2007－MPS－67 (57)
2007－BIO－11 (57)
2007／12／21

# TF-IDFフィルタリングによる機能的に類似した生物情報解析ワークフローの検索手法

瀬尾 淳哉，瀬尾 茂人，竹中 要一，松田 秀雄
大阪大学大学院 情報科学研究科

**概要**： 生物情報解析の分野おいて，解析ツールや生物学のデータを効率的に扱う方法が重要になってきている．これらの解析ツールは様々な機関から提供されており，その数は増えつつある．解析ツールをWebサービスの形態で提供する機関も増えてきており，それらを組み合わせたワークフローという手法が生物情報解析において注目を集めている．ワークフローの作成や実行を補助するためのツールの開発も進められている．同じくワークフローの作成を支援するために，ワークフローの再利用が考えられるようになってきた．しかし現在では，再利用を行うために既存ワークフローの検索を行うことが容易ではない．本論文では，再利用の可能なワークフローを既存ワークフローの中から検索する手法と提案する．提案手法を用いることにより，高い正解率で再利用可能なワークフローを検索することができた．

# Retrieving Functionally Similar Bioinformatics Workflows using TF-IDF Filtering

Junya Seo, Shigeto Seno, Yoichi Takenaka, Hideo Matsuda
Graduate School of Information Science and Technology, Osaka University

**Abstract:** In bioinformatics, dealing with tools to analyze biological data becomes important. Those tools are provided by various institutions and the number of the tools is rapidly increasing. Recently many institutions have been offering those tools and access to databases with Web service technologies. The workflow technology is one of the ways to manage those tools and it is becoming available in bioinformatics. In order to compose workflows, several research groups develop and provide workflow composing tools. And consequently, the concept "Workflow Reuse" is also arisen in order to help workflow composition. Nevertheless it is still difficult to search for the reusable workflows from the repository of workflows in the current situation. In this paper, we propose a method to extract reusable workflows from the repository by using currently available information. We could extract some functionally similar workflows as reusable ones. By extracting reusable workflows efficiently, researchers can compose their workflow more easily.

## 1 Introduction

In the field of bioinformatics, the tools to analyze biological data have been provided by various institutions. There are many types of data and tools for biological analyses. Many tools are developed for the same or similar analyses. For example, BLAST[2] and FASTA[9] are the tools to search DNA/Protein sequence databases. The number of tools has been increasing year by year and now it becomes huge[3]. Therefore how to deal with them becomes important.

Recently several institutions have started to provide programmatic access to biological databases and analysis tools based on Web service technologies[15] (e.g. XEMBL[16], openBQS[12], Soaplab analysis services[13], XML Central of DDBJ[7] and the KEGG API[5]). With these publications of resources as Web services, researchers are making a shift from traditional navigation using hyperlinks through a sequence of Web pages provided by those resources to the use of distributed services such as Web services for experimental design, data analysis and knowledge discovery. Nevertheless it is still difficult to compose distributed services with Web service technology, whether manually or automatically. In order to compose the services, researchers should be familiar with some techniques such as programming.

When researchers use analysis tools, some of them should be installed locally, others can be utilized via Web browsers though the support of Web services. Usually, these tools are not used solely in the biological analysis, but used in combination with each other. In the typical case of using tools sequentially, the output data of a certain tool is supplied to an input of another tool. However, particularly, the use of the Web service requires the high leveled programming skills. The hugeness of the number of tools and various ways of using tools are also become the barriers for using several tools together.

In order to solve the problems, workflow management arose as a method of handling analysis tools[1]. Although the workflows are not widely used yet, they will be in the near future, which can be predicted from the situations in the business field, etc[14]. Researchers can combine Web services, local tools and some other resources into workflows. They potentially allow researchers to describe their experimental processes in structured, repeatable and verifiable ways. By using this technique, programming technique is not needed so much. Fig. 1 shows an example of workflow. The parentheses with "sequence" and "database" at the top of the workflow are input terminals of the workflow and "multiple alignment" at the bottom of the workflow is an output terminal. The squares with "BLAST" and "ClustalW" are the tools used in this workflow and the parentheses with "seq", "database" and "result" around the tools are input/output ports of the tools. In this case, workflow has two functions "Homology Search" and "Get multiple alignment". These functions are invoked sequentially on the workflow.

The increasing of the number of bioinformatics Web ser-

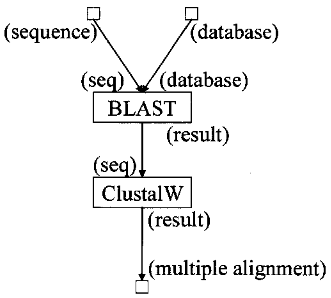


Figure 1: An example of workflow

vices makes management of workflows more important. When researchers compose workflows, they choose some services for their workflows. At this point, it becomes important factor to consider what service or combination of the services suit for their purpose. In order to help workflow composition, the concept to store and reuse workflows arises[10]. In some repository of bioinformatics domain, researchers are now starting to pile up the collections of workflows. Therefore how to utilize the repository will become important.

As a related work on workflow search and reuse, a vision for reuse of scientific workflows is described by Medeiros et al. [6] for a closed system. But the paper does not consider the on-line workflows. Also in some workflow composing tools, such as Taverna, implement only search mechanism of tools on a workflow. Thus users cannot search specific workflows from the piled workflows by workflow function. Now many workflow projects aim how to compose a workflow easily. Thus few workflow repositories have been made publicly available and even fewer have similar workflows in them which can be used to evaluate search techniques. Some projects, such as Kepler[4] and Taverna, are building a platform with workflow reuse in mind. Therefore searching and reusing workflow from the on-line repository are important as a next stage.

In this paper, we propose an extraction method of workflows from the piles of workflows by focusing functional similarity. For this purpose, our method uses data types of workflows indicated by data names. By extracting functionally similar workflows, researchers can use them for composing a workflow for their target analysis.

# 2 Workflow

In this section, we describe the composition, reuse and search for workflows. In order to refer to that, we also describe the current situations around workflows.

## 2.1 Workflows in Bioinformatics

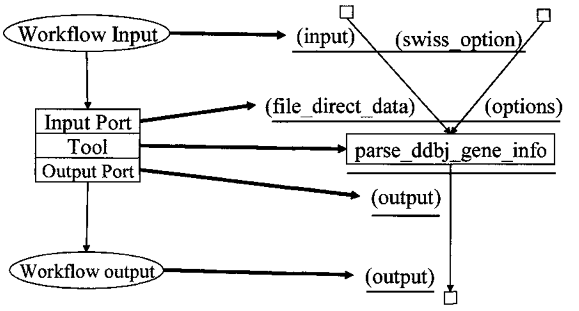
A workflow is a set of flows of operations and data. It consists of local tools, Web services and data flows. The local tools and Web services are executed according as the description in a workflow. In bioinformatics, workflow techniques are spreading among researchers because of increasing the variation of analysis tools and developing of Web service technologies[8]. Researchers frequently use combinations of several analysis tools on their researches. The connections between those tools are often managed manually (e.g. copy and paste in web pages), and manually management is often cumbersome[1]. Researchers can remove the burden of this management by using workflows. Once researchers have composed a workflow, they can run it as many times as they want. They can also make small changes such as input data or thresholds to the workflow. Workflows run automatically from input to output on workflow engines, therefore even if researchers change parameters on their research, it is easy to do whole process again.

Fig. 2 shows the basic composition of workflows. A workflow basically consists of "Workflow Input", "Tool", "Workflow Output" (see the left side of Fig. 2). "Tool" includes local tools and Web services. "Tool" also has "Input port" and "Output port". Those are connected by the arrows. The arrows indicate data-flows. Then right side of Fig. 2 shows an example. The items at the top of workflow are "Workflow Input". In this figure, "input", "swiss_option" are the input terminals of workflow. Workflows basically run by considering these input terminals as the start points. The next items show the input ports of the tool. "file_direct_data" and "options" show input ports of the "parse_ddbj_gene_info". The next item shows the tool that is "parse_ddbj_gene_info". The tools execute some operations on the workflow and combinations of these tools construct the workflow function. The item "output" below the tool name represents the output port of the tool. The tools make some data as results of their work. Those are passed to next tools or workflow output. The item at the bottom of workflow is "Workflow Output". "output" is the output terminal of the workflow. The workflow outputs are equivalent to workflow results and also those are the end points of the workflow.

## 2.2 Workflow Composition

When researchers make workflow, they have to decide components used in the workflow such as tools. There are two ways to compose workflows mainly. One is to use programming languages such as Java. Some programming languages can invoke local tools and Web services. Researchers can compose workflows as they like by using programming languages. However, in order to use those programming languages, researchers should have high programming skill. Another way is to use workflow composing tools such as Taverna[8]. Workflow composing tools, as its name indicates, help researchers with workflow composition. Many workflow composing tools have GUI interface, thus researchers can compose workflows intuitively. Researchers can compose workflows with few or no scripts by using workflow composing tools, because those tools hide complex architectures such as invoking of Web services. This is an easier way of workflow composition.

If researchers compose their workflows with a workflow composing tool, there is the problem about assurance of their workflows. The workflow composing tools do not care whether a tool can receive the data correctly from the previous tool, thus every tool can connect to every tool on those workflow composing tools. But if there is difference of data format between an output of the former tool and an input of the latter tool, data-flow does not run correctly. Therefore researchers should check all data-flow on their workflows



*Workflow basic components* *Taverna workflow example*

Figure 2: Workflow composition

strictly.

## 2.3 Workflow Reuse

For the difficulty of workflow composition, the concept "Workflow Composition by Reusing Validated Workflows" arise[10]. This concept is to modify validated workflows when researchers compose new workflows. Validated workflows are the workflows composed by other researchers and those are confirmed as work correctly. Fig. 3 shows an image of the concept. In this figure, the cylinder with "repository" shows the repository of workflows, the repository holds many validated workflows. "WF" shows validated workflows extracted from the repository for the target workflow. "tool" is a local tool or a Web service. "Target Workflow" is a workflow which researchers want to compose. A workflow can include other workflows as subworkflows. In this concept, the validated workflows from the repository are also used as components like tools for the target workflow. By using validated workflows as components, researchers can compose assured workflows easily, because validated workflows should run correctly. Even if researchers are not satisfied with the validated workflows, they can improve those workflows to compose target workflows. They improve an existing workflow that is close enough to be the basis of a new workflow for a different purpose, and making small changes to it. Such an approach is the popular view in semantic Web services[10]. Actually researchers in bioinformatics often use similar data-flows with small changes. As described in section 1, collections of workflows are now starting to pile up in some institutions such as myGrid (`http://workflows.mygrid.org.uk/repository/`). The workflow reuse will be more important in near future.

However researchers cannot reuse workflows easily. Because it is hard to search for reusable workflows from the repository. The reason of this issue is the lack of information on workflows. Workflows have only flows of tools and simple names of tools, inputs or outputs. Thus researchers can search for one tool within workflows by the tool name, but they cannot search for the combination of tools from the repository by query function such as "Gene Annotation". There are few hints on workflows to consider that whether their functions are "Gene Annotation".

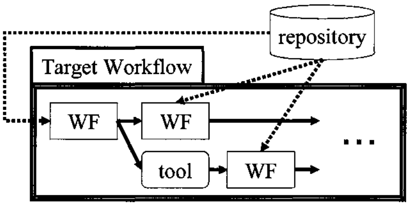


Figure 3: Image of workflow composition by reusing

## 2.4 Search against Repository

In order to extract reusable workflows from the repository, it is needed to extract workflows having similar function to the query function. When researchers want to search for workflows, they have data types of input and output. Once data types are decided, those data types indicate their target function. Therefore it is necessary to search for and extract functionally similar workflows from the repository by using data types of input and output as query. Workflows include one or more functions and one of them can be a target function, thus part of workflows also should be extracted. But there are ambiguities in descriptions on workflows. Thus we should consider how to extract workflows with tackling ambiguities of names and how to extract whole and a part of workflows based on their functions.

# 3 Method

In this section, we propose a method for extracting workflows from the repository of workflows.

## 3.1 Overview

Our method extracts workflows from the repository of workflows by a query. The query consists of a virtual workflow. The virtual workflow has names of input terminals, names of output terminals and one virtual tool. This virtual tool has no name and no ports. It only has a function designed by a user and our method predicts the function. Our method judges whether workflows in the repository have similar function to the function of the virtual workflow. The workflows in the repository are made with Taverna[8]

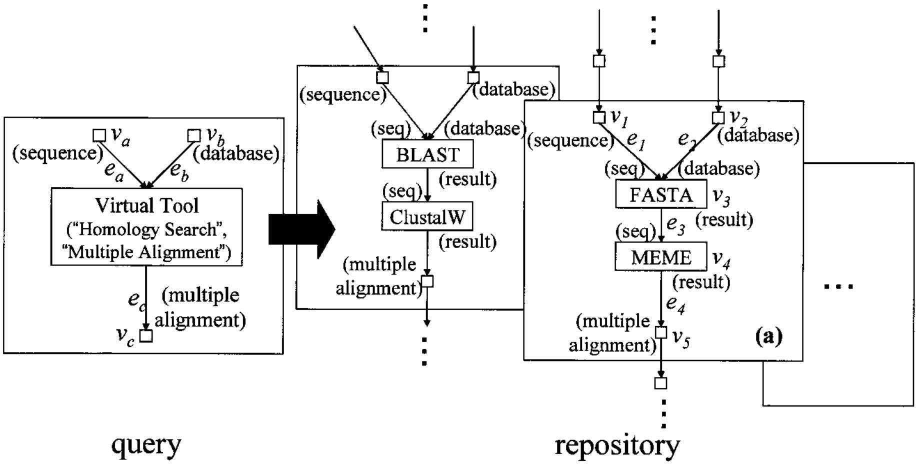


Figure 4: Functionally similar workflows

that is one of the workflow composing tools. Taverna is the most widely used workflow composing tool in bioinformatics. There are ambiguities on workflow descriptions of Taverna, thus information that we can use are restricted. Taverna workflow can only use names of tools, input/output ports, input terminals and output terminals on workflows without data types or data formats.

Our aim is to extract functionally similar workflows by considering functions on workflows. We regard workflows having similar functions hold similar types of input terminals and output terminals. We also regard a combination of data types on the input terminals and output terminals as a function of the workflow. Especially we regard data types on output terminals as important. If the output terminals of two workflows have similar data types, we suppose those are candidates of similar workflows and use data types on input terminals to accurize the similarity. Unfortunately workflows do not have such data types at present. For this problem, we consider names of input terminals, output terminals, input ports and output ports of tools are semantically similar to the data types. For example, when an input terminal has a name "sequence", we consider "sequence" represents a characteristic of the data on the input terminal. In order to extract similar workflows with similarity of data types on input terminals and output terminals, we use simple text matching of their names. Some workflows are still uncertain whether they have similar functions to the function indicated by the query. Thus we use Term Frequency - Inverse Document Frequency (TF-IDF)[11] weight. This weight is a statistical measure used to evaluate how important a word is to a document in a collection. We implemented filtering algorithm with TF-IDF to screen out uncertain results.

Fig. 4 shows the image of extraction by our method. Our method receives a virtual workflow, and then extracts workflows that are considered as having similar function to the function of the virtual workflow from the repository. For example, they input a virtual workflow having input terminals and output terminals such as "sequence", "database" and "multiple alignment" when researchers want to extracts workflows that invoke functions homology search and multiple alignment. Then, in Fig. 4, the combinations "BLAST"-"ClustalW" and "FASTA"-"MEME" are extracted as the combinations having similar function.

## 3.2 Algorithm

Input and output of our method are as follows:

**Input**:
a virtual workflow, repository of workflows

**Output**:
workflows

Our method consists of three steps.

**Step1**:
Extract the candidates of functionally similar workflows from the repository by the names of output terminals in the virtual workflow.

**Step2**:
Screen out candidates by the names of input terminals in the virtual workflow.

**Step3**:
Eliminate uncertain workflows with TF-IDF.

In order to describe our method, we represent workflows as labeled directed acyclic graph. We define the graph with following notations.

**$V$**:
$V$ indicates a set of nodes. The nodes represent tools, input terminals and output terminals of the workflow. Labels of the nodes mean names of the tools and the terminals.

**$E = \{e = (v_1, v_2) | v_1, v_2 \in V\}$**:
$E$ indicates a set of edges. The edges mean the connectivity in workflows. $v_1$ is the source of an edge and $v_2$ is the destination of the edge. The edges have two labels at the source point and destination point. Their labels mean name of the output port and name of the input port respectively.

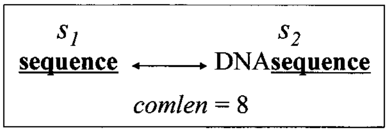


Figure 5: Text matching

$\boldsymbol{L = L(V) \cup L_s(E) \cup L_d(E)}$:

$L$ indicates a set of labels. $L(V)$ means labels of nodes. $L_s(E)$ means labels of source point on the edges. $L_d(E)$ means labels of destination point on the edges.

$\boldsymbol{G = (V, E, L)}$:

$G$ indicates workflows.

For example, the workflow (a) in Fig. 4, $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $E = \{e_1, e_2, e_3, e_4\}$. Then $L(v_3) =$ "FASTA", $L_s(e_1) =$ "sequence" and $L_d(e_1) =$ "seq".

We also use notations $\boldsymbol{G_{query}}$ as the virtual workflow, $\boldsymbol{\mathcal{G}_{repository}}$ as the set of workflows in the repository and $G_{repository}$ as a workflow in $\boldsymbol{\mathcal{G}_{repository}}$. We use a following function in the algorithm.

$\boldsymbol{match(s_1, s_2)}$

Receives text $s_1$ and $s_2$ and returns truth-value by following condition. We define length of $s_1$ as $s_{1len}$, length of $s_2$ as $s_{2len}$. Then we also define the maximum length of perfect matching between $s_1$ and $s_2$ as *comlen*. Fig. 5 shows an example of the text matching. Two strings "sequence" and "DNAsequence" are the arguments $s_1$ and $s_2$ respectively. In case of Fig. 5, *comlen* is 8.

Without loss of generality, we regard $s_{1len} \geq s_{2len}$. When $comlen/s_{1len} \geq 0.5$ and $comlen/s_{2len} \geq 0.8$, *match* returns TRUE. When arguments do not satisfy this expression, *match* returns FALSE.

In order to calculate these thresholds, we had an experiment with small dataset from the repository. We chose some extractable pairs of workflows and checked extractable values of this function respectively. We use the minimum values in the experiment that can extract correct workflows perfectly as the thresholds of this function.

### 3.2.1 Step1: Similarity Search with Output Names

In this step, we extract the candidates of the functionally similar workflows that can be considered as functionally similar from the repository by names of output terminals in the query. We define functionally similar workflow as follows: the workflow receives one or several data as inputs and results of workflow are the similar to the target workflow. Therefore we search nodes from the workflows in the repository that are similar to the names of output terminals on the virtual workflows. Then we consider the sub graph that consists of ancestors of extracted nodes as candidates.

Our method extract nodes according to the following conditions.

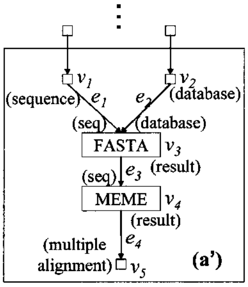


Figure 6: Output example of Step1

$$\begin{aligned} &\exists G_{repository} \in \mathcal{G}_{repository}, \\ &\exists e_{query} \in E_{query}, \exists e_{repository} \in E_{repository}, \\ &match(L_d(e_{query}), L_d(e_{repository})) = TRUE \end{aligned} \qquad (1)$$

Then our method outputs ancestor sub graph of

$$v_{repository\_d} \quad as \quad \boldsymbol{G' = (V', E', L')}$$
$$(e_{repository} = (v_{repository\_s}, v_{repository\_d}))$$

In above equations, $E_{query}$ is a set of edges of a virtual workflow. $E_{repository}$ is a set of edges of a workflow in the repository. $v_{repository\_d}$ is a destination node of an edge and $v_{repository\_s}$ is a source node of an edge.

We extract the nodes satisfies the expression (1). Then we output $\mathcal{G}'$ that is a set of workflows $G'$ consists of the ancestors of the extracted nodes. $G'$ is a sub graph of $G_{repository}$.

For example, the equation (1) is calculated as follows in the situation of Fig. 4.

$$\begin{aligned} e_{query} &= e_c \\ e_{repository} &= e_4 \\ L_d(e_c) &= \text{"}multiple\ alignment\text{"} \\ L_d(e_4) &= \text{"}multiple\ alignment\text{"} \\ match(L_d(e_c), L_d(e_4)) &= TRUE \\ e_4 &= (v_4, v_5) \end{aligned}$$

Therefore ancestors of $v_5$ are extracted as $G'$ like Fig. 6. In workflow (a'), $v_5$ is an end node.

### 3.2.2 Step2: Similarity Search with Input Names

In this step, we search nodes from workflows extracted in Step1 that can be considered as functionally similar with names of input terminals in the query. Our method screens out the candidates of Step1 and outputs survived candidates as $\boldsymbol{\mathcal{G}''}$ by the following condition. $\mathcal{G}''$ is a set of $\boldsymbol{G'' = (V'', E'', L'')}$. $G''$ is a survived workflow in expression (2). In the following condition, $v'$ is a node does not have outbound.

$$\begin{aligned} &\exists e_{query} \in E_{query}, \exists e' \in E', \\ &match(L_s(e_{query}), L_s(e')) = TRUE \end{aligned} \qquad (2)$$

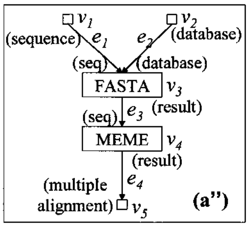


Figure 7: Output example of Step2

For example, the equation (2) is calculated as follows in the situation of Fig. 4 and Fig. 6.

$$\begin{aligned} e_{query} &= e_a \\ e' &= e_1 \\ L_s(e_a) = \text{"}sequence\text{"}, L_s(e_1) &= \text{"}sequence\text{"} \\ match(L_s(e_b), L_s(e_2)) &= TRUE \end{aligned}$$

$$\begin{aligned} e_{query} &= e_b \\ e' &= e_2 \\ L_s(e_b) = \text{"}database\text{"}, L_s(e_2) &= \text{"}database\text{"} \\ match(L_s(e_a), L_s(e_1)) &= TRUE \end{aligned}$$

$v_1$ and $v_2$ satisfy the condition. Therefore a sub workflow (a") is extracted as $G''$ like Fig. 7.

### 3.2.3 Step3: Elimination of Uninformative Results with TF-IDF

In this step, we eliminate the uninformative candidates with TF-IDF. The expression of TF-IDF scoring is as follows.

$$w = tf \times log(\frac{N}{df}),$$

where $w$ is the score of a word, $tf$ is the number of occurrences of the word $w$ in the workflows in the repository, $df$ is the number of workflows containing the word $w$ and $N$ is the total number of the workflows. We used names of input ports, output ports and tools in a workflow as target words for the TF-IDF.

First we calculate TF-IDF for all words in a workflow from $G_{query}$ and $G''$ and get a keyword that has the highest score in a workflow as follows.

***KEY(data)***:
This keyword is obtained from $L_s(e)$, $L_d(e)$, $L''_s(e)$ and $L''_d(e)$ with TF-IDF. Therefore this keyword indicates a name of input, output or ports in $G_{query}$ and $G''$.

***KEY(tool)***:
This keyword is obtained from $L(v)$ and $L''(v)$ with TF-IDF. Therefore this keyword indicates a tool name in $G_{query}$ and $G''$.

Second, we eliminate (or not) according to following conditions.

```
if
        $match(KEY(data)_{query}, KEY(data)'')$
        or
        $match(KEY(tool)_{query}, KEY(tool)'')$
then
        DO NOT eliminate this workflow $G''$.
else
        eliminate this workflow $G''$.
```

At *match* function, in above conditions, we changed the thresholds as follows.

**When $comlen/s1_{len} \geq 0.5$ and $comlen/s2_{len} \geq 0.5$, *match* returns TRUE.**

The thresholds differ from Step2, because the s1 and s2 is manipulated equivalently in Step3. Finally we output the survived workflows in $G''$ as the results of our method.

Table 1: Correctness of Querying

| correct | incorrect | correctness(%) |
|---|---|---|
| 143 | 15 | 90.5 |

# 4 Experiment

## 4.1 Results

In order to evaluate our method, we implemented and applied it to the workflow data of Taverna. The data had 197 workflows and the number of tools included in those workflows was 1458. We got these workflows from repositories of myGrid (http://workflows.mygrid.org.uk/repository/) and the examples distributed with Taverna.

We divide into our method to two parts, querying with virtual workflow and TF-IDF filtering. In order to validate our method, we had some experiments for the querying, filtering and combination of them.

### 4.1.1 Querying with Virtual Workflow

Table 1 shows the extraction result by querying with virtual workflow. It consists of Step1 and Step2. "correct" and "incorrect" show the correct/incorrect number in the extracted results. We had an experiment with leave-one-out to the all workflows. We checked the extracted result manually. When both virtual workflow (query) and the extracted workflow receive similar input data and generate similar output data respectively, we considered two workflows are candidates of the similar workflows. Then we checked whether the extracted workflow works alternatively on some level in a biological process. In this point, when we considered the extracted workflow as it generates almost same data with small changes, we judged the result is correct.

We could extract functionally similar workflows with 90.5% correctness. But there were still 15 incorrect results. Whether TF-IDF filtering can eliminate these incorrect results is important point.

Table 2: Correctness of Filtering

| | correct | incorrect | correctness(%) |
|---|---|---|---|
| (a) | 165 | 358 | 31.5 |
| (b) | 200 | 323 | 61.9 |

Table 3: Correctness of Combination

| | correct | incorrect | correctness(%) |
|---|---|---|---|
| virtual | 143 | 15 | 90.5 |
| combination | 128 | 1 | 99.2 |

### 4.1.2 TF-IDF Filtering

It consists of Step3. In order to validate filtering, we had an experiment with leave-one-out to the all workflows. Then we checked the extracted results manually. The criteria of the functional similarity are same as in querying with virtual workflow.

Result (a) in Table 2 shows that we could not filter the workflows well with the keyword obtained by TF-IDF. This problem was occurred because of a difference between the sizes of two workflows. There were many workflows that have similar parts around the obtained keyword. Therefore we changed criteria of the correct result in the result (b). When two workflows have similar parts around the keywords, we counted the pair as a correct result. Then the correctness improved to 61.9%. This result shows that the correctness will be higher if we compared similar size workflows. Therefore it is necessary to cut out small part from a workflow to use this filtering.

### 4.1.3 Combination

Table 3 shows the extraction results by querying with virtual workflow and the combination (our method). "virtual" indicates the method "Querying with Virtual Workflow". This result is same as Table 1. "combination" indicates the our method that is combination of querying and filtering.

As a result, we could extract workflows in terms of the function by names on workflow. We extracted functionally similar workflows in Step2. The correctness was 90.5% and there were still 15 incorrect results. Then, in Step3, we could eliminate 14 incorrect results with TF-IDF. After Step3, the correctness was improved to 99.2%.

### 4.1.4 Extraction Example

We show two examples of the result workflows. First we show a set of two workflows (Fig. 8). The query workflow was composed of the input and output terminals from the left one. The right one was an extracted workflow by our method. It shows the correct result extracted in Step2 and it also passed the Step3 successfully. The query workflow has a function to get MEDLINE_ID by Probeset ID. The extracted workflow has a function to get MEDLINE Record by Probeset ID. In this case, name of input terminals have same name between two workflows and name of output terminals "medlineid" and "medline" are similar. The query workflow gets "medlineid" with "probesetid" and extracted workflow gets "medline" with a tool "ProbeSetId". Data management on their workflow is almost same. Therefore we considered two workflows as functionally similar workflows, although two workflows do not have same function, extracted one gets "medlineid" on its process after getting "medline".

Table 4: TF-IDF keyword

| | query | extracted |
|---|---|---|
| $KEY(data)$ | test_input | test_output |
| $KEY(tool)$ | merge_string_list_to_string | split |

Second, we also show the two workflows extracted by our method (Fig. 9). The query workflow was composed of the input and output terminals from the left one. The right one was the extracted workflow. This result shows the incorrect result extracted in Step2, but the extracted workflow was eliminated in Step3. The query workflow has a function to merge two input strings. On the other hand, the extracted workflow has a function to get formatted BIND record and split it. BIND is one of the biology databases. In this case there were similar names of input terminals and output terminals on workflow (in this case, there were same names), but two workflows seemed to have opposite function (Marge and Split). Then the method checked TF-IDF keyword on workflows in Step3. Table 4 shows the keywords extracted with TF-IDF filtering. There were few similarity between those keyword, thus we could eliminate this extracted workflow correctly from results (each $KEY_{data}$ were little bit similar, but matching ratio were lower than threshold). This result was also a good result.

In order to help making new workflow, our method outputs figures and workflow descriptions of extracted workflows. As Taverna has a mechanism to use a workflow as a component of another workflow, users can build the extracted workflows into new workflows.

## 4.2 Discussion

Our method could extract workflows from the repository. Results were not limited to whole workflow in the repository. Our method extracted a part of whole workflow (subworkflow).

The correctness of the extracting seemed to have enough high-ratio. Thus there were functional meaning in the combination of names of input terminals and output terminals on workflows. But there were also eliminated workflows in spite of correct workflows. The correctness of the elimination (Step3) was 48.2% (14/29). This ratio was not so high. As a future work, we have to improve elimination in Step3 with TF-IDF or other algorithms.

# 5 Conclusion

We proposed a method to extract workflows from their repository that are similar to a query workflow. In order to extract the workflows, we used text matching of the names and the connection between tools. Then we improved correctness of the extraction with TF-IDF. The results seemed good enough for researchers to use extracted workflow as a component or for improvement to their research.

# References

[1] Addis, M., Ferris, J., Greenwood, M., Li, P., Marvin, D., Oinn, T. and Wipat, A.: Experiences with e-

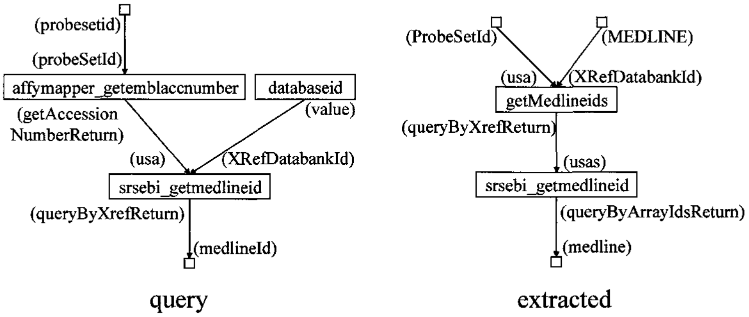


Figure 8: Query and extracted workflows (First result)

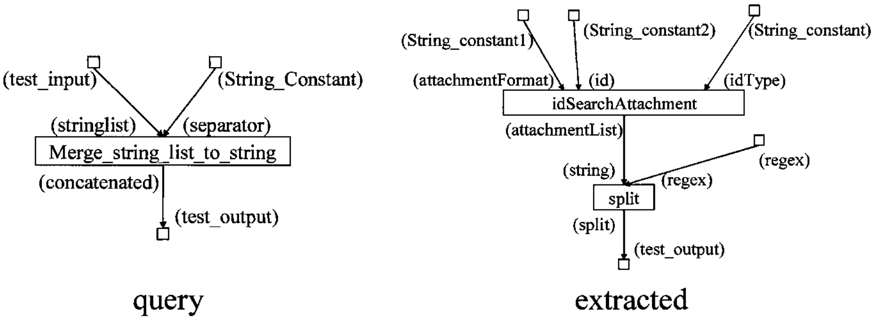


Figure 9: Query and extracted workflows (Second result)

Science workflow specification and enactment in bioinformatics, *Proceedings of UK e-Science All Hands Meeting 2003*, pp. 323–337 (2003).

[2] Altschul, S. F., Gish, W., Miller, W., Meyers, E. W. and Lipman, D. J.: Basic local alignment search tool, *J. Mol. Biol.*, Vol. 215, pp. 403–410 (1990).

[3] Bateman, A.: EDITORIAL, *Nucleic Acids Research*, Vol. 34 (2006). Database Issue.

[4] Bowers, S., Ludascher, B., Ngu, A. H. H. and Critchlow, T.: Enabling ScientificWorkflow Reuse through Structured Composition of Dataflow and Control-Flow, *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDEW'06)*, p. 70 (2006).

[5] Kawashima, S., Katayama, T., Sato, Y. and Kanehisa, M.: KEGG API: A new web service for accessing the KEGG database, *ISMB 2003* (2003).

[6] Medeiros, C. B., Perez-Alcazar, J., Digiampietri, L., G. Z. Pastorello, J., Santanche, A., da Silva Torres, R., Madeira, E. R. M. and Bacarin, E.: WOODSS and the Web: annotating and reusing scientific workflows, *SIGMOD Rec.*, Vol. 34, No. 3, pp. 18–23 (2005).

[7] Miyazaki, S. and Sugawara, H.: Development of DDBJ-XML and its application to a database of cDNA, *Genome Informatics 2000*, Vol. 11, pp. 380–381 (2000).

[8] Oinn, T. et al.: Taverna: a tool for the composition and enactment of bioinformatics workflows, *Bioinformatics*, Vol. 20, No. 17, pp. 3045–3054 (2004).

[9] Pearson, W. R. and Lipman, D. J.: Improved tools for biological sequence comparison, *Proc Natl Acad Sci. US*, Vol. 85, pp. 2444–2448 (1988).

[10] Salton, G. and Buckley, C.: Seven Bottlenecks to Workflow Reuse and Repurposing, *Information Processing and Management*, Vol. 24, pp. 513–523 (1988).

[11] Salton, G. and Buckley, C.: Term-weighting approaches in automatic text retrieval, *Information Processing and Management: an International Journal*, Vol. 24, pp. 513–523 (1988).

[12] Senger, M.: Bibliographic query service (2002). http://industry.ebi.ac.uk/openBQS/.

[13] Senger, M., Rice, P. and Oinn, T.: SoapLab - a unified Sesame door to analysis tools, *Proceedings of the UK e-Science All Hands Meeting*, Vol. 18 (2003).

[14] Shefter, S. M.: Workflow Technology: The New Frontier: How to Overcome the Barriers and Join the Future, *Lippincott's Case Management*, Vol. 11, pp. 25–34 (2006).

[15] Stein, L.: Creating a bioinformatics nation, *Nature*, Vol. 417, pp. 119–120 (2002).

[16] Wang, L., Riethoven, J.-J. and Robinson, A.: XEMBL: distributing EMBL data in XML format, *Bioinformatics*, Vol. 18, pp. 1147–1148 (2002).