ics.uci.edu/~mlearn/MLRepository.html.

[2] Fikes, R. E., and Nilsson, N. J. "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", Artificial Intelligence, 2(3), 1971.

[3] Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L. and J. Myers. "Examining the Challenges of Scientific Workflows." IEEE Computer, vol. 40, no. 12, 2007.

[4] Gil, Y., Ratnakar, V., Deelman, E., Mehta, G. and J. Kim. "Wings for Pegasus: Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows," Proceedings of the 19th Annual Conference on Innovative Applications of Artificial Intelligence (IAAI), Vancouver, British Columbia, Canada, 2007.

[5] Goderis, A., Sattler, U., Lord P. and C. Goble. "Seven bottlenecks to workflow reuse and repurposing." Proc. of the 4th Int. Semantic Web Conference, Galway, Ireland, November 2005.

[6] Goderis, A., Li, P. and C. Goble. "Workflow discovery: the problem, a case study from e-science and a graph-based solution." International Journal of Web Services Research, Vol 5, No 4, 2008.

[7] Goderis, A., De Roure, D., Goble, C., Bhagat, J., Cruickshank, D., Fisher, P., Michaelides, D., and Tanoh, F. "Discovering Scientific Workflows: The myExperiment Benchmarks." Internal project report, submitted for publication. 2008.

[8] Goderis, A., Fisher, P., Gibson, A., Tanoh, F., Wolstencroft, K., De Roure, D. and C. Goble. "Benchmarking Workflow Discovery: A Case Study From Bioinformatics." To appear in Concurrency and Computation: Practice and Experience.

[9] Gupta, K.M., Aha, D.W., and Sandhu, N. "Exploiting Taxonomic and Causal Relations in Conversational Case Retrieval." Proceedings of the European Conference on Case-Based Reasoning (ECCBR), 2002.

[10] Hull, D., Zolin, E., Bovykin, A., Horrocks, I., Sattler, U. and R. Stevens. "Deciding semantic matching of stateless services." Proceedings of the Annual Conference of the American Association for Artificial Intelligence (AAAI), 2006.

[11] Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M., Li, P., and T. Oinn. "Taverna: A Tool for Building and Running Workflows of Services", Nucleic Acids Research, Vol 34, 2006.

[12] Kambhampati, S. and J. A. Hendler. "A Validation Structure-Based Theory of Plan Modification and Reuse", Artificial Intelligence Journal. Vol 55, 1992.

[13] Kim, J., Deelman, E., Gil, Y., Mehta, G. and V. Ratnakar. "Provenance Trails in the Wings/Pegasus Workflow System," Concurrency and Computation: Practice and Experience, Special Issue on the First Provenance Challenge, Vol 20, Issue 5, April 2008.

[14] Leake, D. and J. Kendall-Morwick. "Towards Case-Based Support for e-Science Workflow Generation by Mining Provenance." Proceedings of the European Conference on Case-Based Reasoning, 2008.

[15] Li, L. and I. Horrocks. "A software framework for matchmaking based on semantic web technology." Proceedings of the Twelfth International World Wide Web Conference (WWW), 2003.

[16] McDermott, D. "Regression Planning". International Journal of Intelligent Systems, Vol. 6, Issue 4, 1991.

[17] Moreau, L. and B. Ludaescher (Eds). Special Issue on the First Provenance Challenge. Concurrency and Computation: Practice and Experience. 20(5), 2008.

[18] Reich, M., Liefeld, T., Gould, J., Lerner, J., Tamayo, P., and J. P. Mesirov. "GenePattern 2.0." Nature Genetics Vol 38 No. 5, 2006.

[19] Taylor, I., Deelman, E., Gannon, D., Shields, M., (Eds). "Workflows for e-Science", Springer, 2007.

[20] Veloso, M. M., "Planning and Learning by Analogical Reasoning." Springer Verlag, December 1994.

[21] Witten, I. H. and Frank E. "Data Mining: Practical Machine Learning Tools and Techniques." Morgan Kaufmann, San Francisco, 2nd edition, 2005.