Personal and Ubiquitous Computing (2021) 25:1091–1111
https://doi.org/10.1007/s00779-019-01276-3

ORIGINAL ARTICLE

Basic and personalized pattern-based workflow fragments discovery

Jinfeng Wen[1] · Zhangbing Zhou[1,2] · Fei Lei[3] · Junsheng Zhang[4]

Received: 15 March 2019 / Accepted: 30 May 2019 / Published online: 22 August 2019

Abstract
With an increasing number of scientific workflows accessible on public repositories, the mechanism for discovering and recommending workflow fragments is important to facilitate the reuse and repurposing of legacy workflows when novel workflows are to be constructed, where dependencies among workflow activities, which implies functional patterns with different types and characteristics, are specified in the specification of workflows. Traditional approaches ignore or seldom consider this aspect, which may have certain influence on the quality of personalized recommendation. To address this challenge, this paper proposes a novel workflow fragment discovery mechanism for personalized requirements, where discovery strategies of *basic* and *personalized* patterns are presented independently. Specifically, frequent basic subfunctions are discovered from scientific workflows by applying the frequent subgraph mining algorithm. Similar subfunctions are clustered considering their semantic relevance of topics, and clusters with high functional frequency are assumed as *basic patterns*. Thereafter, the multi-dimensional representation of scientific workflows is constructed to explore workflow relevance. Workflow clustering is conducted, and frequent personalized functions are discovered from clusters with similar workflows and assumed as *personalized* patterns under respective contents. For a personalized requirement given in terms of a workflow template, target basic patterns and candidate subfunctions are discovered, and they compose the backbone structure of solution in a novel coverage strategy. Candidate personalized patterns are applied to cover remaining functionalities of requirement. An optimal solution is obtained through atomic service optimization. Evaluation results show that this technique is accurate on discovering fragment solutions for personalized requirements in comparison with the state-of-the-art techniques.

Keywords Fragment discovery · Basic/personalized pattern · Coverage strategy · Personalized requirement

1 Introduction

With the mature and wide development of Web service technology, resources and processing functions are increasingly encapsulated as Web services and accessible on the Web. Effective approaches are needed urgently to recommend the useful Web services with respect to a customer's requirement [1, 2]. However, certain requirements to be solved have not been satisfied by single or multiple services. In this scenario, service composition is widely used to build complex value-added composite services to meet various coarse-gained requirements [3]. Thus, discovering relevant services as the constituents of composite services is a crucial task. In particular, a series of services compose into the format of scientific workflows in a certain order as shown in Fig. 1 [4, 5], which 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. Currently, a large number of scientific workflows are publicly accessible on the repository (e.g., *myExperiment* [6], *Crowdlabs*, and *Galaxy*); these repositories store thousands of workflows, which have been uploaded by scientists in many different domains (ranging from life science to text analytics or astronomy [7]). When novel scientific workflows are to be constructed, developing them from scratch is typically a knowledge- and effort-intensive and error-prone mission; reusing and repurposing mature *best practices* which have been evidenced by legacy scientific workflows in the repository is considered a cost-effective and error-avoiding strategy [8]. In fact, potential dependencies among

This work was supported by the National Natural Science Foundation of China (Grant nos. 61772479 and 61662021).

✉ Zhangbing Zhou
zbzhou@cugb.edu.cn

Extended author information available on the last page of the article.

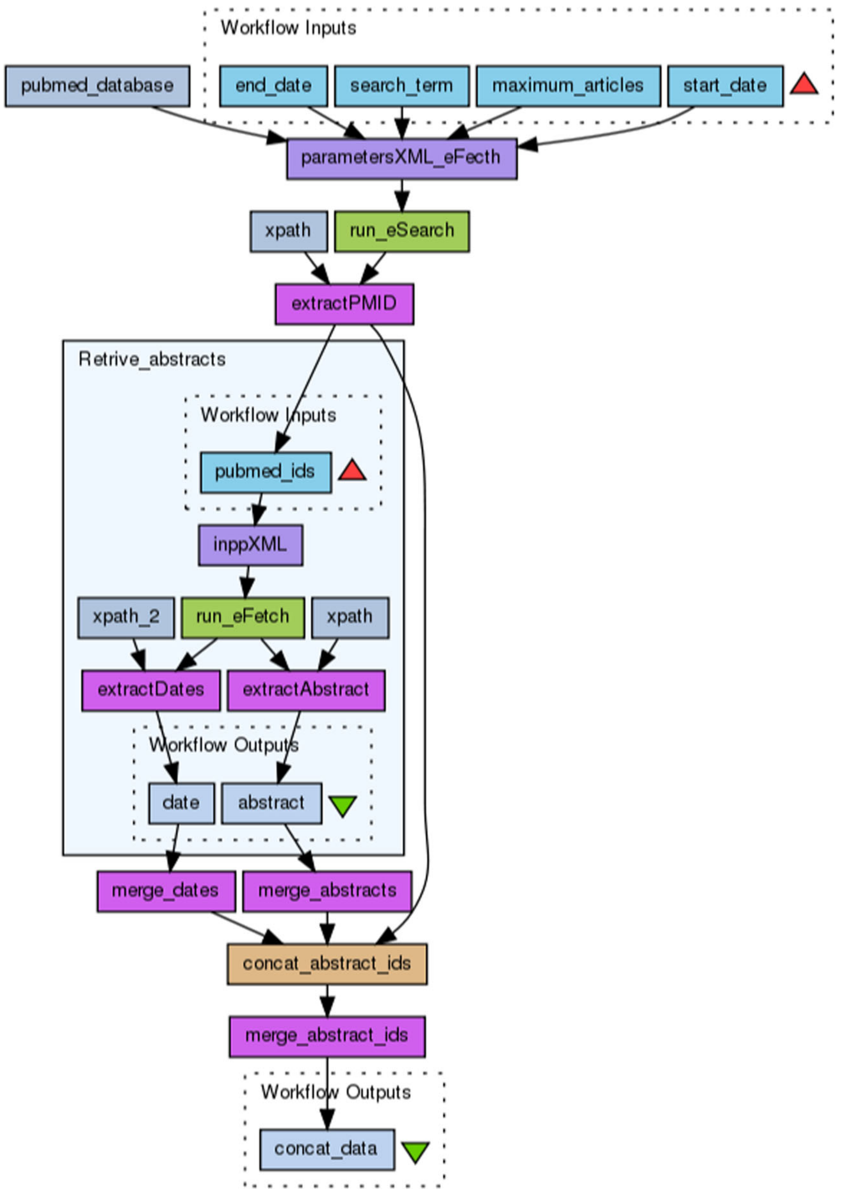


Fig. 1 A sample scientific workflow from *Taverna 2* of *myExperiment* repository with the title "*NCBI BLAST (SOAP)*" and the description "*Perform a BLAST search using ...*"

workflow activities are hidden in the specification of workflows, which have a great influence on the quality of the service composition and fragment recommendation.

Relevant researchers present service patterns to reflect these dependencies among services which are invoked together to accomplish the novel experiment. In the early stage of service computing research, a service pattern is regarded as a high-level abstract "template" such that it can be instantiated to different forms [9, 10]. Subsequently, it is realized that the frequently occurring fragments in historical service solutions are likely to appear in the follow-up ones; thus, mining patterns from history by data mining techniques (e.g., frequent itemset/subgraph mining) becomes a dominating approach [11–14]. There are two types of service pattern mining methods: top-down and down-up. The former identifies patterns from service process models, and the latter does from service execution logs. These methods both try to automatically find out repetitive process fragments, identify structurally and functionally similar fragments, and then combine them as patterns in a higher level of abstraction [13]. Therefore, understanding service patterns can promote the creation of service composition. Besides, it is believable that dependency information in service patterns is a kind of strong indicative hint for recommendation work.

Inspired by experiment functionalities in reality, it is observed that scientific workflows are mainly provided with three obvious characteristics: (i) *Generalization*: Services or composite services accomplish a certain goal that is always related with multiple basic scientific functions, e.g., data transformation and gene query. We call them basic functional patterns here. (ii) *Individuation*: Some personalized functions are required to achieve the corresponding specific goal, e.g., *KEGG* pathway insertion and chemical solubility prediction. We call them personalized functional patterns here. (iii) *Context-aware*: Text context or specification constitutes an essential part of workflows. When the context changes, service type and composite services may change. It can be seen that scientific workflows are represented in the format of the function-oriented, personalized way, and a service pattern does not exist independently but has to be bundled with the function context. Thus, the fragment recommendation study upon patterns should not focus on frequent patterns only. There are some differences among discovered patterns, such as the generalized component to reach the basic data transformation and the individuation component to obtain a certain special implementation. In this setting, discovering functional patterns with different types and characteristics is important to facilitate the recommendation of personalized requirements. Unfortunately, in most state-of-the-art approaches, this aspect is somehow ignored, and researchers seldom consider and focus on mining problems of different types of patterns for accomplishing novel and personalized scientific experiments. To address this problem, this article proposes a novel workflow fragment discovery mechanism for personalized requirements considering basic patterns and personalized patterns. The contribution of this article can be summarized as follows:

- The discovery strategy of basic patterns is presented, where frequent basic subfunctions are mined from scientific workflows through frequent subgraph mining method (*gSpan*). Semantic relevance of these subfunctions is quantified leveraging topics, which are generated by applying the biterm topic model (*BTM*) on their documents represented by contained service names. Community discovery clustering algorithm is adopted to explore basic functionalities, where subfunctions contained in a certain cluster with high functional frequency are assumed to solve the same or similar functionality thus can be abstracted as a *basic pattern*.
- Personalized patterns under different experiment contents are generated by workflow similarity and frequent subgraph mining. Specifically, the multi-dimensional representation for scientific workflows is constructed to express the workflow relevance from multiple aspects: titles in string, descriptions in pain-text, and contained services. Workflow similarity is calculated through representation vectors, and community clustering algorithm is applied on workflows. Frequent personalized functions discovered from a certain cluster with similar workflows are viewed as *personalized patterns* under the current experiment content, and they are retained in the corresponding personalized pattern container.
- For a given personalized requirement in the format of workflow fragment and its title and description, target basic patterns are selected considering functional similarity, and candidate basic subfunctions are discovered and ranked according to metrics: coverage ratio, redundancy ratio, and cost value. A novel coverage strategy is applied to generate the backbone structure of the solution. Thereafter, target personalized pattern container is obtained, where contained personalized patterns are filtered and applied to cover remaining functionalities of requirement. Consequently, the current solution is optimized by complementing atomic services, to generate the optimal solution with respect to this requirement.

Extensive experiments are conducted, and evaluation results demonstrate the accuracy of this technique on discovering fragment solutions for personalized requirements in comparison with the state-of-the-art techniques.

The rest of this article is organized as follows. Section 2 presents the discovery strategy of basic patterns. Section 3 explores the personalized pattern generation approach. Section 4 discovers and recommends the optimal fragment solution. Section 5 presents experimental settings, and Section 6 compares the evaluation results of this technique with the state of the art. Section 7 discusses related techniques, and finally, Section 8 concludes this article.

2 Basic pattern discovery

This section aims to discover basic functional patterns for accomplishing reuse of generalized functions. Generally, basic functional patterns represent recurrent valuable components, which are composed of services frequently invoked together in most of the experiments. Importantly, the appeared frequency of this kind of function is higher than individuation functions. For instance, 60∼70% experiments may all deliver common basic functionalities, such as information search and data transformation. Therefore, such frequent fragments (subgraphs) can be identified as basic functional patterns. However, since the habits from developers are different, some subfunctions are in fact functionally similar and can be interchangeable with each other to a certain extent. In this setting, similar subfunctions can be clustered together to represent the corresponding basic pattern. To conclude, the main processes of the basic pattern mining involve (i) frequency subfunction discovery, (ii) subfunction similarity calculation and clustering, and (iii) basic pattern discovery.

2.1 Frequency subfunction discovery

A service composition pattern documents a recurring solution to the same problem that happens over and over again. Thus, we first mine frequent functional subgraphs from all scientific experiments in the format of the directed graphs. *gSpan* [15] algorithm is applied, which is the state-of-the-art subgraph mining method for handling the directed graphs. It is noteworthy that there are several techniques developed in *gSpan*, including mapping each graph to a DFS code (a sequence), building a novel lexicographic ordering among these codes, and constructing a search tree based on this lexicographic order. For the detailed illustration, refer to [15]. Importantly, it combines the creation of new subgraph candidates with subgraph isomorphism testing.

Basic features of *gSpan* are summarized, since it is essential to focus on those for functional discovery. In *gSpan*, a graph represented as a linear sequence of its edges, called a DFS code, with each edge being represented as 5-tuple $(i, j, l_i, l_{i,j}, l_j)$ with i and j being the identifier numbers of its incident services (assigned by their order of appearance in the sequence) in a certain workflow graph, l_i and l_j being their service labels and $l_{i,j}$ being the label of the edge. Note that services with same name and description are considered to belong to the same service label. In our experiments, services are only specified as the invocation relation. Thus, edge labels default to 0 uniformly. Not every possible edge sequence forms a valid DFS code but only those representing a depth-first traversal of a workflow graph. The DFS code representation of a workflow graph is not unique, but a linear order on DFS codes is defined by the insertion position of the edges into the sequence and by the lexicographic order of the label types. The smallest DFS code representation of a workflow graph is defined as its minimum DFS code, which is unique and used as canonical form for the mining process. The search space is then defined as a DFS code tree consisting of nodes that represent the DFS codes and edges that indicate that the child node grows from the parent node by adding one new edge at the end of its DFS code. Yan and Han [15] also show that each minimum DFS code is the child of another minimum DFS code so that the search space for frequent subgraph mining can be pruned wherever non-minimum DFS codes occur. Anyway, the problem of mining frequent subgraphs is equivalent to mining their corresponding minimum DFS codes.

Specifically, *gSpan* uses a sparse adjacency list representation to store workflow graphs. The discovery process of frequent subgraphs is explained as follows:

- Step 1: Sort the labels in the graph dataset D (referred to scientific workflow repository) by their frequency of vertices and edges.
- Step 2: Remove infrequent vertices and edges.
- Step 3: Relabel the remaining vertices and edges and arrange them in descending order.
- Step 4: Leverage the set S^1 to save all frequent one-edge graphs in D, and sort S^1 in DFS lexicographic order, which is a linear order.
- Step 5: Store elements in S^1 to result set S.
- Step 6: Iterate unilateral edge e in S^1. Execute step 7 and step 8.
- Step 7: Initialize the subgraph s with e. The symbol D_s means the set of graphs in which s is a subgraph.
- Step 8: Mine frequent subgraphs for s through the $Subgraph\_Mining(D, S, s)$ function. This method is recursively called to grow the graphs and find all their frequent descendants. It stops searching when its code is not a minimum DFS code, which means the graph and all its descendants have been generated and discovered before.

To sum up, after initial identification of frequent and minimum one-edge workflow subgraphs, the search space is traversed in a depth-first manner, and for each frequent and minimum DFS code encountered during the traversal, all its occurrences in the database are accessed for support computation of its candidate extensions. Whenever an extension turns out to be infrequent or not minimal, it need not be accessed for further extension but can be pruned from the search space.

In this phase, the frequency threshold is not set, which is theoretically greater than 1 to discover all frequent subgraphs with their actual supports. Importantly, this strategy contributes to avoiding the loss of subfunction generation. It is worth noting that the number of services is essential in a frequent functional subgraph. A functional subgraph with very few services (e.g., 2 services) may provide incomplete functionality for reuse. We propose that a functional subgraph contains at least 3 services, which can accomplish basic data functions in our scenario. Note that the required number of services may vary in different settings, and it can be set according to specific demands. Some discovered subfunctions are shown in Table 1, which contains *subgraph ID*, *vertex information*, *edge information*, and *subgraph support*. For instance, there are three vertices and two edges in *subgraph 1*, where (*v 0 3*) represents that the service label of a vertex, whose identifier number is 0, is service 3 while (*e 0 1*) denotes that this edge points from 0 to 1 in terms of identifier numbers in this graph. In fact, edge information implies the connection between services. We also obtain the corresponding support 10 for *subgraph 1*.

Particularly, a scientific workflow represents an experiment as shown in Fig. 1. According to our previous work [4, 5], a scientific workflow can be defined through Definition 1.

Definition 1 (Scientific workflow) A scientific workflow swf is a tuple $(tl, des, S, SWF_{\text{sub}}, E)$, where:

- tl is the title of swf.
- des is the description information of swf.
- S is a set of services contained in swf.
- SWF_{sub} is a set of subworkflows contained in swf.
- E is a set of control flows on S.

Table 1 The structure illustration of mined subfunctions

Subgraph ID	1	2	...	5
Vertex information	v 0 3 v 1 1 v 2 4	v 0 3 v 1 7 v 2 4 v 3 8	...	v 0 1406 v 1 1408 v 2 1407 v 3 1409 v 4 1410
Edge information	e 0 1 e 1 2	e 0 1 e 0 3 e 1 2	...	e 0 1 e 1 2 e 2 3 e 3 4
Subgraph support	10	5	...	6

In fact, a subworkflow, such as "*Retrive_abstracts*" in Fig. 1, can be regarded as a service (function) with relatively *coarse* granularity, which corresponds to the specific service implementation. Note that the internal implementation in a certain subworkflow is treated as a small directed graph, and if it satisfies the defined condition (i.e, at least 3 services), it is also assumed to be a subfunction. In this situation, frequent subfunctions with different granularities are discovered.

2.2 Subfunction similarity calculation and clustering

Although the naming and structure of subfunctions may be slightly different due to habits of developers, they are actually similar in functionalities. Thus, functionally similar subfunctions are considered to solve the same or similar problem. In this setting, the semantic relevance of these frequent subfunctions is quantified through the similarity evaluation mechanism. Services contained in every subfunction own their name and text description. Importantly, the name, which is usually represented in terms of the conjunction of very few carefully selected keywords or their abbreviations, should contain more accurate information than the words in relatively long text description. In this setting, the document description of every subfunction can be constructed leveraging names of contained services. Based on subfunction documents, semantic relevance of subfunctions is explored through the format of topics, which are generated through applying the biterm topic model (*BTM*) [16] on the document corpus of subfunctions. The specific procedure is illustrated as follows:

- Phase 1: *Document digitization.* Documents of subfunctions are digitized and transformed to required format of *BTM*.
- Phase 2: *Biterm extraction.* Every subfunction document is treated as a separate text fragment. Any pair of distinct words is extracted as a biterm, and these biterms are treated as the training dataset of topic learning.
- Phase 3: *Topic learning.* The corpus can be regarded as a mixture of topics, where each biterm is drawn from a specific topic independently.
- Phase 4: *Document topic inferring.* Since topic proportions for subfunction documents cannot be discovered directly by applying the topic learning, the expectation of topic proportions of biterms is utilized to infer topics for each subfunction.

In particular, the number of topics is a sensitive factor for the topic extraction. Authors select perplexity as the criterion for evaluation the model [17]. Generally, a smaller value of perplexity determines the optimal number of topics, and which supposes to generate a better predictive effect and generalization ability. Tested topic number and the corresponding perplexity are shown in Table 2. In our corpus of subfunction documents, when the number of topics is set as 360, it produces the smallest perplexity 14.776. Thus, the optimal number is determined as 360 in our current experiments.

After obtaining the topic distribution of subfunction documents st, the distance can be calculated between a pair of subfunctions st_i and st_j through adopting *Jensen-Shannon* (*JS*) divergence in formula (1):

$$D_{\mathrm{JS}}(st_i, st_j) = 0.5 \times D_{\mathrm{KL}}(st_i, (st_i + st_j)/2) + 0.5 \times D_{\mathrm{KL}}(st_j, (st_i + st_j)/2) \quad (1)$$

where D_{KL} denotes the *Kullback-Leibler* divergence. D_{JS} returns a value between 0 and 1, where 1 means totally functionally different and 0 means the equivalent. Leveraging the distances between subfunctions, a network can be constructed where vertices correspond to subfunctions, and the weight upon edges specifies the distance between a certain pair of subfunctions. Subfunctions with slight functional relevance are far away from each other in the network, which is useless for similarity exploration. Thus, the edge between subfunctions whose distance exceeds a certain distance threshold should be pruned. The edges between functionally similar subfunctions, whose distances are relatively small in value (i.e., they are closely related), are retained in the network. Community structure is formed; thus, a modularity-based clustering algorithm *Louvain* [18] is applied to discover clusters with similar functionalities. Particularly, the density of links within communities as

Table 2 Tested topic number and the corresponding perplexity

Topic number	300	310	320	330	340	350	*360*	370	380
Corresponding perplexity	15.098	15.071	15.023	14.987	14.956	14.950	*14.776*	14.874	14.946

The italicized data are used as examples in the article to explain compared with links between communities can be measured leveraging modularity optimization. The specific process of *Louvain* is explained:

1. Scan all subfunctions in the network, traverse the neighbor subfunctions for each subfunction, and measure the modularity gain brought by adding the subfunction to the community where the neighbor is located. Select the neighbor subfunction corresponding to the maximum revenue and join the community in which it is located. This process repeatedly guides each subfunction until the community does not change anymore.
2. Fold the communities formed in the above step into points, and calculate the border weights between newly generated "community points" and the border weights between all points in the community. Continue the previous step for the next round.

The determination of the distance threshold is related to the modularity value obtained by *Louvain*. Test multiple distance thresholds to generate different subfunction-based networks, then apply *Louvain* to them. Consequently, the threshold and network with the optimal modularity are determined together. In our case, the distance threshold is determined as 0.25, and 22 clusters are generated in the network where the optimal modularity is 0.388.

2.3 Basic pattern discovery

The discovery of functionally similar subfunctions allows us to present functional patterns with a higher level of abstraction and reflect basic functionalities of services and their control flows. The valuable clusters are presented to solve the same or similar functional problem, called *basic patterns* in our scenario. The functional frequency representing the cluster significant is calculated by the sum of subfunction supports contained in the corresponding cluster. Therefore, clusters are ranked according to the functional frequencies, and top K_{bp} (e.g., 40% = 0.4) clusters are selected as basic patterns (BP). The most frequent subfunction is selected as the representative subfunction for each basic pattern, which is considered to prepare the basic pattern recommendation in Section 4. The definition of a basic pattern (bp) is illustrated as follows:

Definition 2 (Basic pattern) A basic pattern bp is a tuple (sf_{rpt}, SF), where:

- sf_{rpt} is the representative subfunction of bp .
- SF is the set of similar subfunctions of bp .

Note that each subfunction in SF is a tuple (S, E), where S is the set of services and E represents a set of their control flows on S.

3 Personalized pattern discovery

Personalized subfunctions are related with the designed content; similar scientific experiments usually reuse certain same and specific subfunctions. In this setting, we aim to mine the personalized patterns from a group of workflow experiments with same or similar contents, where relevance of workflows should be explored considering multiple aspect information. Thus, the personalized pattern discovery mainly involves two parts: (i) multi-dimensional representation generation of workflows and (ii) personalized pattern discovery.

3.1 Multi-dimensional representation generation of workflows

As shown in Fig. 1, the title, description, and contained services are basic information of scientific workflows. To better utilize them, workflows are represented in a novel way from different significant aspects.

Aspect 1: *Workflow-topic distribution.* The title of workflows is composed of a few keywords, which actually express its designed meaning. Due to the briefness of title, it may hardly convey complete information. Thus, considering the topic distribution on biterms in the corpus based on short documents, the above problem can be addressed at the document-level through *BTM*. In this setting, topics are learned to generate workflow-topic distribution (wt), which means a group of correlated words are used to represent the title information in another way. Note that the current topic number is also determined (e.g., 370) like the same way in Section 2.2. Consequently, the workflow-topic distribution is generated, and an example is shown in Table 3, where the row of this distribution represents the identifier number of workflows, and the column represents the identifier number of topics. tn in Table 3 denotes the number of topics while N denotes the number of workflows in our experiment. Specially, wt_{ik} expresses the probability value of the kth topic in the ith workflow, i.e., $wt_{ik} = 1.3e{-}05$ in Table 3. In this setting, the similarity (denoted $Sim_{\mathrm{JS}}(wt_i, wt_j)$) between two workflows swf_i and swf_j can be calculated in terms of topics through formula (2), where bases on *JS* divergence in formula (1). When Sim_{JS} trends to 1, it illustrates the more similar between swf_i and swf_j.

$$Sim_{\mathrm{JS}}(wt_i, wt_j) = 1 - D_{\mathrm{JS}} \tag{2}$$

Aspect 2: *Workflow-feature distribution.* The redundancy of description makes the information expression unclear. Some words are even useless to understand its intention. Therefore, how to utilize the more valuable information to enhance the expression of workflow descriptions is

Table 3 Workflow-topic distribution (wt)

swf \ Topic	t_1	...	t_k	...	t_{tn}
swf_1	$5.9e-07$	...	$2.5e-07$	...	$4.3e-07$
...	...	...	...	...	...
swf_i	$1.0e-05$	...	*1.3e − 05*	...	$7.6e-06$
...	...	...	...	...	...
swf_N	$4.9e-06$	...	$2.1e-06$	...	$6.6e-07$

The italicized data are used as examples in the article to explain

becoming an important issue. In this setting, we extract keywords efficiently from the descriptions. When the term is assigned its weight, the more important this word in the description, the greater the weight of words. The *TF-IDF* [19] method is adopted to select feature words. This method bases on the bag-of-word scheme in which a document can be represented by a collection of words used in the document. *TF-IDF* also assumes that if a word is important for a document, it should repeatedly appear in that document whereas it should rarely appear in other documents. The *TF* is associated with the former assumption whereas the *IDF* is associated with the latter assumption. The parameter tf_{ij} is defined as the number of times word i appears in description document j; the larger the value, the more important the word is. The parameter df_i is the number of description documents in which word i appears at least once; the larger the value, the more common the word is. If word i can be considered important for description document j, it should have a large tf_{ij} and a small df_i. For example, words such as "workflows" and "show" would frequently appear in each description document. However, they cannot be considered important words because they are prevalent in all documents. Hence, *TF-IDF* is defined as follows:

$$\begin{aligned} TF-IDF_{ij} &= tf_{ij} \times idf_i \\ &= tf_{ij} \times log(N/(df_i+1)) \end{aligned} \tag{3}$$

where the *IDF* takes the logarithm of the ratio of the number of total description documents in the corpus to the description document frequency of word i with *IDF* smoothing to prevent it from being divided by zero, i.e., $idf_i = log(N/(df_i+1))$.

Each word located in description documents calculates the corresponding value of *TF-IDF* according to formula (3). Given a certain description document, their scores of words are ranked, and words with top 20% high *TF-IDF* scores are selected as representative features. Note that the top 20% words selected are appropriate to express the actual semantic of each description document through analyzing information accuracy in our experiments. Features selected from description documents are loaded into a feature set. The word frequency of each feature is calculated to generate the corresponding word frequency vector for each description document. Consequently, a workflow-feature distribution (wf) is constructed by respective vectors as shown Table 4, where the row represents the identifier number of workflows, and the column represents the identifier number of features. fn denotes the number of features (e.g., 403) while N denotes the number of workflows in our experiments. wf_{ik} expresses the frequency of the kth feature in the ith workflow, i.e., $wf_{ik} = 2$ in Table 4. In particular, the similarity (denoted $Sim_{\mathrm{ED}}(wf_i, wf_j)$) between two workflows swf_i and swf_j can be calculated leveraging *Euclidean distance* and the corresponding transformation in terms of features in formula (4). Sim_{ED} returns a value between 0 and 1, where 1 means totally equivalent and 0 means different.

$$Sim_{\mathrm{ED}}(wf_i, wf_j) = \frac{1}{1+\sqrt{\sum_{k=1}^{fn}(wf_{ik}-wf_{jk})^2}} \tag{4}$$

Aspect 3: *Workflow-service distribution.* Obviously, services play an important role in scientific experiments. We intend to establish workflow-service distribution (ws) for evaluating the similarity of workflows and show in Table 5, where the row represents the identifier number of workflows, and the column represents the identifier number of services. sn denotes the number of services (e.g., 609) while N denotes the number of workflows in our experiments. ws_{ik} expresses whether the ith workflow contains the kth service, i.e., $ws_{ik} = 1$ in Table 5. ws_{ik} is set to 1 when the kth service exists in the ith workflow, otherwise 0. The similarity (denoted $Sim_{\mathrm{ED}}(ws_i, ws_j)$) between two workflows swf_i and swf_j is calculated in terms of contained services through formula (4).

Based on the similarity evaluation of distributions from various aspects, the eventual similarity of a pair of workflows swf_i and swf_j is calculated through formula (5). When $Sim(swf_i, swf_j)$ trends to 1, it represents the more similar between two workflows.

$$\begin{aligned} Sim(swf_i, swf_j) &= \alpha \times Sim_{\mathrm{JS}}(wt_i, wt_j) \\ &+ \beta \times Sim_{\mathrm{ED}}(wf_i, wf_j) + \gamma \times Sim_{\mathrm{ED}}(ws_i, ws_j) \end{aligned} \tag{5}$$

Table 4 Workflow-feature distribution (wf)

swf \ Feature	f_1	...	f_k	...	f_{fn}
swf_1	0	...	4	...	0
...	...	...	...	...	...
swf_i	1	...	*2*	...	0
...	...	...	...	...	...
swf_N	0	...	0	...	2

The italicized data are used as examples in the article to explain

Table 5 Workflow-service distribution (ws)

swf \ Service	s_1	...	s_k	...	s_{sn}
swf_1	1	...	0	...	1
...	...	...	...	...	...
swf_i	0	...	*1*	...	0
...	...	...	...	...	...
swf_N	1	...	1	...	0

The italicized data are used as examples in the article to explain

where the sum of α, β, and γ equals to 1, and these factors reflect the importance of different aspects. In our scenario, topics and services have relatively higher significance than features. Adopting multiple similarity testing, when α and γ are both set as 0.4 while β is set as 0.2, actually, the relevance of workflows is relatively accurate. Thus, the optimal parameters are determined in our experiments, and they can be adjusted according to other data or experiment applications. In fact, similar workflows are closely related to each other in terms of distance and show the community structure. However, for irrelevant workflows, their distances are far and similarities are small, which are useless for discovering similar workflows. Thus, define a similarity threshold and prune such connected links. Similarly, we test multiple variable similarity thresholds to detect which one can obtain a workflow-based network with the maximum modularity after applying *Louvain*. Consequently, a network is generated, and a series of clusters are discovered. For instance, 0.35 is the optimal pruned threshold, and 23 workflow clusters are obtained when the optimal modularity of network is 0.809, which synthesizes the overall structural relevance to balance the entire situation of the clustering process.

3.2 Personalized pattern discovery

In this section, we intend to discover personalized patterns separately from each cluster with the specific content and similar scientific experiments. The *gSpan* algorithm is the best way to handle the directed graphs. For each cluster, the corresponding frequency threshold is determined by the factor $thrd_{\text{frq}}$ (e.g., 30% = 0.3), i.e., $thrd_{\text{frq}} \times |SWF_{\text{cluster}}|$, where $|SWF_{\text{cluster}}|$ is the number of workflows contained in the given cluster. In particular, when the number of workflows in a certain cluster is relatively small, frequency threshold may be less than 2, which is meaningless to discover frequent personalized patterns. Thus, in order to guarantee that the mined personalized subgraphs will appear at least twice, the minimum value of frequency threshold is set to 2.

In addition, we need to select the corresponding representative workflows from these clusters, which contribute to find similar scientific workflows for a certain requirement in the recommendation phase. According to the workflow similarities, a workflow that makes the important connection can be found compared with the average similarity of one workflow to other workflows in a certain cluster. The workflow with the largest average similarity is assumed as the representative workflow corresponding to this cluster. After the above process, personalized patterns (PP) are generated under different contents and saved in respective personalized pattern containers (PPC). Particularly, the definition of a personalized pattern container (ppc) is illustrated under a certain content as follows:

Definition 3 (Personalized pattern container) A personalized pattern container ppc is a tuple (swf_{rpt}, PP), where:

- swf_{rpt} is the representative workflow of ppc.
- PP is a set of personalized patterns of ppc.

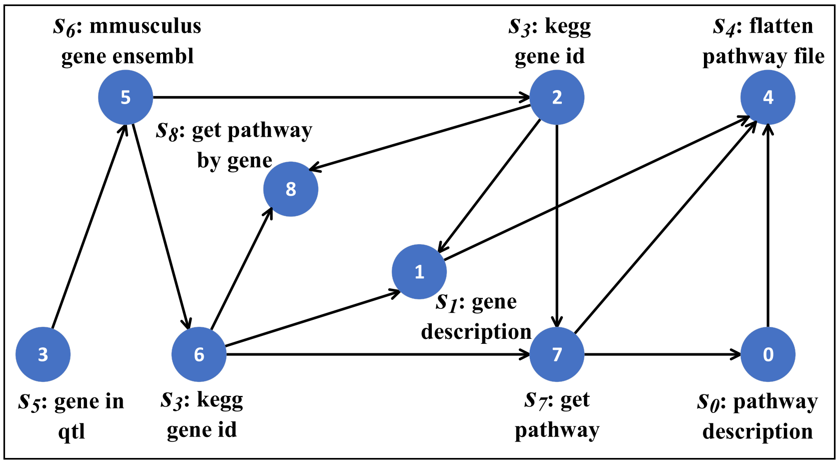
Information of swf_{rpt} is represented as multiple dimensional vectors. PP shows a group of discovered personalized patterns under the similar experiments and contents.

4 Pattern-based workflow fragment discovery

This section presents our pattern-based fragment discovery approach for personalized requirements considering basic patterns and personalized patterns. In fact, pattern can be viewed as a big granularity service entity and represents a service process fragment that is frequently used and proven by applications. We use different types of service patterns as the fundamental units to service composition to obtain service solution more quickly and efficiently with respect to certain personalized requirements. For a given requirement in the format of workflow fragment and its title and description, select a set of basic subfunctions, personalized patterns, and atomic services from candidate service sets to generate an optimal service solution. A sample requirement Req_{sampl} is shown in Fig. 2, where 9 services cooperate to accomplish this scientific experiment.

4.1 Preliminary knowledge

A requirement fragment can be abstracted to a directed graph. Similarly, a service pattern and an atomic service can be abstracted to a directed subgraph and a node, respectively. We call that a service pattern can cover a requirement fragment, if and only if there is overlap between the subgraph representing the service pattern and the graph representing the requirement fragment, which means the service pattern can implement the



s_6: mmusculus gene ensembl
s_3: kegg gene id
s_4: flatten pathway file
5
2
4
s_8: get pathway by gene
8
1
s_1: gene description
3
6
7
0
s_5: gene in qtl
s_3: kegg gene id
s_7: get pathway
s_0: pathway description

Fig. 2 A sample requirement with the title "*Pathways and gene annotations for QTL region*" and the description "*This workflow searches for genes ... KEGG gene identifiers...*"

functionalities of the requirement fragment partly or completely, and the mapping relationship between service activities in the overlapped part of the service pattern and the requirement fragment is identical. Thus, pattern-based service composition approach is attributed to a kind of covering problem. We use three performance indicators to evaluate the coverage degree and coverage effectiveness, i.e., coverage ratio, redundancy ratio, and cost value.

The coverage ratio of a certain pattern $cvr(p_i, Req)$ is the ratio between the number of services $|CS_{p_i}|$ that service pattern p_i can cover the service process of Req and the total number of services $|S_{p_i}|$ of service pattern p_i. Thus, it is calculated through formula (6):

$$cvr(p_i, Req) = |CS_{p_i}|/|S_{p_i}| \tag{6}$$

where the result returns a value between 0 and 1. Coverage ratio implies to what degree that a service pattern meets the requirement. For instance, Fig. 3 has two sample service patterns p_1 and p_2. For p_1, the structure of $\{s_6 \text{ -> } s_3\}$ can satisfy the part of the given requirement in Fig. 2. Thus, $cvr(p_1, Req_{sampl})$ equals to 2/3 = $\frac{2}{3}$ in this situation. For p_2, the structure of this service pattern is covered to Req_{sampl} as a whole, so $cvr(p_2, Req_{\text{sampl}})$ equals to 4/4 = 1.

The redundancy ratio $rdd(p_i, Req)$ is the ratio between the number of atomic services in service pattern p_i which are useless to cover the service process and the total number $|S_{p_i}|$ of atomic services in p_i. rdd is calculated for a certain pattern as follows:

$$rdd(p_i, Req) = |S_{p_i} - CS_{p_i}|/|S_{p_i}| \tag{7}$$

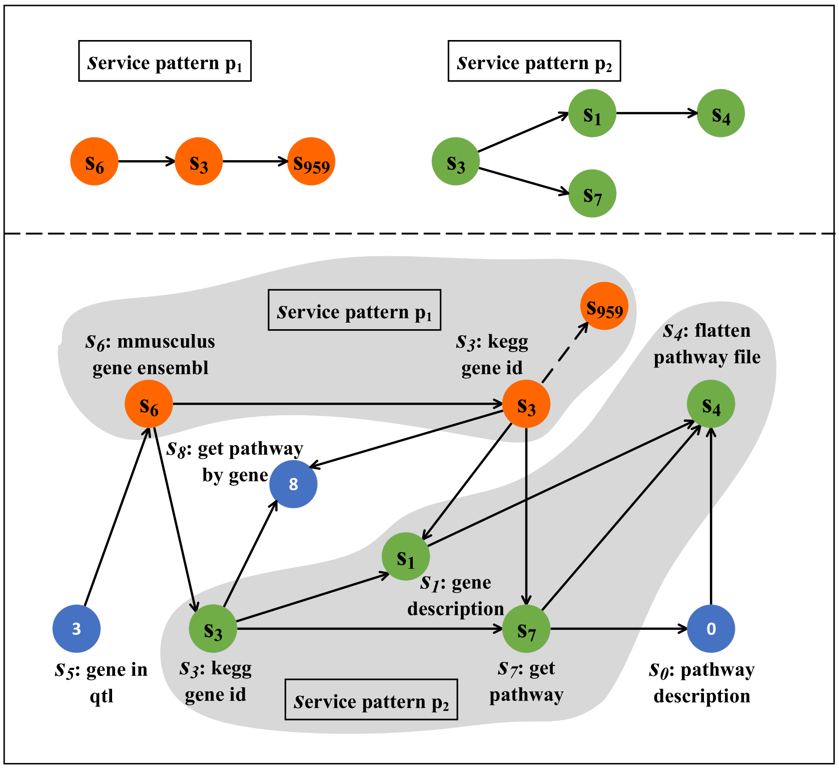


Fig. 3 An example of service patterns covering the sample requirement in Fig. 2

where result returns a value between 0 and 1. In some case, not all atomic services in a service pattern are able to cover requirement fragment; redundancy ratio of a service pattern is used to measure the redundancy degree. Service pattern works as a whole whose redundant atomic services cannot be cut down; otherwise, it will affect the whole service pattern. For instance, the redundancy of p_1 can be calculated as $rdd(p_1, Req_{\text{sampl}}) = (3 - 2)/3 = \frac{1}{3}$ in Fig. 3. Since the whole structure of p_2 is used, its redundancy ratio is $rdd(p_2, Req_{\text{sampl}}) = (4 - 4)/4 = 0$.

In service composition, we'd better choose the service patterns which have lower redundancy ratio and higher coverage ratio. Thus, we define the cost value rdd_cvr of each pattern leveraging rdd and cvr through formula (8):

$$rdd\_cvr(p_i, Req) = rdd(p_i, Req)/cvr(p_i, Req) \quad (8)$$

where rdd_cvr is relatively small in value; the corresponding service pattern is assumed to be optimal. For instance, the cost value of p_1 is calculated as $rdd\_cvr(p_1, Req_{\text{sampl}}) = \frac{1}{3}/\frac{2}{3} = \frac{1}{2}$ in Fig. 3, while $rdd\_cvr(p_2, Req_{sampl}) = 0/1 = 0$. Note that $cvr(p_i, Req)$ is guaranteed not to be 0 in our experiments; we filter out patterns without services contained in the requirement, which can accelerate the pattern selection.

The above metrics are the measure of a certain pattern. In a similar way, we also establish evaluation metrics for the solution being constructed Sol_{now}, whose coverage ratio is obtained compared with a certain requirement Req. Thus, $cvr(Sol_{now}, Req)$ reflects the ratio between the number of services that the current solution Sol_{now} can cover the Req and the total number of services contained in Req. Assume that only two patterns p_1 and p_2 in Fig. 3 are selected to generate the current solution for Fig. 3, $cvr(Sol_{now}, Req_{sampl}) = 6/9 = \frac{2}{3}$. The redundancy ratio $rdd(Sol_{now}, Req)$ shows the ratio between the number of atomic services in Sol_{now} which are useless to cover the Req and the total number of atomic service in Sol_{now}. For instance, $rdd(Sol_{now}, Req_{\text{sampl}}) = 1/7 = \frac{1}{7}$ in Fig. 3. According to $cvr(Sol_{now}, Req)$ and $rdd(Sol_{now}, Req)$, the cost value of the current solution being constructed is calculated by $rdd\_cvr(Sol_{now}, Req) = rdd(Sol_{now}, Req)/cvr(Sol_{now}, Req)$. In this setting, the current solution is calculated as $rdd\_cvr(Sol_{now}, Req_{\text{sampl}}) = \frac{1}{7}/\frac{2}{3} = \frac{3}{14}$.

4.2 Basic pattern recommendation

This section aims to select appropriate subfunctions contained in different basic patterns, to construct the backbone structure of the corresponding solution for the given requirement. Firstly, candidate basic patterns and corresponding subfunctions are determined according to the fragment similarity evaluation. Then, our novel coverage strategy is designed to generate the backbone structure of the solution with respect to the given requirement. The procedure is explained as follows:

Phase 1: *The basic pattern discovery.* The requirement may involve several generalized functionalities; basic patterns are acquired to discover and reuse. In this setting, leveraging the representative subfunctions sf_{rpt} of basic patterns, functional similarities to a certain partial function in the requirement can be calculated in order to select the candidate representative subfunctions set. The specific strategy is designed, the functional similarity between any sf_{rpt} and Req is obtained leveraging the respective contained services. Each service in each sf_{rpt} calculates service similarities with services in Req, in order to obtain the most similar service in Req and form the service mapping in terms of semantic relevance of services. Note that service similarity is calculated by comparing their name similarity, which adopts the minimum cost and maximum flow algorithm where *WordNet* is used calculate the semantic similarity for words in English. The average functional similarity of every sf_{rpt} is got through service mappings; thus, we can infer the correlation of sf_{rpt} and the (partial) requirement. Set the threshold to select sf_{rpt}; in our experiments, the average of these functional similarities about all sf_{rpt} is calculated and determined as the selected threshold. Traverse each sf_{rpt}; if whose similarity exceeds the threshold, this sf_{rpt} is retained and the corresponding basic pattern is chosen as target candidates.

Phase 2: *The basic subfunction discovery and ranking.* According to candidate basic patterns, contained subfunctions are added to a temporal basic subfunction set. Note that we need to remove subfunctions that do not contain any covered services in Req from this temporal set, since their coverage degree of such subfunctions is zero which does not make any sense to our combination plan. Consequently, the satisfied basic subfunctions are saved in BFS. The coverage ratio, redundancy ratio, and cost value of subfunctions in BFS are calculated, and they are ranked according to their cost value. First and foremost, we present the metric calculation process in Algorithm 1, where Req is a given requirement in format of the directed graph. The main idea is these relevant metrics are calculated leveraging the mapping relation between a certain subfunction p and Req. In detail, obtain the edge set $P.E$ from p and handle each edge $pedge$. The information of $pedge$ is represented through identifier numbers in p; thus, it is required to transform into the edge in terms of service labels (lines 3–6). The source and sink nodes with the identifier number $pidf_{\text{src}}$ and $pidf_{\text{snk}}$ are separated from $pedge$ (lines 3 and 4). By using the search function $getSvcLbl$, service labels of $pidf_{\text{src}}$ and $pidf_{\text{snk}}$ are found in order to match partial fragments of Req (lines 5 and 6). According to the edge set $Req.E$ of the given requirement, judge whether the service relation edge $edge(psvc_{\text{src}}, psvc_{\text{snk}})$ in p has mapping structures compared with Req (line 7). If it exist, establish its corresponding mapping set $NdSet$, where $redge$ is the matched edge in edge set $Req.E$ of Req, and matched source and sink nodes with the identifier number in Req are added into respective sets ($NdSet_{pidf_{\text{src}}}$ and $NdSet_{pidf_{\text{snk}}}$) (lines 8 and 9). Note that a certain node in p may exist many mapping nodes of Req, since Req may be composed of multiple same service labels. In this setting, the generated multiple mapping structures are equally considered to the next composition phase. After accomplishing the discovery of mapping pairs of p, the coverage ratio, redundancy ratio, and cost value are calculated (lines 12–14). $|p.map.keySet|$ represents the node number in the mapping set of p. In addition, $|p.N|$ denotes the node number in p. When p does not produce any mapping relation, which its coverage ratio will be 0 even if p exists relevant services, such subfunctions will be removed by us. Consequently, satisfied subfunctions with metrics are retained in a new set BFS_{new} (lines 15–17).

Algorithm 1 Subfunction metric calculation.

```
Require:
    - $BFS$: a set of candidate basic subfunctions
    - $Req$: a certain requirement specification
Ensure:
    - $BFS_{new}$: a set of redetermined subfunctions with their $cvr$, $rdd$, $rdd\_cvr$ and mapping pairs with respect to $Req$
1:  for all $p \in BFS$ do
2:      for all $pedge \in P.E$ do
3:          $pidf_{src} \leftarrow pedge.src$
4:          $pidf_{snk} \leftarrow pedge.snk$
5:          $psvc_{src} \leftarrow getSvcLbl(p.N, pidf_{src})$
6:          $psvc_{snk} \leftarrow getSvcLbl(p.N, pidf_{snk})$
7:          if $IsExst(Req.E, edge(psvc_{src}, psvc_{snk}))$ then
8:              $p.map \leftarrow NdSet_{pidf_{src}} \cup \{redge.src\}$
9:              $p.map \leftarrow NdSet_{pidf_{snk}} \cup \{redge.snk\}$
10:         end if
11:     end for
12:     $p.cvr \leftarrow |p.map.keySet|/|p.N|$
13:     $p.rdd \leftarrow |p.N - p.map.keySet|/|p.N|$
14:     $p.rdd\_cvr \leftarrow p.rdd / p.cvr$
15:     if $p.cvr \neq 0$ then
16:         $BFS_{new} \leftarrow BFS_{new} \cup \{p\}$
17:     end if
18: end for
19: $BFS_{new} \leftarrow$ a new set of subfunctions with their $cvr$, $rdd$, $rdd\_cvr$ and mapping pairs with respect to $Req$
```

Phase 3: *The backbone structure generation of solution.* According to BFS_{new} obtained from Algorithm 1, the pattern-based coverage strategy is presented to generate the current solution with the minimum cost value leveraging Algorithm 2. Particularly, given the candidate basic subfunction set BFS_{new} with the respective coverage ratio, redundancy ratio, and cost value, subfunctions in BFS_{new} are first ranked according to their rdd_cvr in ascending order (line 1). Next, we need initiate solutions to facilitate the coverage expansion with respect to Req. Since we have illustrated that multiple mapping relations may generate for a certain subfunction with respect to Req, thus, multiple solutions are considered in this phase. In this situation, the set of solutions SOL is initiated by using the ranked p_1 with the minimum cost value (lines 2–12). The set of multiple mapping graphs MG of p_1 is got through the function $getMpGrph$, where the mapping pairs are saved (line 2). Iterate each mapping graph mg in MG and produce initiated solutions sol, where seven important properties are calculated to update the representation information of sol, e.g., $SgId$, CN, RN, SG, cvr, rdd, and rdd_cvr (lines 4–10). For each solution, the subfunction label of p_1 needs to record and add into the set $sol.SgId$, which contributes to understanding the situation of covered subfunctions in sol (line 4). According to the current mapping graph, the set of mapping requirement nodes $mg.N_{req}$ is obtained and saved in the covered node set $sol.CN$ of sol (line 5). Certainly, the node set that doesn't produce mapping nodes of p_1 (i.e., $p_1.N - mg.N_{p_1}$) is assigned to the redundancy node set $sol.RN$ of sol. Note that the symbol $mg.N_{p_1}$ represents the mapping node set generated in p_1 (line 6). In addition, the edge structure of p_1 becomes a part of solution $sol.SG$ (line 7). As regards the calculation of coverage ratio, redundancy ratio and cost value for a certain solution, we have explained their calculation method in Section 4.1. We will not introduce them (lines 8–10). Consequently, initiated solutions are generated and added to the set of solutions SOL (line 11).

The extension strategy of the backbone structure is presented based on SOL, we follow the principle of "priority to minimum cost value" to use candidate basic subfunctions to quickly cover the requirement process (lines 13–37). Each solution of SOL finally generates an optimal pattern-based composition fragment by means of our method. Particularly, the tested subfunctions are taken starting at the second of ranked BFS_{new} because the subfunction with the minimum cost value (i.e., p_1) has carried the initialization operation (line 14). In order to ensure that the structure is not lost before adding a new subfunction p_i, sol_{now} is used to retain the original structure sol. The multiple mapping graphs MG_{tmp} are obtained leveraging the function $getMpGrph$ in the same way (line 15). Iterate each the mapping relation mg of MG_{tmp} to start the coverage operation (lines 16–35). The mapping requirement nodes of p_i are respectively assigned into different sets $sol_{now}.CN$ and $sol_{now}.RN$ through the function $addNd$ (line 17). Note that if a mapping requirement node does not exist in the solution sol, it will belong to the covered node set $sol_{now}.CN$ to become a node that will be overwritten in Req; otherwise, it is added into the redundancy node set $sol_{now}.RN$ as a redundant node since its involved structure has been covered in Req. In addition, some nodes nd that do not generate mapping pairs for p_i (i.e., $(p_i.N - mg.N_{p_i})$) belong to the redundancy node set $sol_{now}.RN$ (line 18). After updating the two key sets, relevant three metrics will be calculated for sol_{now} (lines 19–21). The principle is that if the cost value of sol_{now} is smaller than the original solution sol, it is illustrated that this p_i can further cover the part of Req appropriately on the basis of sol (line 23). The current subfunction label $p_i.sgid$ and edge structure $p_i.E$ of p_i are appended to $sol_{now}.SgId$ and $sol_{now}.SG$ individually (lines 24 and 25). In this setting, sol_{now} becomes the structural basis sol for the next composition (line 26). However, in the solutions generated during the initialization phase, p_1 may be completely covered, such as the sample service pattern p_2 in Fig. 3. In that case, $sol.RN$ is 0 so that $sol.rdd$ and $sol.rdd\_cvr$ are also 0. Thus, cost values between the current solution sol_{now} and sol cannot be compared, the other indicators, e.g., the coverage ratio and the number of new coverage nodes, will be leveraged in this article. When the coverage ratio becomes larger as certain mapping requirement nodes are added into $sol_{now}.CN$, such subfunctions are considered to compose the origin solution (line 29). But, if only one coverage node is added, the redundancy obtained is relatively large for sol_{now}. In this situation, we calculate the number of new coverage nodes $(|sol_{now}.CN| - |sol.CN|)$. When it is not less than 2, sol_{now} satisfies the coverage condition to extend on sol (lines 28–34). The corresponding graph information saves into sol_{now}, and sol_{now} becomes the current solution sol for the next composition (lines 30–32).

However, if p_i cannot meet these coverage conditions, this algorithm will go into the next service subfunction in current subfunction list BFS_{new} in turn until finding an available one. Generated solutions in SOL are ranked in ascending order according to their rdd_cvr. The solution with minimum cost value is selected as the optimal solution sol_{min} at this stage, which is assumed as the backbone structure of solution with the given requirement.

Algorithm 2 Pattern coverage strategy.

```
Require:
    - $BFS_{new}$: a set of candidate subfunctions obtained by Algorithm 1
    - $Req$: a certain requirement specification
Ensure:
    - $sol_{min}$: a solution with the minimum cost value
1: Rank subfunctions in $BFS_{new}$ according to $rdd\_cvr$ obtained by Algorithm 1 in ascending order
2: $MG \leftarrow getMpGrph(p_1.map)$
3: for all $mg \in MG$ do
4:     $sol.SgId \leftarrow sol.SgId \cup p_1.sgid$
5:     $sol.CN \leftarrow mg.N_{req}$
6:     $sol.RN \leftarrow p_1.N - mg.N_{p_1}$
7:     $sol.SG \leftarrow p_1.E$
8:     $sol.cvr \leftarrow |sol.CN| / |Req.N|$
9:     $sol.rdd \leftarrow |sol.RN| / (|sol.RN| + |sol.CN|)$
10:    $sol.rdd\_cvr \leftarrow sol.rdd / sol.cvr$
11:    $SOL \leftarrow SOL \cup sol$
12: end for
13: for all $sol \in SOL$ do
14:     for all $p_i \in BFS_{new}$ ($i$ starts at 2) do
15:         $sol_{now} \leftarrow sol$; $MG_{tmp} \leftarrow getMpGrph(p_i.map)$
16:         for all $mg \in MG_{tmp}$ do
17:             $addNd(mg.N_{req}, sol_{now}.CN, sol_{now}.RN)$
18:             $sol_{now}.RN \leftarrow sol_{now}.RN \cup \{nd\}$, where $nd \in (p_i.N - mg.N_{p_i})$
19:             $sol_{now}.cvr \leftarrow |sol_{now}.CN| / |Req.N|$
20:             $sol_{now}.rdd \leftarrow |sol_{now}.RN| / (|sol_{now}.RN| + |sol_{now}.CN|)$
21:             $sol_{now}.rdd\_cvr \leftarrow sol_{now}.rdd / sol_{now}.cvr$
22:             if $sol.rdd\_cvr \neq 0$ then
23:                 if $sol_{now}.rdd\_cvr < sol.rdd\_cvr$ then
24:                     $sol_{now}.SgId \leftarrow sol_{now}.SgId \cup p_i.sgid$
25:                     $sol_{now}.SG \leftarrow sol_{now}.SG \cup p_i.E$
26:                     $sol \leftarrow sol_{now}$
27:                 end if
28:             else
29:                 if $(sol_{now}.cvr > sol.cvr)$ and $(|sol_{now}.CN| - |sol.CN|) \geq 2$ then
30:                     $sol_{now}.SgId \leftarrow sol_{now}.SgId \cup p_i.sgid$
31:                     $sol_{now}.SG \leftarrow sol_{now}.SG \cup p_i.E$
32:                     $sol \leftarrow sol_{now}$
33:                 end if
34:             end if
35:         end for
36:     end for
37: end for
38: $sol_{min} \leftarrow$ rank solutions and get the one with the minimum $rdd\_cvr$
```

4.3 Personalized pattern recommendation

After generating the backbone structure, there may still be parts of the personalized functions to implement to the remaining structure of requirement, especially, in the context-oriented experimental background.

Phase 1: *Personalized pattern container selection.* Since the title, description, and services are contained in Req, we can learn and obtain the corresponding multi-vector representation (i.e., workflow-topic vector, workflow-feature vector, and workflow-service vector). Similar experiments are found according to the multiple information of Req, in order to reuse their personalized functional patterns. Particularly, leveraging representative workflow of each personality pattern container, their similarities can be determined with respect to Req through vector similarity calculation in formula (5). The personalized pattern container with the most similarity is selected as the target personalized pattern container.

Phase 2: *Personalized pattern discovery and coverage.* According to the target personalized pattern container, included personalized patterns are checked whether whose services exist in the uncovered parts of Req. Remove unsatisfied personalized patterns, and calculate metrics (i.e., cover ratio, redundancy ration, and cost value) of each saved pattern through Algorithm 1. Personalized patterns are ranked through their cost values in ascending order and retained to the PPS. Similarly, the pattern coverage strategy is carried out based on PPS through Algorithm 2, where lines 13–37 are executed analogously to assign personalized patterns in PPS to the backbone structure sol_{min} in the previous stage. Consequently, the current structure is further covered by satisfied personalized patterns, to generate a more suitable solution with the smaller cost value to accomplish the composition work based on personalized patterns.

4.4 Atomic service recommendation

Appropriate patterns compose together to achieve the solution with the minimum cost value with respect to Req. Although service patterns can cover more service activities in Req with a higher coverage ratio and lower redundancy ratio, the remaining uncovered part of Req lacks of atomic functions to help optimize the experimental design. Candidate services are selected from service repository according to the uncovered services in Req. When a certain candidate service is added, make sure that it owns the invocation relationship with related connected services on sol. In this situation, this service is allowed to insert to the current solution sol; otherwise, it is discarded. Final metrics of sol are calculated and updated. Obviously, this step makes the cost value of the solution smaller. The optimal solution is generated with respect to a certain requirement.

Supposing that the number of candidate subgraphs is p (basic subfunctions and personalized patterns) and the average size of these service subgraphs is s, there are n services in a certain requirement and g mapping graphs are produced for each service subgraph on average with respect to the requirement. In this situation, the time complexity of Algorithm 2 reflects in line 1, lines 3–12, and lines 13–37. In detail, line 1 iterates candidate patterns to calculate the coverage ratio, redundancy ratio, and cost value through Algorithm 1 and ranks patterns by the ranking algorithm. For the calculation part of Algorithm 1, $\mathcal{O}(psn)$ is obtained while $\mathcal{O}(p \log p)$ is time complexity for ranking patterns. Thus, the time complexity of line 1 is $max(\mathcal{O}(psn), \mathcal{O}(p \log p)) = \mathcal{O}(psn)$. For the initiation phase of solutions in lines 3–12, depending on the number of mapping graphs for p_1, $\mathcal{O}(g)$ is its time complexity. In addition, for lines 13–37, the pattern coverage strategy is presented and the time complexity is $\mathcal{O}(gpsg)$. Obviously, the time complexity of our entire Algorithm 2 is $max(\mathcal{O}(psn), \mathcal{O}(g^2ps))$. Generally, the number of produced mapping graphs g is relatively small while the service number of requirements n is comparatively large in our experimental setting. Thus, $\mathcal{O}(psn)$ is the final time complexity of Algorithm 2. Note that the pattern-based approach reduces the complexity of service composition compared with atomic services composition $\mathcal{O}(nt)$ (assumed t is the number of all services in the repository; generally, t is relatively large). Further, different types of pattern mining will speed up the scope of discovery and facilitate the reuse of workflow fragments.

5 Experimental setting

The prototype has been implemented in Python and Java projects. Scientific workflows in the category of *Taverna 2* of the *myExperiment* repository are cleansed for the experimental verification and evaluation, where 1573 scientific workflows are contained in this category. Experiments are conducted on a desktop with an Intel(R) Core(TM) i5-6500 processor, a 12.00GB memory, and a 64-bit Window 10 system and a virtual Linux system.

5.1 Data collection and cleansing

Before conducting our experiments, these workflows are cleansed as follows. Firstly, we remove scientific workflows where the title or description lacks corresponding declarative information. In this case, the multi-dimensional representation of workflows cannot be generated to promote the personalized pattern discovery. Secondly, workflows with incomplete structure are removed, since the mining works of basic and personalized patterns is closely related to the graph structure. Thirdly, services with the slim type should be filtered out. These services are mostly the *glue* codes not to represent invocation relations between functionalities. Consequently, semantic similarity between services is required to evaluate similarity for them. Importantly, improper *noise* words are not recognized by *WordNet*, which are also meaningless word entities in similarity calculation. Thus, we need to supplement and transform them to recognizable formalization. The specific process is explained as follows:

- These are 334 abbreviations used in the bioinfomatics domain, but they cannot be recognized by *WordNet*. An example is "snp," which means "single nucleotide polymorphism" after searching on the Web. A conversion table is manually established, which aims to transform these abbreviations into their full description. If a full description cannot be found through browsing the web like "martijn," it is assumed to be dissimilar to any other words. These abbreviations should be removed.
- Regarding the abbreviation which is just a part of a word, we can convert it to the full description. An example is "bio" which corresponds to "biology."
- Words, whose meaning is not clear and information is incomplete, are complemented according to context and Web network. An example is "cpath," which is rewritten as "Canadian Professional Association for Transgender Health" in the life science domain.
- When two or several words are joint without delimiters, they are separated manually. An example is "documentfind," which is transferred into two words "document" and "find."
- When some words are incorrect in spelling, they are corrected according to the word dictionary. An example is "searcg," which is corrected as "search."

In addition, the multi-dimensional representation of workflows is leveraged to calculate similarities between workflows by formula (5) in the personalized pattern discovery. However, topic model and feature extraction techniques are sensitive for word entities. Thus, we also need to handle the title and description of workflows to improve the efficiency of the *BTM* and *TF-IDF* algorithm:

- Words like *accessed*, *accesses*, and *accessing* have a common word root *access*. Affixes are removed to keep the root only. The lemmatization technology is applied by establishing common reduction rules.
- Stopwords such as *can*, *of*, and *and* are removed in order to improve the accuracy of topic and feature selection. A stopword table is established according to the dataset, which contributes to the dataset cleansing.

After data cleansing, workflows with relatively accurate title, description, service structure, and service information are retained to conduct our technique. In the next section, we intend to present experimental setup to conduct our experiments.

5.2 Experimental setup

In order to explore the experimental effect of sample requirements with various scales, we select scientific workflows in the repository as experimental sample requirements. Thus, these scientific workflows are counted in terms of the number of contained services, where the corresponding proportions are generated and shown in Fig. 4. It is seen that the service number of most workflows focuses on 5∼9 services at a scale. The proportion of workflows decreases as the number of services increases from the overall trend. In this situation, requirement scales to be selected are divided into 7 intervals, which are [4–10, 13], and [15, 25]. For the sake of notation, these intervals are denoted as *scale1*, *scale2*, *scale3*, *scale4*, *scale5*, *scale6*, and *scale7*. We respectively extract 5 sample requirements from each interval to explore the accuracy of our technique under different requirement scales. The detailed information for sample requirements are shown as Table 6, where contained workflows are illustrated. In addition, the special

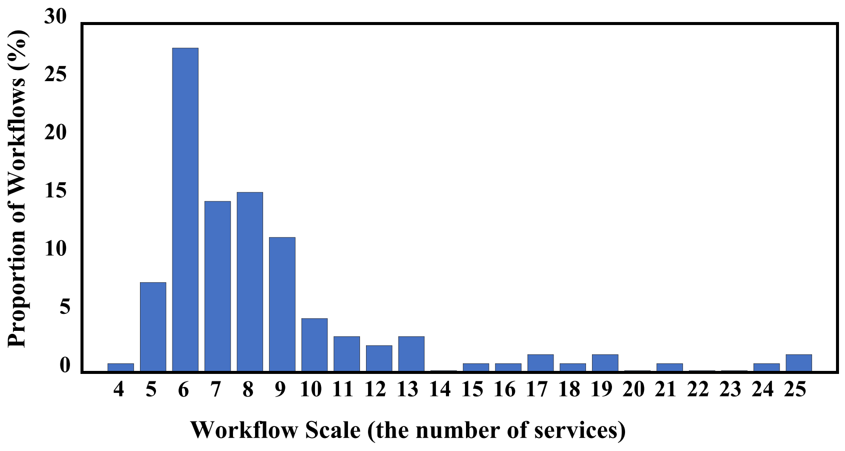


Fig. 4 The proportion of workflows under the various numbers of contained services

Table 6 Selected sample requirements according to various scales

Scale \ swf	1	2	3	4	5
$scale1$	swf_{12}	swf_{24}	swf_{58}	swf_{77}	swf_{111}
$scale2$	swf_{20}	swf_{44}	swf_{68}	swf_{72}	swf_{100}
$scale3$	swf_{3}	swf_{19}	swf_{35}	swf_{50}	swf_{122}
$scale4$	swf_{8}	swf_{18}	swf_{31}	swf_{69}	swf_{92}
$scale5$	swf_{0}	swf_{5}	swf_{10}	swf_{30}	swf_{60}
$scale6$	swf_{1}	swf_{15}	swf_{36}	swf_{48}	swf_{112}
$scale7$	swf_{2}	swf_{23}	swf_{41}	swf_{34}	swf_{90}

situation is considered, which some changes are applied to sample requirements to achieve the purpose of testing novel personalized requirements.

5.3 Pattern-based workflow fragment discovery when no changes are applied to sample requirements

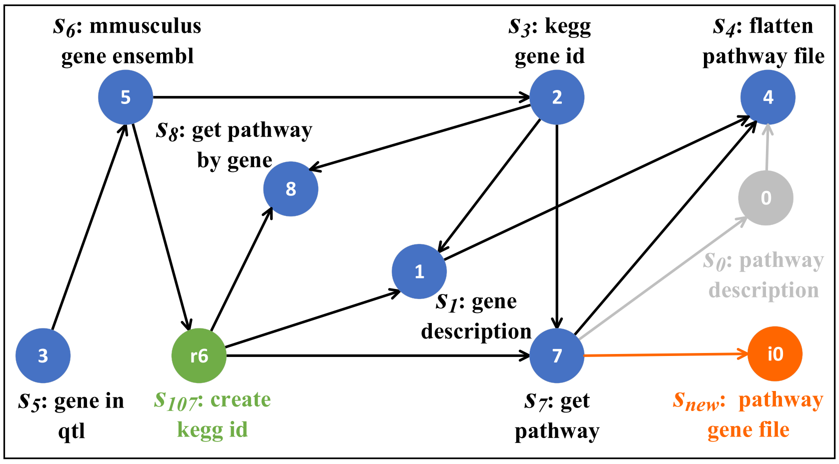
Experiments are conducted when sample requirements remained as they are retrieved from dataset. A requirement is to discover basic subfunctions, personalized patterns, and atomic services that can be reused through our technique, which arrives the minimal cost value on the final solution. Experiments for 35 sample requirements return their respective final solutions. For instance, the sample requirement req_{swf_0} developed from swf_0 in Table 6 that has 9 service nodes. Our technique can discover 2 basic subfunctions and 1 personalized pattern to reuse; these subgraphs cover the most of service structure (i.e., 7 services) in req_{swf_0}. In addition, 2 reusable atomic services are recommended to optimize the experimental design. In this situation, in addition to generating some redundant services, all functions that should be implemented can be found in reusable parts. To sum up, our technique maximizes the use of basic and personalized patterns to accomplish the given requirement, and the optimal solution can be recommended by our strategy.

5.4 Pattern-based workflow fragment discovery when changes are applied to sample requirements

Novel requirements may not be equivalent to sample workflow requirements in the repository. In this case, certain sample requirements are changed to generate additional testing requirements. Changing operation include *insertion*, *deletion*, and *replacement* as follows:

- *Insertion*. Services not included in a sample requirement req_{smpl} are inserted as part of a testing requirement req_{tst}. Generally, if some services belong to another, workflows or new services are constructed, and they express similar meanings as req_{smpl}; req_{tst} is usually more appropriate to share the text information with req_{smpl}. For instance, service $node_{i0}$ named *pathway gene file* is inserted in Fig. 5.
- *Deletion*. Services are deleted from a sample requirement. An example is the service $node_0$ named *pathway description*, which is deleted from a requirement as shown in Fig. 5.
- *Replacement*. Services in a sample requirement req_{smpl} are replaced by the others which are not specified in req_{smpl}. Similar to the case of *insertion*, if services have similar meanings to the replaced services, the testing requirement is similar with req_{smpl}. An example is the service $node_6$ = "*kegg gene id*" which is to be replaced by another similar service $node_{r6}$ = "*create kegg id*" as shown in Fig. 5.

Experiments are conducted for changes applied upon sample requirements, which are changed through *insertion*, *deletion*, and *replacement* operations. In addition, some novel services may be constructed in sample requirements, which is more in line with new features of real applications



s_6: mmusculus gene ensembl
s_3: kegg gene id
s_4: flatten pathway file
s_8: get pathway by gene
5
2
4
8
0
1
s_0: pathway description
s_1: gene description
3
r6
7
i0
s_5: gene in qtl
s_{107}: create kegg id
s_7: get pathway
s_{new}: pathway gene file

Fig. 5 A testing requirement is developed from Fig. 2. Changes are made through inserting 1 service, deleting 1 service, and replacing 1 service

such that certain requirements put forward by scientists may be different but similar with previous experiments. Consequently, 10 modified requirements are generated to conduct experiments of specified requirements. An example is shown in Fig. 5, where 1 service is inserted, 1 service is deleted, and 1 service is replaced. Specifically, the service $node_{i0}$ named *pathway gene file* is inserted into $node_7$ to build a new connection ($node_7$ -> $node_{i0}$). The service $node_0$ is deleted, and service $node_6$ is replaced by $node_{r6}$ that has similar meaning as $node_6$.

Regarding these 10 testing workflow requirements, results for 7 testing requirements return the solutions that contained services in the requirement that can be covered by patterns and atomic services discovered from our technique. Three experiments, whose solutions can not cover all functional services, are set to add newly constructed services, e.g., $node_{i0}$ in Fig. 5. Their generated solutions can only be composed of existing structures (patterns) and services in the repository to obtain relatively optimal designs. According to recommendation results, it is shown that our technology is appropriate and reasonable when certain changes are applied to sample requirements.

6 Experimental evaluation

The assessment approach is taken when experiments are conducted, and performance metrics, evaluation strategy, and the corresponding result are presented as follows.

6.1 Performance metrics

The important performance measure is considered in evaluating the effectiveness of the proposed method: F1-measure is calculated by leveraging the precision and recall in the recommendation process. In particular, the recommended solution is generated for every personalized requirement, which contains several basic subfunctions, personalized patterns, and atomic services to accomplish the given experiment. Since recommended solutions generally cover most structures of requirements through basic pattern recommendation, personalized pattern recommendation, and service recommendation, the recall is the relatively high value, but the precision plays a sensitive role depending on redundancy services. In this situation, it is a key issue to evaluate the whole effect of recommendation. Therefore, the F1-measure is finally confirmed as our evaluated performance metric, which combines precision and recall with an equal weight in formula (9):

$$\mathrm{F1} = \frac{(\text{precision} \times \text{recall})}{(\text{precision} + \text{recall})/2} \tag{9}$$

where (i) the precision is the ratio between the number of correctly covered services $|CS_{sol}|$ that the recommended solution sol can cover the process fragment of the given requirement Req and the total number of services $|S_{sol}|$ in sol and (ii) the recall is the ratio between $|CS_{sol}|$ and the total number of services $|S_{Req}|$ in Req. Particularly, precision and recall are computed as follows:

$$\text{precision} = |CS_{sol}| \div |S_{sol}| \tag{10}$$

$$\text{recall} = |CS_{sol}| \div |S_{Req}| \tag{11}$$

6.2 Evaluation strategy and result

In this section, we present our evaluation strategy and results of our technique, and they are illustrated as follows:

- The impact and determination of key parameters
- The effect of our technique compared with two baseline covering approaches

The results of our experiments in (i) Section 5.3, where sample requirements remained as they are, are retrieved from the dataset, and Section 5.4, where changes are applied to sample requirements, are adopted for comparisons.

6.2.1 Key parameter determination

In this article, there are two crucial and undetermined key parameters. They are considered influential factors to be explored and explained as follows:

- K_{bp}: It influences the number of subfunction clusters selected as basic patterns in Section 2.3.
- $thrd_{\mathrm{frq}}$: It is the frequency threshold factor of mining personalized patterns from respective workflow clusters in Section 3.2.

A solution is generated leveraging basic pattern recommendation, personalized pattern recommendation, and atomic service recommendation. Importantly, the reuse ratio about subgraphs is different at each stage, which can reflect the covering effect of our technique as the key parameter changes. Thus, in the basic subfunction and personalized pattern compositions, their respective reuse ratios are considered to determine the optimal parameters. Note that the reuse ratio of basic subfunctions (or personalized patterns) is the percentage of the number of candidate reused basic subfunctions (or personalized patterns) to the total number of basic subfunctions (or personalized patterns) produced in the corresponding phase. We call it as BP-based reuse ratio (or *PP*-based reuse ratio).

To investigate the impact of K_{bp} to BP-based reuse ratio, *PP*-based reuse ratio, and F1 value, we set $thrd_{\mathrm{frq}}$ to 0.2,

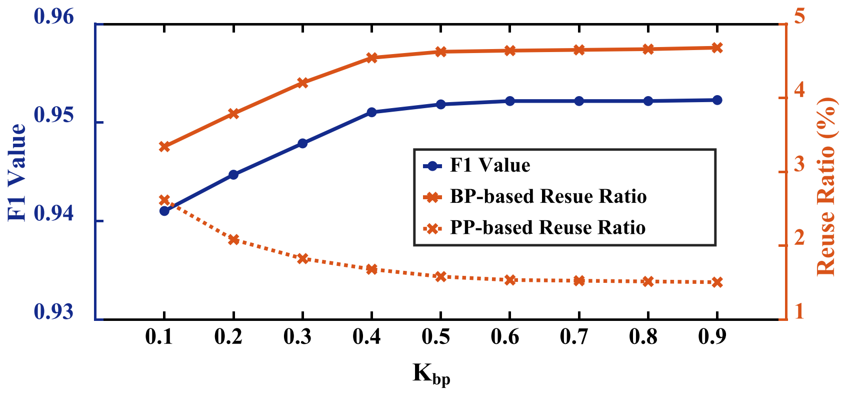


Fig. 6 BP-based reuse ratio, *PP*-based reuse ratio, and F1 value for our technique, when $thrd_{frq}$ is set to 0.2, and K_{bp} is set from 0.1 to 0.9 with an increment of 0.1

and K_{bp} to a value from 0.1 to 0.9 with an increment of 0.1. Figure 6 shows that the BP-based reuse ratio and F1 value increase and *PP*-based reuse ratio drops along with the increasing of K_{bp}. Specifically, more and more basic subfunctions are reused when K_{bp} increases, since more basic patterns are selected to consider generalized function reuse. However, when K_{bp} is set to be relatively large (from 0.4 to 0.9), Fig. 6 shows that BP-based reuse ratio is relatively stable, since the majority of reusable basic subfunctions have been recommended to composite solutions. It is shown that BP-based reuse ratio remains at the 4.6% level. In addition, *PP*-based reuse ratio decreases to an extent as the use of basic subfunctions increases where K_{bp} is set from 0.1 to 0.4, since the backbone structure of the solution has been covered by the more basic subfunctions. When K_{bp} is adjusted from 0.4 to 0.9, *PP*-based reuse ratio tends to a steady state and its values remains 1.5%, because the backbone structure has been stably determined by the basic pattern recommendation. After analyzing of BP-based and *PP*-based reuse ratios, the whole recommendation effect (i.e., F1-measure) is also obtained in Fig. 6. It is shown that F1 value is gradually rising as K_{bp} increases. Besides, when K_{bp} is set from 0.1 to 0.4, results show that F1 value begins to increase to a small extent from 0.94 to 0.95, since more and more correct subfunctions are recommended to solutions leveraging our technique. Meanwhile, when K_{bp} is set from 0.4 to 0.9, F1-measure keeps at a stable value (0.95), because most of expected structure and services have been discovered and included in solutions. According to the above explanation, an optimal K_{bp} is got and determined as 0.4 in our experiments, where BP-based reuse ratio, *PP*-based reuse ratio, and F1-measure are high values.

To explore the impact of $thrd_{\mathrm{frq}}$ to BP-based reuse ratio, *PP*-based reuse ratio, and F1 value, we set K_{bp} to 0.4 and $thrd_{\mathrm{frq}}$ to a value from 0.1 to 0.9 with an increment of 0.1. As shown in Fig. 7, BP-based reuse ratio remains unchanged, *PP*-based reuse ratio decreases, and F1 value increases. Specifically, when K_{bp} is set to a certain fixed value, no matter how $thrd_{\mathrm{frq}}$ changes, candidate reusable basic subfunctions does not change at the basic pattern recommendation, which makes BP-based reuse ratio invariable at 4.6% value. Importantly, $thrd_{\mathrm{frq}}$ only influences the process of personalized pattern recommendation. In this setting, the backbone structure of the solution has been determined; the remaining structure needs to be covered by personalized patterns. Figure 7 shows that *PP*-based reuse ratio decreases to a large extent (from 6.4 to 1.5%) when $thrd_{\mathrm{frq}}$ is set from 0.1 to 0.4, since the demand of frequency threshold is more strict

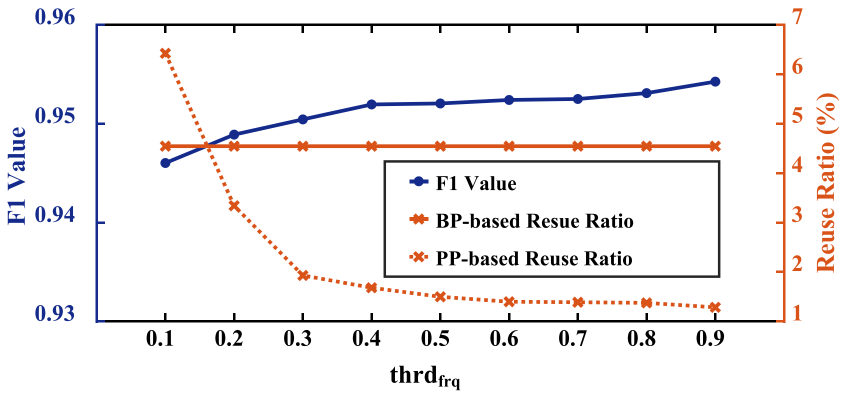


Fig. 7 BP-based reuse ratio, *PP*-based reuse ratio, and F1 value for our technique, when K_{bp} is set to 0.4 and $thrd_{\mathrm{frq}}$ is set from 0.1 to 0.9 with an increment of 0.1

for mining personalized patterns from workflow clusters, and the number of candidate reusable personalized patterns becomes smaller. Thus, *PP*-based reuse ratio shows the decreased trend. When $thrd_{\text{frq}}$ is adjusted from 0.4 to 0.9, *PP*-based reuse ratio tends to be stable, as most of expected personalized patterns have been minded to composite the solution. In addition, Fig. 7 also shows that F1 value is gradually rising when $thrd_{\text{frq}}$ increases. Through our analysis, although BP-based reuse ratio remains unchanged and *PP*-based reuse ratio decreases, our technique can still obtain appropriate atomic services to accomplish requirements. Since the number of redundancy services decreases, F1 value will increase. Consequently, an optimal $thrd_{\text{frq}}$ is determined as 0.4, where *PP*-based reuse ratio and F1-measure are relatively high values.

6.2.2 Baseline techniques comparison

The following two covering strategies are chosen as the baseline for comparison with our technique:

- Random strategy: It is to choose the basic subfunctions or personalized patterns with the highest coverage ratio to conduct composition work according to the current uncovered structure of the requirement.
- Continuous strategy: It is to choose the basic sub-functions or personalized patterns, which will cover the services neighboring the covered services, to conduct composition work according to the neighboring structure of the requirement.

This section presents and discusses the evaluation results of our technique, random covering strategy, and continuous covering strategy in terms of different requirement scales. Note that requirement scales have been illustrated in Section 5.2. As discussed in Section 6.2.1, the optimal K_{bp} and $thrd_{\text{frq}}$ have been determined according to the above experimental comparisons. In this setting, to explore the respective effects of our technique, random strategy, and continuous strategy to F1-measure under different scales, we set K_{bp} to 0.4 and $thrd_{\text{frq}}$ to 0.4. As shown in Fig. 8, our technique performs better than random strategy and continuous strategy in F1-measure. As the scale of requirements increases, our technique can maintain a high F1 value range (between 0.94 and 0.98). However, random strategy and continuous strategy are not stable in the covering composition. In particular, when the requirement scale is $scale3$, F1 value of random strategy is only 0.75. Since random strategy chooses the optimal subgraph to cover at the early composition stage, it is easy to leave a large number of "discrete fragment" (including less services and more redundancy services), and at the late composition stage, that will reduce the using efficiency of service patterns. Figure 8 also shows that F1 value of continuous strategy is 0.71 when the requirement scale is $scale5$, which is the lowest F1 value under all experiments distributed in different requirement scales. Through our analysis, it is found that the continuous strategy will not cause discrete fragments like random strategy; however, it may not find the optimal subgraph to compose the solution. Thus, continuous strategy shows the lower recommended effect compared with our technique and random strategy. Importantly, our technique uses three meaningful metrics (i.e., coverage ratio, redundancy ratio, and cost value) to select the optimal subfunctions and personalized patterns, and the cost value metric of the current solution is also considered to evaluate quality of solutions. According to the above explanation, it is concluded that our technology can arrive optimal recommendations compared with other two covering strategies for supporting real-word applications.

7 Related work and comparison

As the number of available Web services and scientific workflows is rapidly increasing, the discovery of relevant services and workflow fragments from massive candidates

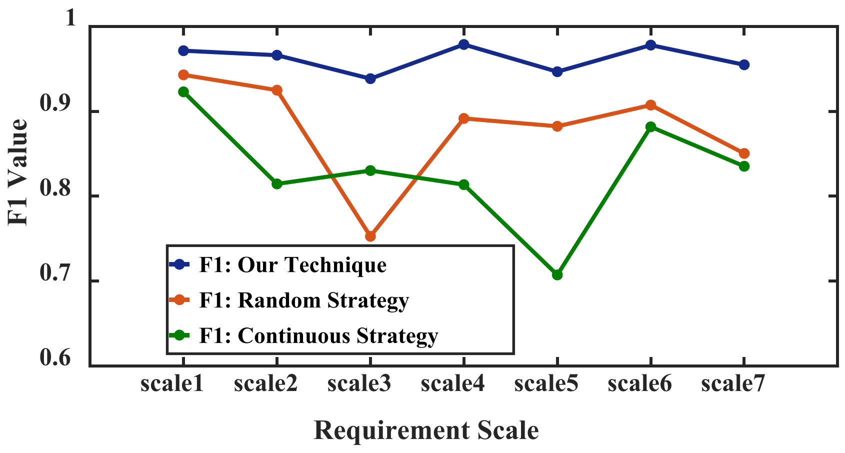


Fig. 8 F1 values for our technique, random strategy, and continuous strategy in different requirement scales, when K_{bp} is set to 0.4 and $thrd_{frq}$ is set to 0.4

is quite crucial and demands a lot of efforts [20]. For example, Wenmin et al. propose the history-based method; the solution is determined according to actual recorded Qos of candidate services in the history [2]. However, the efficiency of this method is low because the historical Qos values stand alone, and it causes the loss of implicit relevance information, which is the connection among different functionalities or services in historical experience. In addition, the QoS may be not accurate enough due to a dynamic environment which is always full of uncertainty; all these will impact the optimality of service solutions. Therefore, a large number of approaches and composition techniques are proposed to enhance efficiency, bringing a hot topic in the research community. Among these tools and techniques, frequent pattern-based composition approaches receive a great deal of attention [21].

Given the dataset of previously proposed scientific workflows, it can always find out some frequently occurred fragment patterns from history and then based on these patterns make service composition in the future. A great number of service composition approaches based on this idea are proposed and proved to be very successful. These frequent pattern usually give rules that summarize the relationships of correlations, collaborations, and complements between services in historical compositions. Original research on "patterns" emphasized on the "abstraction" characteristic with the objective of "reuse" [9]. Developers write patterns in terms of high-level functionalities and decide on the function signature they want to associate to a functionality appearing in a pattern [10]. Hu et al. define the reusable logic in the business processes as the patterns, which can be shared among users to facilitate the construction of new service solution [11]. In [12], frequent process fragments appeared in the service composition (including a set of frequently co-occurring services and the control flows among these services) are extracted to prepare for the follow-up service composition be requisitioned by using data mining technique. In developing service-oriented applications, similarly, a service pattern is defined as a set of services that are repetitively used together in various applications and records the best practices [13]. Zhang et al. extract the repeated process fragments in the service flow to expect service reuse in new solution construction [22]. It has been proven that the utilization of patterns can drastically improve the efficiency of service composition and service mashup [23]. Huang et al. further utilize fragment service process for composite service selection including control flow and bound services [14]. However, experimental requirements are usually shown as the function-oriented personalized way; certain functions have to bundle with specific demands. For instance, the generalized component to reach the basic transformation and the individuation component to obtain certain special functionalities are required. Thus, service composition studies upon patterns should not focus on frequent composite solutions only. Unfortunately, in the above state-of-the-art approaches, this key aspect is somehow ignored, and relevant researchers seldom consider and focus on the mining task of different type patterns. It is crucial to figure out how to reuse these diverse pattern fragments at the right time for accomplishing personalized requirements. Importantly, in our article, we work on discoveries of basic patterns and personalized patterns and leverage them to composite the optimal solution through a novel composition strategy with respect to requirements.

It is noteworthy that service composition remains a functional topic in the field of service-oriented computing [24]. A typical service composition process carries two fundamental assumptions. First is that a planning phase has resulted in a structured workflow plan [25], i.e., a reference model. Second is that identified service components will be linked to each other directly [26], meaning that the output of a service component has to be plugged into another service component as input. In contrast, our work raises the bar and aims to fill the gap by composing service patterns mined by their past collaboration history [27, 28]. Service relationships are remained to promote service composition. These service connections in our work come from historical successful service connections happened in past workflows. For scientific researchers, such service connections are more trustworthy.

In addition, in this article, we leverage the significant technique of frequency pattern mining. In the pattern scenario or graph mining, the frequent subgraph discovery is a challenge task, which deals with identifying frequently occurring subgraphs in a given graph dataset. A subgraph can be considered frequent when the number of its occurrences within the dataset exceeds a specified threshold. There are two variants of this problem, in the first variant [29, 30], the input graph data contain a set of small to medium-size graphs, which are called transactions. In the second variant [31, 32], a single input graph data is used (i.e., one very large graph, graph with hundreds of thousands nodes). Obviously, our frequent subfunction mining belongs to the first variant in this article. In the transaction scenario, miner/frequent subgraph mining algorithm recursively generates all possible extensions from empty graph by adding edges and nodes to already-generated subgraphs. Then, isomorphism testing will be performed in each new possible extensions to determine whether it appears frequently. Early miner/frequent subgraph mining algorithms generated extensions in a breadth-first search (*BFS*) way, e.g., *AGM* [33] and *FSG* [34] based on Apriori properties. In view of the shortcoming that a large number of candidate subgraphs will be produced. Thus, the depth-first search (*DFS*) approaches are presented, which need less memory and almost show better performance. Authors in [35] have summarized three main subproblems (i.e., purposive refinement, efficient enumeration, and focused isomorphism testing) and made a quantitative and detailed comparison of some typical DFS algorithms, e.g., *Mofa*, *FFSM* [36], *gSpan* [15], and *Gaston* and some special extensions of them, e.g., *CloseGraph*. In this situation, the DFS algorithms will be applied into our work. Due to the directivity of workflow structure, *gSpan* is considered, which is the state-of-the-art subgraph mining method for handling the directed graphs.

8 Conclusion

This article proposes a novel workflow fragment discovery mechanism for personalized requirements considering basic patterns and personalized patterns. Specifically, the discovery strategy of basic patterns is presented, where frequent subfunctions are mined from scientific experiments leveraging the frequent subgraph mining approach. Semantic relevance of these subfunctions is quantified leveraging their topics obtained from the biterm topic model. Subfunctions with same or similar functionality are clustered together, and clusters with high functional frequency are assumed to be *basic patterns*. In addition, the discovery strategy of personalized patterns is presented, where the multi-dimensional representation for scientific workflows is constructed to express the workflow relevance from multiple aspects: title, description, and contained services. Workflow similarities are calculated through representation vectors. Workflow clustering is conducted, where frequent personalized subgraphs discovered from a certain cluster with similar workflows are viewed as *personalized patterns* under the current content. Given a personalized requirement specified in terms of its title, description, and workflow template, target basic patterns and candidate basic subfunctions are discovered to compose the backbone structure of solution according to related metrics and a novel coverage strategy. Based on the backbone structure and requirement template, candidate personalized patterns are conducted to cover the remaining structure of requirement. Consequently, the current solution is optimized through complementing atomic services, to generate the final solution with respect to the given requirement. Evaluation results show that our technique is accurate on discovering fragment solutions for personalized requirements in comparison with the state of the art.

References

1. Zheng Z, Ma H, Lyu MR, King I (2010) Qos-aware web service recommendation by collaborative filtering. IEEE Trans Serv Comput 4(2):140–152
2. Lin W, Dou W, Luo X, Chen J (2011) A history record-based service optimization method for qos-aware service composition. In: IEEE international conference on web services, pp 666–673
3. Lemos AL, Daniel F, Benatallah B (2016) Web service composition: a survey of techniques and tools. ACM Comput Surv (CSUR) 48(3):33
4. Zhou Z, Cheng Z, Zhang L-J, Gaaloul W, Ning K (2018) Scientific workflow clustering and recommendation leveraging layer hierarchical analysis, vol 11
5. Wen J, Zhou Z, Shi Z, Wang J, Duan Y, Zhang Y (2018) Crossing scientific workflow fragments discovery through activity abstraction in smart campus. IEEE Access 6:40530–40546
6. Goble CA, Bhagat J, Aleksejevs S, Cruickshank D, Michaelides D, Newman D, Borkum M, Bechhofer S, Roos M, Li P (2010) Myexperiment: a repository and social network for the sharing of bioinformatics workflows. Nucleic Acids Res 38(suppl_2):W677–W682
7. Garijo D, Alper P, Belhajjame K, Corcho O, Gil Y, Goble C (2014) Common motifs in scientific workflows: an empirical analysis. Futur Gener Comput Syst 36(3):338–351
8. Eshuis R, Lecue F, Mehandjiev N (2016) Flexible construction of executable service compositions from reusable semantic knowledge. ACM Trans Web 10(1):5
9. Tut MT, Edmond D (2002) The use of patterns in service composition. In: International workshop on web services, e-business, and the semantic web. Springer, pp 28–40
10. Melloul L, Fox A (2004) Reusable functional composition patterns for web services. In: IEEE international conference on web services, pp 498–505
11. Wang X, Niu W, Li G, Yang X, Shi Z (2012) Mining frequent agent action patterns for effective multi-agent-based web service composition. In: Agents and data mining iteration, pp 211–227
12. Zhang M, Zhang B, Na J, Zhang X (2009) Composite service selection based on dot pattern mining. In: SERVICES-I, pp 740–746
13. Upadhyaya B, Tang R, Zou Y (2013) An approach for mining web service composition patterns from execution logs. Journal of Software: Evolution and Process 25(8):841–870
14. Huang Z, Huai J, Liu X, Zhu J (2010) Business process decomposition based on service relevance mining. In: International Conference on Web Intelligence and Intelligent Agent Technology, pp 573–580
15. Yan X, Han J (2002) gspan: Graph-based substructure pattern mining. In: IEEE international conference on data mining, pp 721–727
16. Cheng X, Yan X, Lan Y, Guo J (2014) Btm: Topic modeling over short texts. IEEE Trans Knowl Data Eng 26(12):2928–2941
17. Blei D, Ng A, Jordan M (2003) Latent dirichlet allocation. Machine Learning Reasearch, pp 933–1022
18. Blondel VD, Guillaume J-L, Lambiotte R, Lefebvre E (2008) Fast unfolding of communities in large networks. J Stat Mech: Theory Exp 2008(10):155–168
19. Kim D, Seo D, Cho S, Kang P (2019) Multi-co-training for document classification using various document representations: Tf–idf, lda, and doc2vec. Inf Sci 477:15–29
20. Liu J, Jiang J, Cui X, Yang W, Liu X (2015) Power consumption prediction of web services for energy-efficient service selection. Pers Ubiquit Comput (PUC) 17(7):1063–1073
21. Chen J, Zhou X, Jin Q (2013) Recommendation of optimized information seeking process based on the similarity of user access behavior patterns. Pers Ubiquit Comput (PUC) 17(8):1671–1681
22. Hu H, Han Y, Huang K, Li G, Zhao Z (2004) A pattern-based approach to facilitating service composition. In: Grid and cooperative computing workshops, pp 90–98
23. Daniel F, Rodriguez C, Chowdhury SR, Nezhad HRM, Casati F (2012) Discovery and reuse of composition knowledge for assisted

mashup development. In: International conference companion on world wide web, pp 493–494
24. Rodriguez-Mier P, Mucientes M, Lama M (2015) Hybrid optimization algorithm for large-scale qos-aware service composition. IEEE Trans Serv Comput 10(4):547–559
25. Deng S, Wang D, Li Y, Cao B, Yin J, Wu Z, Zhou M (2016) A recommendation system to facilitate business process modeling. IEEE Transactions on Cybernetics 47(6):1380–1394
26. Bevilacqua L, Furno A, Di Carlo VS (2011) E. zimeo, "compositions from owl-s described services. In: IEEE international conference on software, knowledge information, industrial management and applications (SKIMA) Proceedings. IEEE, pp 1–8
27. Bie R, Mehmood R, Ruan S, Sun Y, Dawood H (2016) Adaptive fuzzy clustering by fast search and find of density peaks. Pers Ubiquit Comput (PUC) 20(5):785–793
28. Verkasalo H (2009) Contextual patterns in mobile service usage. Pers Ubiquit Comput (PUC), 13(5):331–342
29. Thomas LT, Valluri SR, Karlapalem K (2010) Margin: Maximal frequent subgraph mining. ACM Trans Knowl Discov Data (TKDD) 4(3):10
30. Chaoji V, Hasan MA, Salem S, Zaki MJ (2008) An integrated, generic approach to pattern mining: data mining template library. Data Min Knowl Disc 17(3):457–495
31. Elseidy M, Abdelhamid E, Skiadopoulos S, Kalnis P (2014) Grami: frequent subgraph and pattern mining in a single large graph. Proceedings of the Vldb Endowment 7(7):517–528
32. Teixeira CH, Fonseca AJ, Serafini M, Siganos G, Zaki MJ, Aboulnaga A (2015) Arabesque: a system for distributed graph mining. In: The 25th symposium on operating system principles. ACM, pp 425–440
33. Inokuchi A, Washio T, Motoda H (2000) An apriori-based algorithm for mining frequent substructures from graph data. In: European conference on principles of data mining & knowledge discovery, pp. 13–23
34. Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: IEEE International Conference on Data Mining, pp 313–320
35. Worlein M, Meinl T, Fischer I, Philippsen M (2005) A quantitative comparison of the subgraph miners mofa, gspan, ffsm, and gaston. In: European conference on principles & practice of knowledge discovery in databases, pp 392–403
36. Huan J, Wang W, Prins J (2003) Efficient mining of frequent subgraphs in the. In IEEE international conference on data mining. IEEE, pp 549–552

Affiliations

Jinfeng Wen[1] · Zhangbing Zhou[1,2] · Fei Lei[3] · Junsheng Zhang[4]

✉ Fei Lei
leifei@bjut.edu.cn

Jinfeng Wen
wenjinfeng.cugb@gmail.com

Junsheng Zhang
zhangjs@istic.ac.cn

1 School of Information Engineering, China University of Geosciences (Beijing), Beijing, 100083, China

2 Computer Science Department, TELECOM SudParis, 91011, Evry, France

3 Faculty of Information Technology, Beijing University of Technology, Beijing, 100124, China

4 Institute of Scientific and Technical Information of China, Beijing, 100038, China