Documentation: Motif Finder

# Introduction

Motif Finder is a project designed to find commonalities among scientific workflow templates and executions. The project explores a workflow repository and analyzes its contents applying different graph mining techniques. As a result, a set of the most common workflow fragments is obtained.

The final results are linked to the original contents of the repository according to the Workflow Fragment Description Ontology: <http://purl.org/net/wf-fd>. In addition, some of the fragments are exposed as internal macro and composite workflow motifs, as defined in <http://purl.org/net/wf-motifs#InternalMacro> and <http://purl.org/net/wf-motifs#CompositeWorkflow> respectively.

Additionally, crucial metadata regarding the algorithm used and the date of creation of the fragment dataset is provided **(CURRENTLY ONGOING)**.

# Overview

Motif Finder exposes an extensible framework to:

1. **Query different scientific workflow specifications published in a repository**. (Currently supporting OPMW/OPM workflows).
2. **Reformat these specifications to produce ready for execution scripts for the different graph mining techniques**. After retrieving the dependencies between the different steps of the workflow, we expose the resultant graphs according to the format required by the different graph mining algorithms. (Currently supporting the SUBDUE algorithm (WIN OS), and PAFI (Linux)).
3. **Interpret the results to find the different fragments found by each technique**.
4. **Produce the links between the fragments obtained and the different workflow specifications**. By stating this relationship, we are able to determine where the different fragments were found in the repository, and how often they appear.
5. **Expose the final results in RDF**, with the Wf-fd model, so we can query them via SPARQL queries.
6. **Find where the fragments can be found in the original corpus**, by issuing SPARQL queries to the repository used to extract the templates.
7. **Validate the produced results**. In order to check that the results have been produced correctly, the framework exposes different tests to check the functionality of individual steps (retrieve templates from the repository, validate fragments, etc.).

# Abstract Workflow

Given the functionality explained above, the goal of the framework is to be **used to develop a workflow** to manipulate the data for the different graph matching algorithms (which are not part of the main functionality of the framework, as they have been developed by other scientists), interpret their results and expose them in RDF.

Although the framework has been designed to be used (and extended) with different graph matching algorithms and different workflow languages and repositories, there is a sequence of common steps, summarized as the abstract workflow exposed in Figure 1.

Figure 1: Main abstract workflow for retrieving the most common workflow fragments, for any repository and any graph mining algorithm. The rectangles represent the different steps, while the orange ellipses represent the inputs and outputs. Each input is connected by an arrow to the step that produces it or consumes it. It is worth mentioning that normally the “instantiate results” step might be performed together with the transformation to RDF.

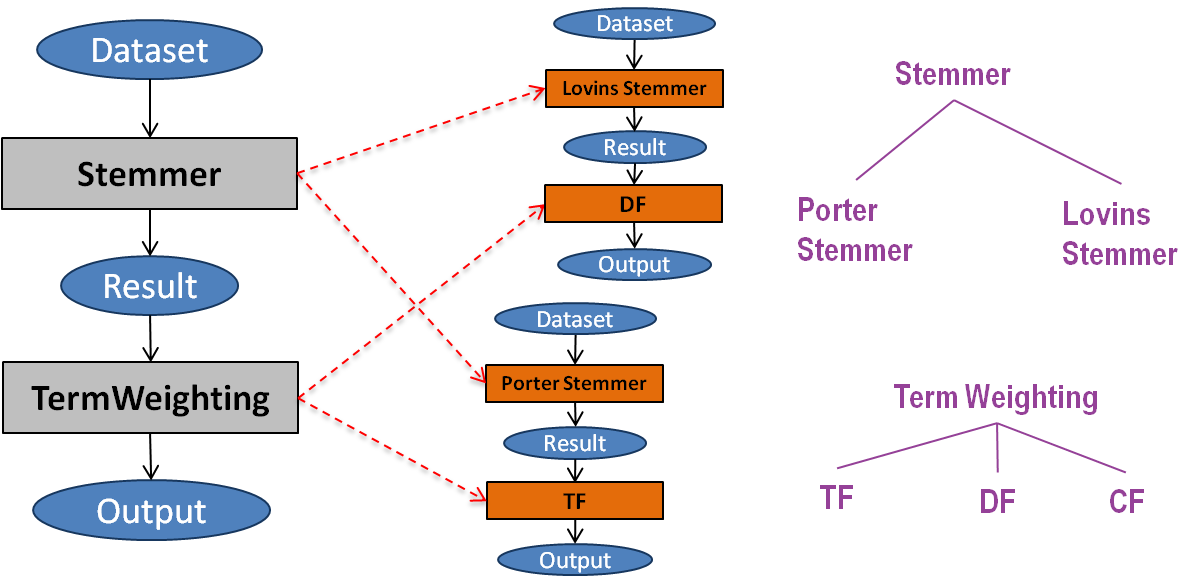
As Figure 1 depicts, the abstract workflow has 5 main steps:

1. **Query Repository and reformat input**: The first step is to retrieve successfully the template specification or runs in which we want to find the most common workflow patterns and transform them to a dependency graph. Once the data from the repository has been interpreted, it may be transformed to the format required by the different graph mining algorithms.
2. **Graph mining algorithm**: The second step consists on the execution of the graph mining algorithm of choice by the user (currently supported: SUBDUE, FAPI). The output of the algorithm is the fragment catalog, which has to be parsed by the framework to be properly interpreted.
3. **Instantiate results**: Once we have the fragment catalog, we aim to find for each fragment where it has been found in the workflow catalog. This is a step necessary to understand the support of each detected fragment by the workflow catalog, and critical to understand how the different workflows are related to each other.
4. **Transformation of the catalog to RDF**: Once the fragment catalog has been instantiated in step 3, the results of the instantiation and the fragment catalog are transformed to RDF following the Wf-fd model. As a result, an RDF file is produced. The file will also contain all the metadata related to the algorithm and process creation followed. (Note: in some cases, steps 3 and 4 are performed together in a single step)
5. **Generation of statistics**: Optional step designed to generate additional statistics from the detected fragment catalog (e.g., multi-step fragment, fragments included in other fragments, relevant fragments, etc.).

## Generalization workflow

One of the features of the framework is that we can generalize workflow templates and workflow execution provenance traces in order to derive abstract workflow fragments. With the generalized workflow fragments, the graph mining algorithms are likely to find more commonalities and bigger common fragments.

An example of the generalization can be seen in Figure 2 below:



**Figure 2: 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.**

In Figure 2, an abstract workflow template (on the left) is generalized from 2 different workflow templates (in the middle), according to the taxonomy on the right. In order to generalize a workflow, **a taxonomy of components is needed**. If the taxonomy is not available, no generalization will be possible.

The generalization step modifies the abstract workflow presented in Figure 1 by adding an additional step (CreateAbstractCatalog):

Figure 3: Abstract workflow with the Generalization step incorporated (Create Abstract Catalog step).

## The OPMW - SUBDUE workflows

Specialized workflows designed to query the OPMW repository (<http://opmw.org/sparql>), format the results for the SUBDUE (<http://ailab.wsu.edu/subdue/>) algorithm, parse the results and produce the statistics and Wf-fd representation. A figure of the extended workflows can be seen below.

Figure 4: Specialized workflows using the OPMW repository and the SUBDUE algorithm

## The OPMW - PAFI workflows

Same as the previous one.

(TO DO) represent these workflows as Wings Workflows and past them here.

# Functionality

blah

## Architecture

UML diagram here simplifying everything. Or instead of that just a high level overview.

## Data Structures

Graph, nodes, Fragments, etc.

Here we have to talk about the input formats and the output formats as well. And the usage as well.

Different steps and operations go here! In theory I should explain here how the invocation works, after giving an insight on the architecture and structures that are being used.

What is the main architecture of the system?

What are the structures being used, and how do they relate to each other?

What is the main Diagram of UML classes modeling everything?

I have to explain here also the different functionalities of what the project does: how to process a template or trace form a repository (and which classes to extend), how to change a domain, how to save the files in a format, how produce the script to actually run SUBDUE or any other graph, how to create statistics, how to validate the fragments found, how to generate the fragment catalog in wffd, etc.

How to switch from different algorithms and how to attack different sources (which is in fact easy).