"Failure" Service Pattern Mining for Exploratory Service Composition

Yunjing Yuan, Jing Wang(✉), Yanbo Han, Qianwen Li, Gaojian Chen, and Boyang Jiao

Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, School of Information Science and Technology, No.5 Jinyuanzhuang Road, Shijingshan District, Beijing, China
wang_jing@ncut.edu.cn, yhan@ict.ac.cn

Abstract. To adapt to uncertain and dynamic requirements, exploratory service composition enables business users to construct service composition processes in a trial-and-error manner. A large number of service composition processes are generated, which can be learned to improve the reusability of the service composition processes. By mining these service composition processes and abstracting the mining results to service patterns, the efficiency of service composition can be effectively improved. At present, there has been some research on the methods of service pattern mining. Most of the work focuses on successful service composition processes, but the ones that fail are also valuable. For example, by using the mining results of failure service composition processes, the accuracy of service recommendations can be improved. To solve this problem, this paper proposes a "failure" service pattern mining algorithm (FSPMA) for exploratory service composition, which extends the gSpan algorithm, and can mine "failure" service patterns from service composition processes for further reuse. Meanwhile, the exploratory service composition model and the service pattern model are explained for the FSPMA. The prototype implementation of the exploratory service composition environment is introduced, which integrates the FSPMA. The experimental evaluation is explained to verify the algorithm, and the result shows that the efficiency of the FSPMA has a significant improvement in mining "failure" service patterns compared with the gSpan algorithm and the TKG algorithm. Finally, the application of "failure" service patterns in service recommendations is given.

Keywords: Exploratory service composition · Service pattern mining · gSpan

1 Introduction

In the big data era, people's lives rely more and more on the services delivered via the Internet, and servitization becomes one of the most important trends [1, 2]. User requirements are increasingly complicated, and no single service could completely fulfill a coarse-grained requirement [3]. Service composition has become increasingly popular in both business and scientific domains [4, 5]. In some situations, end users can't make a thorough experiment plan in advance and cannot decide on which steps to take next without reviewing the results of the previous steps [6]. They may want to

H. Gao and X. Wang (Eds.): CollaborateCom 2021, LNICST 406, pp. 38–53, 2021.
https://doi.org/10.1007/978-3-030-92635-9_3

dynamically adjust the composition logic at runtime. Therefore, exploratory service composition [6] is paid great attention in academics these years. There are a large number of service composition process instances generated at runtime, which may contain valuable knowledge. The knowledge refers to previous experiences and is valuable for reuse. If it can be fully utilized, the efficiency of exploratory service composition will be improved.

Through the analysis of service composition process instances generated by exploratory service composition, it is found that users' selection of services has certain commonalities. Some services often appear together in one service composition process track, as a form of a fragment. The fragment shows the characteristic of large granularity and high reusability. Related research refers to this service fragment which frequently appears as a service pattern [7]. There are two kinds of tracks for service composition process instances generated by exploratory service composition which are successful track and failure track. So, the corresponding service patterns can be divided into two kinds, one is the "success" service pattern, and the other is the "failure" service pattern. Applying these two kinds of service patterns to exploratory service composition, on the one hand, can improve the efficiency of constructing service composition processes, on the other hand, can make full use of previous knowledge to improve the reusability of service composition processes. Recently, most of the work is only focused on the mining of "success" service patterns, and there is little research on the mining of "failure" service patterns. The value of the corresponding historical dataset has not been fully utilized. In addition, if the method of mining "success" service patterns is used in mining "failure" service patterns, it will cause unnecessary waste of time and resources. Because if a track is a failure track, it does not mean the entire track is failed, but only part of the track is failed. Mining can be limited to the scope of failure.

To solve this problem, this paper proposes a method for modeling exploratory service composition instance and its corresponding service pattern and designs a "Failure" Service Pattern Mining Algorithm (FSPMA) which extends the gSpan algorithm. Through the proposed models and algorithm, "failure" service patterns in the failure process tracks can be efficiently mined. Compared with the gSpan algorithm and the TKG algorithm, the mining efficiency is significantly improved.

The rest of the paper is organized as follows. Section 2 of this paper introduces the analysis of related research status and the problems that need to be solved. Section 3 gives the formal definitions of exploratory service composition instance and service pattern. Section 4 elaborates on how to mine the "failure" service patterns based on the FSPMA algorithm that focuses on probe points and failure process tracks. Section 5 designs and implements the prototype. Section 6 gives experimental analysis and performance evaluation of the FSPMA algorithm. Section 7 introduces the application of the "failure" service patterns that are used in service recommendations. Finally, summarize the full paper and look forward to future work.

2 Related Work

2.1 Log-Based Service Pattern Mining

Currently, there are mainly four kinds of log-based service pattern mining algorithms. One is the α algorithm [8]. This algorithm scans all instances in the log, abstracts the basic relationship between activities, and directly constructs the processes according to the type of basic relationships. It can handle various control flow structures, but cannot deal with the noise in the log. The second is the heuristic algorithm [9], which mainly considers the frequency of process instances in the log. It can mine the main behaviors of the processes and deal with the log noise, but it ignores the details of the processes and cannot handle log diversity and quality monitoring. The third is the genetic algorithm [10], which is a search technology that simulates the evolution of biological processes. It can handle various types of control flow structures and log noise at the same time, but the algorithm may not get the optimal process model in the end. The last is based on the log classification algorithm. It clusters the execution instances which are saved in the log [11], divides them into multiple sub-logs, and uses existing mining algorithms on the sub-logs. This algorithm can handle the log diversity well, but it relies on the existing specific mining algorithms.

2.2 Process-Based Service Pattern Mining

Process-based service pattern mining can be abstracted as frequent subgraph mining (FSM). A survey done by literature [12] presents several significant FSM algorithms. According to the survey, no "new" algorithms were proposed recently but there had been much work on developing variations of existing algorithms [13]. FSM algorithms mainly adopt two algorithm ideas, the Apriori algorithm and the FP-growth algorithm.

The mining algorithms that apply Apriori's idea include the AGM algorithm [14] and the improved algorithms based on the AGM algorithm. The AGM algorithm is based on recursive statistics. It can mine all frequent subgraphs, but its execution efficiency is lower for large databases. Its improved algorithms include the FSG algorithm [15] and AcGM [16] algorithm, their execution efficiency is higher than that of the AGM algorithm. In the current research, the representative work of service pattern mining algorithms that use Apriori's idea is shown in the literature [17].

Another algorithm idea is the FP-growth algorithm [18, 19], which compresses the data into a frequent pattern tree, stores the association relationship of the items, and finally generates frequent sets for the pattern tree. Since it does not need to generate candidate frequent sets repeatedly, its execution efficiency is higher than that of the Apriori algorithm. FSM algorithms based on the idea of this algorithm such as gSpan algorithm [20] and FFSM algorithm [21]. The rightmost path expansion and frequent pruning strategy of gSpan algorithm greatly reduce the running time of the algorithm. FFSM algorithm only scans the embedding set when calculating the support degree, which outperforms gSpan. But FFSM cannot be used in the context of directed graphs; while gSpan, with some minor changes, can accommodate directed graphs [12]. Besides, TKG [22], which extends gSpan algorithms, utilizes a dynamic search procedure to always explore the most promising patterns first. Experiments show it has

almost excellent performance as well as gSpan, but its memory use is high. According to the survey, the representative work of applying the idea of the FP-growth algorithm in service pattern mining is present in literature [23].

The related work of service pattern mining above is aimed at the mining of successful service composition process. They mine the entire service composition process. For failed tracks, they will produce unnecessary waste of time and resources. Therefore, this paper comprehensively considers the characteristics of current service pattern mining and its existing problems, adopts a process-based service pattern mining algorithm, considers the mining efficiency and feasibility of these optional algorithms, and finally chooses to extend the gSpan algorithm so that the algorithm can support the mining of "failure" service patterns.

3 Model Definition

This section mainly introduces the exploratory service composition instance model and the service pattern model.

3.1 Exploratory Service Composition Instance Model

Exploratory service composition can support business users to incrementally construct service composition processes at runtime. During the composition, if the intermediate result is incorrect or unsatisfactory, probe points can be added at the corresponding activity, so that new tracks can be derived from the original track until a successful track is explored. The exploratory service composition instance model is shown in Fig. 1.

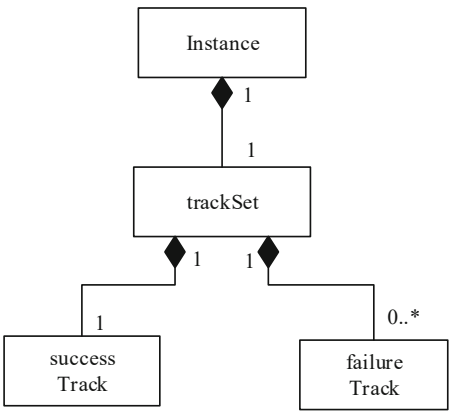

Fig. 1. Exploratory service composition instance model.

Definition 1 (Instance). The *Instance* describes the exploratory execution for certain goals through service composition and is regarded as a package of tracks. It can be

depicted as a 3-tuple: *Instance* = <*instanceID, name, trackSet*>. The *InstanceID* represents the unique identifier of the exploratory service combination instance. The *name* represents the name of the exploratory service composition instance. The *trackSet* represents the track set contained in the instance.

Definition 2 (Track). The *Track* is the ultimate executable process unit, and can be depicted as *Track* = <*trackProfile, instanceID, status, activities, transitions, direct-Deriv, dataPocket, exploredstate*>. The *trackProfile* represents the basic track information, like *trackID*, *name*, *createTime*, etc. The *InstanceID* indicates the instance to which the track belongs, a track must belong to one certain instance, so the *instanceID* cannot be null. The *status* ∈ *{init, running, suspend, complete, terminated}* [4], which represents the execution status of the track. The *activities* and *transitions* represent the activities and the transition relationships of the tracks respectively. The *directDeriv* locates the original track in the derived relationship, which is shown as Fig. 2, defined as *directDeriv* = <*originalTrack, probePoint*>, where *originalTrack* marks the parent track of the current derived track, *probePoint* marks the probe point in the original track when it is derived. And the operation of adding a probe point is shown as Definition 5. The *dataPocket* is the data information generated by the track. The *exploredstate* ∈ *{success, failure}*, represents the track state after explored, which can be successful or failed.

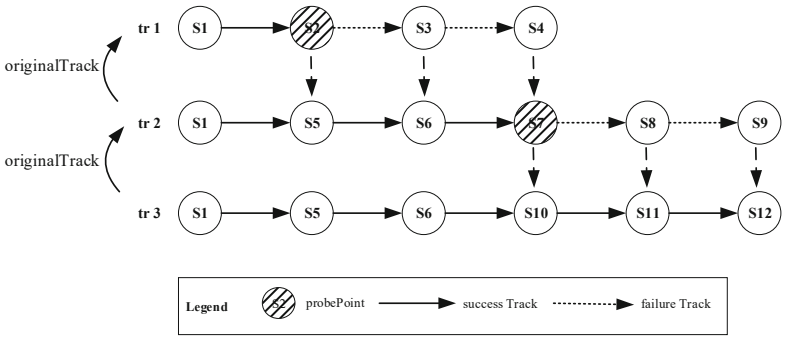

Fig. 2. The original track and its derived tracks.

Definition 3 (Activity). The *Activity* is one of the primary elements of the exploratory service composition instance model and the service pattern model. It is the executable task in service composition processes. It can be depicted as *Activity* = <*activityID, Input, Output, QoS, type, status*>. The *activityID* represents the unique identifier of the activity. The *Input* represents the set of input parameters. The *Output* represents the set of output parameters. The *QoS* means the quality of the activity. The *type* ∈ *{service, start, end, orSplit, orJoin, andSplit, andJoin}* [4] represents the type of the activity. The *status* ∈ *{init, running, suspend, executed}* is the execution status of the activity.

Definition 4 (Transition). The *Transition* is the other primary element of the models and is a mapping from one activity to another activity. It can be depicted as *Transition = <tranID, fromAct, toAct, dataMapping>*. The *tranID* represents the unique identifier of the transition relationship. The *fromAct* represents the source activity of the transition relationship. The *toAct* represents the target activity of the transition relationship. The *dataMapping* $\in$ *{Exact, Plugin, Container, Disjoint}* represents the collection of data mapping relationships between the source activity and the target activity.

Definition 5 (the Operation of Adding a Probe Point). The *Operation of Adding a Probe Point* can be recorded as *addProbePoint (pt, a)*, where *pt* is the track and *a* is the activity. If the *status* of *pt* is not *terminated* and the *status* of *a* is *executed*, then the operation of adding a probe point is allowed. This operation will generate a new track *pt'*. The *dataMapping* of the transition relationship generated by cloning from the original track remains unchanged, so are the predecessor activities of probe point *a*. The probe point *a* and the successor activities are reset and initialized after being cloned.

3.2 Service Pattern Model

After the execution of the exploratory service composition instance, both the successful tracks and the failed tracks can be generated. Two kinds of service patterns can be gained by mining these two kinds of tracks. Considering the characteristics of a service pattern, it can be defined as follows:

Definition 6 (Service Pattern). A service pattern is a frequent subgraph mined from a set of process tracks, which can be depicted as *SP = <spID, activities, transitions, type>*. The *spID* represents the unique identifier of the service pattern. The *activities* are the activities contained in the service pattern. The *transitions* are the transitional relationships that exist between activities. The *type* $\in$ *{success, failure}* represents which kind of tracks the service pattern is mined from.

4 FSPMA

Our problem is defined as: mining the tracks of exploratory service composition instances and obtaining service patterns. And we mainly focus on mining the failure tracks to obtain the "failure" service patterns. To solve the problem, this paper proposes a Failure Service Pattern Mining Algorithm (FSPMA) which extends the gSpan algorithm.

Compared with the gSpan algorithm, the extension of the FSPMA is that it limits the mining scope to the vicinity of the probe points. In the following, the algorithm is introduced in detail. It takes a set of failure tracks as input. Firstly, the input called T is abstracted as a graph set *D*. The predecessor activities of the probe points on the tracks are marked as success points (can also be said as vertices). The successor activities of the probe points are marked as failure points (Algorithm 1 line 1). Secondly, vertices and edges are sorted based on their frequencies. The infrequent vertices and edges will be removed (Algorithm 1 line 2–3), and the remaining frequent edges will be sorted in DFS lexicographical order (Algorithm 1 line 5). Thirdly, to discover "failure" service patterns, frequent subgraphs must contain at least one probe point or failure point. So, the algorithm only performs Subgraph_Mining for three kinds of frequent edges which are shown in Fig. 3. This step is the main part where FSPMA extends the gSpan algorithm (Algorithm 1 line 8). These three types of frequent edges can grow based on the gSpan algorithm's search strategy and growth strategy which are the depth-first search and the rightmost path expansion strategy (Subprocedure 1 line 4). For the strategies, details can be seen in [20] or [22], we will not introduce them further. After mining the frequent subgraphs, remove the mined edge from the original graph (Algorithm 1 line 11), continue with the expansion mining of the next frequent edge which must be one of the three kinds of edges in Fig. 3, and repeat these procedures until all frequent subgraphs are found. Finally, after filtering the frequent subgraphs (Algorithm 1 line 14), we can get the "failure" service patterns.

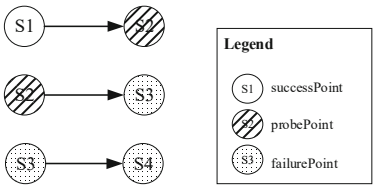

Fig. 3. Three kinds of frequent edges that can grow.

An example is given below to illustrate the operation of the algorithm, as shown in Fig. 4. Given the 8 graphs which are converted from tracks and the minimum support threshold *minSup* = 2, the algorithm sorts vertices and edges according to the frequency, removes infrequent ones, and obtains all frequent edges. Then, only three types of frequent edges which are shown in Fig. 3, can grow based on the depth-first search and the rightmost path expansion. When no new frequent subgraphs are generated, the growth for the next edge starts. Repeat these procedures and get all frequent subgraphs. Finally, remove the repeated frequent subgraphs, and obtain the final result.

Algorithm 1: FSPMA

Input: Track set T, *minSup*
Output: "Failure" service pattern collection *S*

```
1: Abstract T to graph set D and label success points and failure points;
2: sort the labels in D by their frequency;
3: remove infrequent vertices and edges;
4: S^1 ← all frequent 1-edge graphs in D;
5: sort S^1 in DFS lexicographic order;
6: S ← S^1;
7: for each edge e in S^1 do
8:     if both e.from and e.to are failure points
                or e.from is probePoint and e.to is a failure point
                or e.to is probePoint;
9:         Initialize s with e, set s.D by graphs which contain e;
10:        S ← Subgraph-Mining (D, S, s);
11:        D ← D – e;
12:        if |D| < minSup;
13:            break;
14: remove the repeated frequent subgraphs from S;
15: return S;
```

Subprocedure 1: Subgraph-Mining

Input: *D*, "Failure" service pattern collection *S*, frequent fragment *s*, *minSup*
Output: *S*

```
1: if s ≠ min(s)
2:     return S;
3: S ← S ∪ {s};
4: enumerate s in each graph in D and count its children;
5: for each c, c is s' child do
6:     if support(c) ≥ minSup;
7:         s ← c;
8:         S ← Subgraph-Mining (D_s, S, s);
```

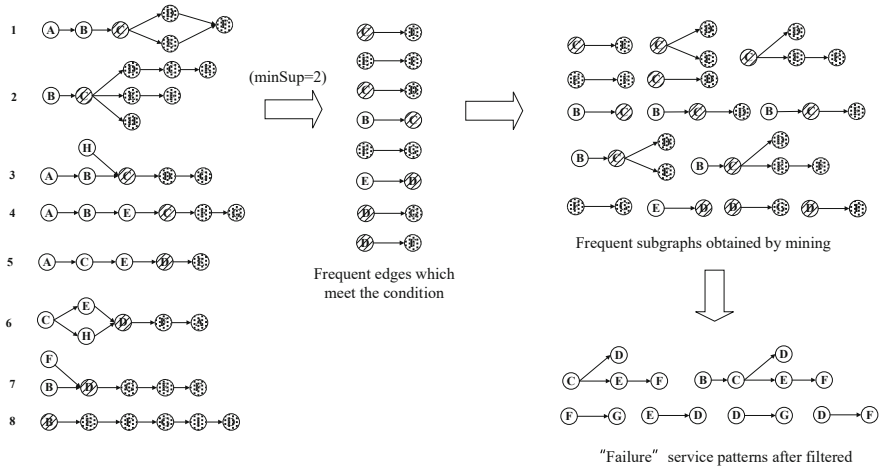

Fig. 4. An example of the algorithm.

5 Prototype Implementation

Exploratory service composition environment is an online service composition system developed to support the FSPMA algorithm. Figure 5 shows its architecture. The environment has five modules, which are the modules of exploratory service orchestration tool, runtime user interaction, execution engine, service pattern mining, and service library.

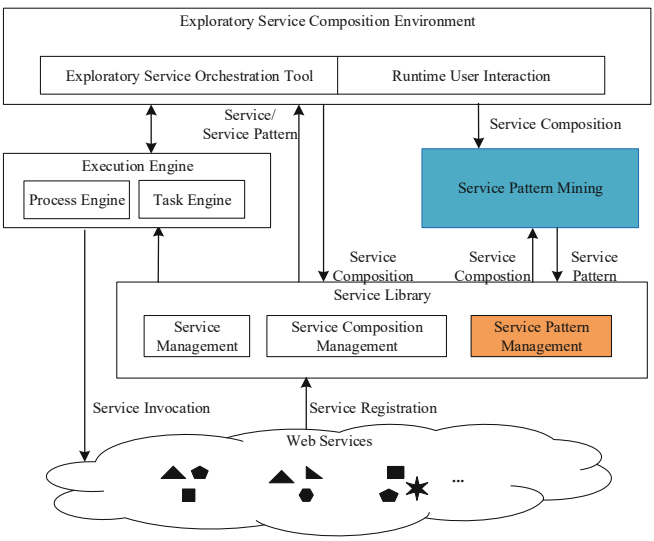

Fig. 5. The architecture of exploratory service composition environment.

The modules of exploratory service orchestration tool, runtime user interaction, execution engine, and service library are the fundamental modules of the environment. The module of exploratory service orchestration tool supports users to build the service composition processes. And users can execute and adjust the processes in the runtime user interaction module. Both the module of exploratory service orchestration tool and the module of runtime user interaction can interact with the execution engine module. There are two engines contained in the execution engine, which are the process engine and the task engine. This module can provide the functions of application parsing, service collaboration, service invocation, and exception handling. The service library module provides services and service patterns to exploratory service orchestration tool and runtime user interaction and saves the service composition processes which are generated by these two modules.

For the mining part, the module of service pattern mining has two data sources, which are the service composition process instances generated by runtime user interaction and the historical service composition process instances saved in the service library. Because the tracks of the process instances can be divided into two kinds, success and failure, this module has two submodules, one of them integrates the FSPMA algorithm to mine "failure" service patterns from failure tracks. The other integrates the gSpan algorithm to mine "success" service patterns from success tracks. These service patterns are written back to the service library in the end. Therefore, the function of service pattern management is developed for the module of the service library, to manage the service patterns.

6 Experiment

6.1 Dataset and Environment

This section uses simulation experiments to evaluate the performance and effectiveness of the FSPMA algorithm. To improve the credibility of the experiment, this paper uses 1405 processes crawled from the myexperiment (www.myexperiment.org) research community as the experimental dataset. The data is in the XML or t2flow format with the highest visits and downloads in the community. This community is a collaborative environment for global biocomputing researchers to publish and share information about biocomputing processes and experimental plans. Scientists can safely publish their processes and experiments, share them with groups, and find other researchers' processes. It can reduce follow-up experiment time, share professional knowledge, and avoid reengineering.

To meet the needs of the algorithm, this paper analyzes the collected biological processes, abstracts services as vertices, sets service names as vertex labels, and abstracts the relationships between services as edges. To obtain more frequent service patterns, we model the most sixteen frequent services as probe points. Since the processes crawled from the website can be executed successfully, failure process tracks are constructed through simulation. A total of 10,490 failure tracks are constructed for the mining algorithm.

The simulation program is developed using Anaconda 3.6 and PyCharm, runs on a PC with a Windows operating system, CPU of AMD Ryzen 7 2.90 GHz, and memory of 16G.

6.2 Experimental Verification

The experiment mainly observes mining efficiency improvement of the FSPMA algorithm compared with the gSpan algorithm and the TKG algorithm. The impacts of *minSup*, the number of tracks, and the number of mining results on the runtime of the algorithms are analyzed while the experiments' mining accuracy is the same. The TKG algorithm can find the top-k frequent subgraphs, where the only parameter is *k*, the number of patterns to be found. It is also an extension of gSpan. Since TGK can generate the optimal minSup according to the value of *k*, we only compare the runtime with the TKG algorithm in terms of the number of tracks and the number of mining results.

The algorithm parameters are mainly the *minSup* and the number of tracks. As shown in Fig. 6, we use 10,000 tracks and set the *minSup* from 3 to 15. The result shows that when the *minSup* is adjusted to 10, the runtime of the FSPMA reduces the most compared with the gSpan algorithm. As can be seen from the histogram, the number of mining results that the two algorithms produce is the same. Besides, by comparing the contents of the output files, the frequent subgraphs obtained by the two algorithms are exactly the same. The mining accuracy of other subsequent experiments is also exactly the same.

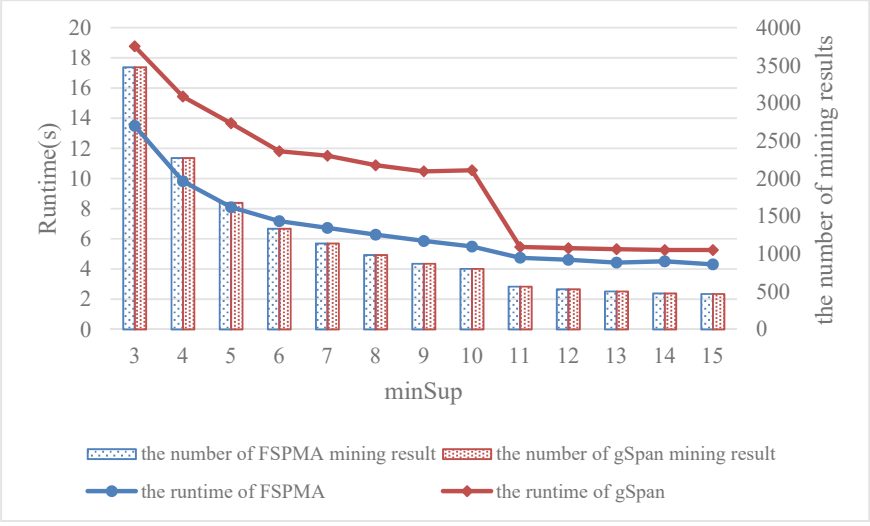

Fig. 6. The impact of *minSup* on runtime.

Then, we select the *minSup* = 10 and analyze the runtime under different numbers of tracks. By changing the number of tracks from 1000 to 10,000, Fig. 7 illustrates the runtimes of these three algorithms and shows the FSPMA achieves better performance compared with the gSpan algorithm and the TKG algorithm. The histogram can show their accuracy is the same.

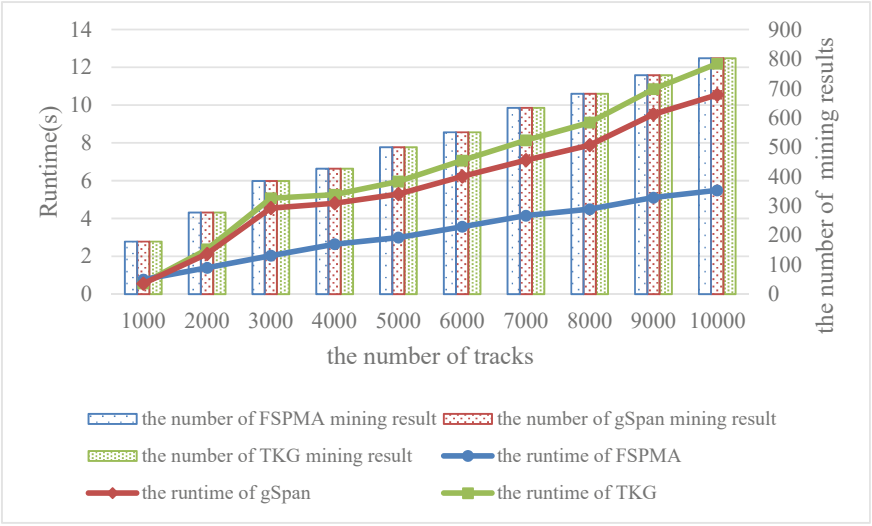

Fig. 7. The impact of the number of tracks on runtime.

In addition, the number of mining results may also affect the runtime of the algorithm. When the *minSup = 10*, the runtimes of the three algorithms under different numbers of mining results are shown in Fig. 8. As the number of mining results increases, the runtime of the FSPMA algorithm is greatly lower than that of the gSpan algorithm and the TKG algorithm.

In summary, the FSPMA algorithm can effectively focus mining on the generation of "failure" service patterns, avoiding the mining of the entire tracks of service composition process instances which the gSpan algorithm and the TKG algorithm do. The efficiency is improved under different *minSup* and the number of tracks. When the *minSup* is set to 10 under the current dataset, the efficiency of FSPMA is improved the most. And under this *minSup*, as the number of tracks increases, the efficiency improvement of the FSPMA algorithm is stable at about 43% compared with the gSpan algorithm, and about 50% compared with the TKG algorithm.

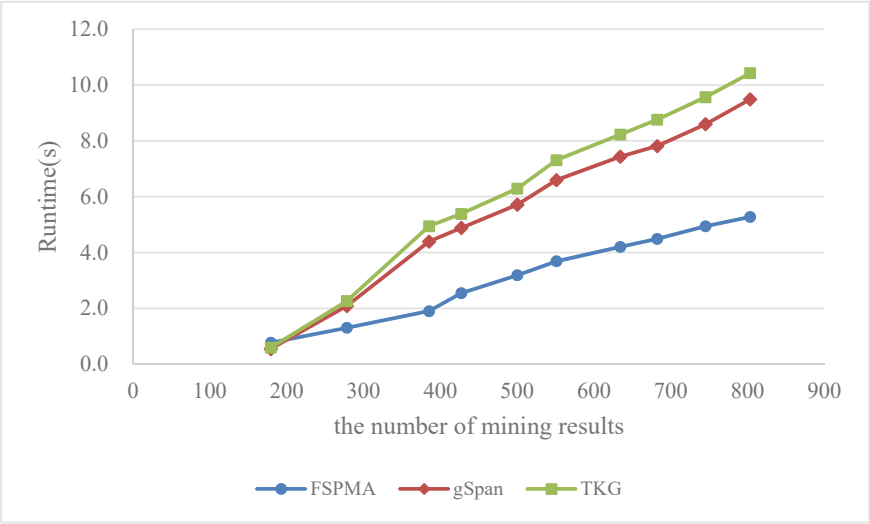

Fig. 8. The impact of the number of mining results on runtime.

7 Application

7.1 Service Recommendation Using "Failure" Service Patterns

In the traditional service recommendation, the operation dependency is used to recommend services that are highly constrained by current services. The operation dependency can be identified by combining the semantic matching of inputs and outputs interfaces between service operations and the analysis of process instances [5]. Through the semantic matching of inputs and outputs interfaces between service operations, the semantic matching degree μ can be calculated. The analysis of process instances can generate the reuse degree λ. These two values are used for recommending the Top N services with high strength of dependencies as candidate services through pattern matching.

However, this recommendation method also recommends the services where their strengths of dependency are high, but there are a large number of error process instances during the analysis of process instance. Therefore, we improve the traditional service recommendation method by using the "failure" service patterns, which can improve the accuracy of service recommendations.

The strengths of dependency (τ) between services considers the frequency degree (γ) of "failure" service patterns which is generated by FSPMA, besides the semantic matching degree (μ) and the reuse degree (λ).

The calculation formula is as follows:

$$\tau = w_1 \times \mu + w_2 \times \lambda - w_3 \times \gamma \tag{1}$$

Among them, w_1 is the weight of the semantic matching degree, w_2 is the weight of reuse degree, w_3 is the weight of the frequency degree of the "failure" service pattern, $w_1 > 0$, $w_2 > 0$, $w_3 > 0$, and $w_1 + w_2 + w_3 = 1$.

When the strength of dependency satisfies $\tau > \chi$, where χ is the minimum dependency strength, it is considered that there may be a relationship of dependency between services. Otherwise, there is no relationship of dependency. Finally, the recommendation can be given by selecting Top N services that satisfy the relationship of dependency.

7.2 An Example

In this section, we use an example of a bioinformatics process to illustrate how the use of the "failure" service patterns can improve the accuracy of service recommendations. As shown in Fig. 9. The traditional method does not consider the "failure" service patterns. The correctness of the process instances cannot be guaranteed during process instance analysis. In this case, *remove_entrez_duplicates, remove_Nulls2, create_report* are recommended. After considering the "failure" service patterns, the fragment *hsapiens_gene_ensembl -> remove_entrez_duplicates* which is used to recommend by the traditional method is a sub-segment of the "failure" service pattern. Although its reuse is high, the wrong process instances are mainly used in the analysis of process instances. This recommendation may cause the execution of the current process to fail. After considering the "failure" service pattern, this problem can be effectively solved, and the recommended results are *remove_Nulls2, create_report, split_for_duplicates*.

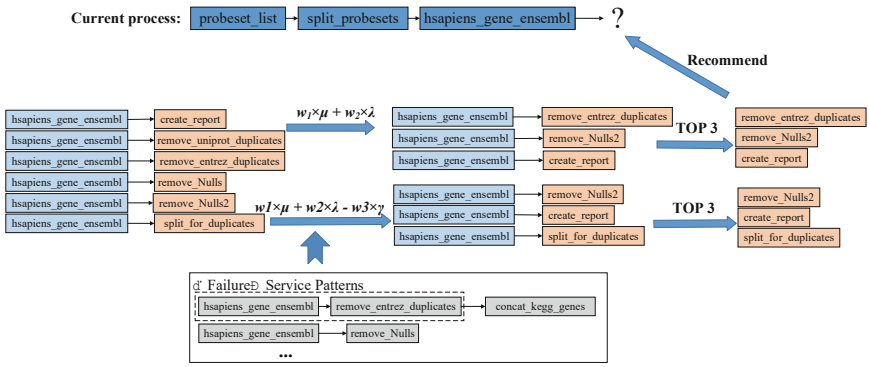

Fig. 9. Comparison of service recommendation using different methods.

8 Conclusion

This paper focuses on the issues related to mining "failure" service patterns from failure tracks of exploratory service composition instances. To make full use of historical service composition process instances, an exploratory service composition instance model and a service pattern model are defined at first, then the FSPMA algorithm is

proposed, which extends the gSpan algorithm and focuses the mining on the vicinity of the probe points in the failure tracks to get "failure" service patterns. The prototype is introduced to support the FSPMA algorithm in the paper. Evaluation experiments are given to compare the FSPMA algorithm with the gSpan algorithm and the TKG algorithm, which proves the effectiveness and practicability of the FSPMA algorithm. Finally, we use the "failure" service patterns to improve the accuracy of service recommendations.

In future research, the incremental mining method will be studied to support the dynamic evolution and timely update of the service patterns in the service library, and the evolutionary trend of service patterns will be analyzed. In addition, the experimental verification of the "failure" service pattern applied to service recommendations will be realized.

Acknowledgements. The research work was supported by the International Cooperation and Exchange Program of National Natural Science Foundation of China (No.62061136006).

References

1. Xu, H., Wang, X., Wang, Y., Li, N., Tu, Z., Wang, Z., Xu, X.: Domain priori knowledge based integrated solution design for internet of services. In: 2020 IEEE International Conference on Services Computing (SCC), pp. 446–453. IEEE, Beijing, China (2020)
2. Liu, R., Wang, Z., Xu, X.: Parameter tuning for S-ABCPK: an improved service composition algorithm considering priori knowledge. Int. J. Web Serv. Res. (IJWSR) **16**(2), 88–109 (2019)
3. Liu, M., Wang, M., Shen, W., Luo, N., Yan, J.: A quality of service (QoS)-aware execution plan selection approach for a service composition process. Futur. Gener. Comput. Syst. **28** (7), 1080–1089 (2012)
4. Ding, W., Wang, J., Han, Y.: ViPen: a model supporting knowledge provenance for exploratory service composition. In: 2010 IEEE International Conference on Services Computing, pp. 265–272. IEEE, Miami, FL, USA (2010)
5. Yan, S., Wang, J., Liu, C.: An approach to discover dependencies between service operations. IEEE Int. J. Software **3**(9), 36–43 (2008)
6. Yan, S., Han, Y., Wang, J., Liu, C., Wang, G.: A user-steering exploratory service composition approach. In: 2008 IEEE International Conference on Services Computing, pp. 309–316. IEEE, Honolulu, HI, USA (2008)
7. Xu, X., Liu, R., Wang, Z., Tu, Z., Xu, H.: RE2SEP: a two-phases pattern-based paradigm for software service engineering. In: 2017 IEEE World Congress on Services (SERVICES), pp. 67–70. IEEE, Honolulu, HI, USA (2017)
8. Fang, X., Gao, X., Yin, Z., Zhao, Q.: An efficient process mining method based on discrete particle swarm optimization. Inf. Technol. J. **10**(6), 1240–1245 (2011)
9. Sangaiah, A.K., Hosseinabadi, A.A.R., Shareh, M.B., Bozorgi Rad, S.Y., Zolfagharian, A., Chilamkurti, N.: IoT resource allocation and optimization based on heuristic algorithm. Sensors **20**(2), 539 (2020)
10. Bratosin, C., Sidorova, N., van der Aalst, W.: Discovering process models with genetic algorithms using sampling. In: Setchi, R., Jordanov, I., Howlett, R.J., Jain, L.C. (eds.) KES 2010. LNCS (LNAI), vol. 6276, pp. 41–50. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15387-7_8

11. Song, M., Günther, C.W., van der Aalst, W.M.P.: Trace clustering in process mining. In: Ardagna, D., Mecella, M., Yang, J. (eds.) BPM 2008. LNBIP, vol. 17, pp. 109–120. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00328-8_11
12. Jiang, C., Coenen, F., Zito, M.: A survey of frequent subgraph mining algorithms. Knowl. Eng. Rev. **28**(1), 75–105 (2013)
13. Wijesinghe, C.R., Weerasinghe, A.R.: Mining frequent patterns in bioinformatics workflows. Int. J. Biosci. Biochem. Bioinf. **10**(4), 161–169 (2021)
14. Zhou, G., et al.: An improved method of AGM for high precision geolocation of SAR images. ISPRS Int. Arch. Photogrammetry Remote Sens. Spat. Inf. Sci. **42**(3), 2479–2485 (2018)
15. Akoglu, L., Tong, H., Koutra, D.: Graph-based anomaly detection and description: a survey. Data Min. Knowl. Disc. **29**(3), 626–688 (2015)
16. Huynh, B., Nguyen, D., Vo, B.: Parallel frequent subgraph mining on multi-core processor systems. ICIC Express Lett. **10**(9), 2105–2113 (2016)
17. Meng, H., Wu, L., Zhang, T., Chen, G., Li, D.: Mining frequent composite service patterns. In: 2008 Seventh International Conference on Grid and Cooperative Computing, pp. 713–718. IEEE, Shenzhen, Guangdong, China (2008)
18. Shafiq, M., Alhajj, R., Rokne, J.: Reducing search space for web service ranking using semantic logs and semantic FP-tree based association rule mining. In: Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015), pp. 1–8. IEEE, Anaheim, CA, USA (2015)
19. Labbaci, H., Medjahed, B., Aklouf, Y.: Learning interactions from web service logs. In: Benslimane, D., Damiani, E., Grosky, W.I., Hameurlain, A., Sheth, A., Wagner, R.R. (eds.) DEXA 2017. LNCS, vol. 10439, pp. 275–289. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-64471-4_22
20. Yan, X., Han, J.: gSpan: graph-based substructure pattern mining. In: 2002 IEEE International Conference on Data Mining, 2002, Proceedings, pp. 721–724. IEEE, Maebashi City, Japan (2002)
21. Fan, Z., Peng, Y., Choi, B., Xu, J., Bhowmick, S.: Towards efficient authenticated subgraph query service in outsourced graph databases. IEEE Trans. Serv. Comput. **7**(4), 696–713 (2014)
22. Fournier-Viger, P., Cheng, C., Lin, J.-W., Yun, U., Kiran, R.U.: TKG: efficient mining of top-K frequent subgraphs. In: Madria, S., Fournier-Viger, P., Chaudhary, S., Reddy, P.K. (eds.) BDA 2019. LNCS, vol. 11932, pp. 209–226. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37188-3_13
23. Liu, R., Xu, X., Wang, Z., Sheng, Q., Xu, H.: Probability matrix of request-solution mapping for efficient service selection. In: 2017 IEEE International Conference on Web Services (ICWS), pp. 444–451. IEEE, Honolulu, HI, USA (2017)