Service Process Improvement Based on Business Process Management

Jia-Xing Wang[1], Si-Bin Gao[2], Cong-Er Yuan[1], Da-Peng Tan[3], Jing Fan[1]

[1] College of Computer Science, Zhejiang University of Technology, China
[2] Department of Computing, Research Institute of CETHIK Group, China
[3] College of Mechanical Engineering, Zhejiang University of Technology, China
wjx@zjut.edu.cn, sibin.gao@gmail.com, yce3612635@126.com, tandapeng@zjut.edu.cn, fanjing@zjut.edu.cn

Abstract

The service vendors desire to improve their service processes to retain consumers and increase profits. Most existing methods require a mass of domain knowledge to improve their service processes, which are time-consuming and error-prone. This paper proposes a method named **Diff-BPI** to automatically improve a service process, reducing execution cost while guaranteeing consumers' waiting/process time. Given the existing two versions of a service process, **Diff-BPI** first detects their differences and all possible candidate improved service processes are constructed by connecting their same parts and combinations of different parts. After that, **Diff-BPI** calculates three metrics for each candidate: "quality of improvement (QoI)", "longest execution time (LET)" and "stability of time (SoT)", and filters the candidates with LETs higher than the given time limit and QoIs/SoTs less than the given one. Finally, **Diff-BPI** picks the best candidate with the lowest cost less than the costs of two existing versions. A case study shows that **Diff-BPI** can construct an improved service process with a lower cost than the existing two versions. The efficiency evaluation reveals that **Diff-BPI** can save more than 20% of the running time for improving a service process using the filter strategy when the number of differences between the two versions is more than 3.

Keywords: Service process, Process difference, Business process improvement

1 Introduction

Nowadays, service vendors provide various service processes to consumers in different scenarios [1-2]. Since the cut-throat competition of market share or even survival, service vendors need to improve their service processes to retain consumers and increase profits [3]. Usually, the service vendor has multiple versions with different control flow patterns and resource allocations for a specific service process, corresponding to different performances, such as execution time and consumer satisfaction [4-5]. Thus, weaken or even cancel the low-efficiency and high-cost parts in the existing versions of a service process, remain the high-efficiency and low-cost parts, and provide a service process with better performance than the existing versions is challenging [6].

Since the control flow of a service process can be modeled as a business process [7-8], the business process improvement (BPI) technique can be introduced to improve the service processes [9]. Most existing BPI methods manually or semi-automatically improve the service processes based on the knowledge gained through process analyzers' experience. However, the service processes' control flows get more complicated than these in the past, so the service process improvement rounds in these methods become time-consuming, expensive, and error-prone [10]. Hence there is an urgent need to develop automatic methods to improve the service processes, which can lower costs through reduced working hours.

To recognize which parts in the service process produce high cost and low efficiency, we compare two existing versions of this service process and detect their differences. Based on these differences, we propose a BPI approach **Diff-BPI** to automatically build an improved service process with a lower cost than the existing versions. First, **Diff-BPI** identifies the differences between two versions of a service process. Since the existing difference detection methods [7, 11-14] just display the compositions of differences and do not point out which control flow patterns a difference belongs to and in which location, we redefine the differences by considering their positions. All possible candidate improved service processes are then constructed by connecting the same parts and the combinations of different parts between two versions. Next, **Diff-BPI** calculates three metrics for each candidate: "quality of improvement (QoI)", "longest execution time (LET)" and "stability of time (SoT)", and filters the candidates with LETs larger than the given time limit and QoIs/SoTs less than the given one.

*Corresponding Author: Jing Fan; E-mail: fanjing@zjut.edu.cn
DOI: 10.53106/160792642021092205015

Finally, **Diff-BPI** selects the best one with the minimal cost less than the existing versions' costs. Therefore, the improved service process can save the service vendor's cost and guarantee the waiting/process time of consumers.

The contributions of this paper are summarized as follows:

(1) The candidate improved service processes are automatically built by connecting the same parts and the combinations of different parts between two versions of a service process.

(2) Three new metrics: quality of improvement, longest execution time and stability of time, are proposed to filter the candidate improved service processes and select the best one with a lower cost than the existing versions while guaranteeing consumers' waiting/process time.

(3) A case study is provided to show the practical use of the proposed method.

(4) The efficiency evaluation shows that **Diff-BPI** can save more than 20% of the running time for building an improved service process using the filter strategy when the number of differences between two existing versions is greater than 3.

The remainder of this paper is structured as follows. Related work and preliminaries are given in Section 2 and Section 3. Section 4 and Section 5 present the details and experiments of **Diff-BPI**. Section 6 concludes this paper.

2 Related Work

BPI has the potential to various aspects concerning a service process, including reduced execution time and cost, and increased consumer satisfaction. There are many methods to improve a service process, albeit under different titles: process innovation, process change, process evolution, and process refactor.

Pyon et al. [16] present a web-based decision support system for business process management employing customer complaints and handling data for service process improvement. Ghattas et al. [10] develop a semi-automated method to improve process performance by learning and deriving decision criteria formulated as decision rules. Beerepoot et al. [15] spot which activities are essential in improvement projects based on organizational size, culture and resources. They use a multiple case study approach to determine how to tackle the improvement organizations of different contexts. Attong et al. [17] provide tools, agendas and activities detailing each of the six stages of BPI. Jin et al. [18] analyzed the real causal relations between business tasks based on data operation dependency analysis, and refactored business process models with process mining technology. In this way, some sequence structures could be refactored to parallel structures, and the efficiency of business processing could be correspondingly improved.

Griesberger et al. [19] analyze the existing BPI techniques and give hints about how to select a suitable method for a specific improvement situation. Yousfi et al. [20] propose a BPI technique based on ubiquitous computing, which positively impacts the process performance metrics. They later introduce ubiquitous decision-aware business processes and explain how to use these processes for improvement [9]. Seethamraju et al. [21] explain the importance and role of process knowledge in the BPI methodology with the help of a case study. Sallos et al. [22] propose a business process improvement framework for knowledge-intensive entrepreneurial ventures, which integrates critical concepts from the knowledge-intensity and knowledge management literature. Iren et al. [23] proposed an approach to facilitate BPI by providing analytical capabilities to detect and resolve requirement conflicts, analyze impacts, and develop actionable BPI plans. Based on the fundamental assumption of improvement in BPM, i.e., redesigns deliver refined and improved versions of business processes, Satyal et al. [24] proposed a middle ground through shadow testing, where a new process version can be simulated using historical data from the old version.

Most existing methods manually or semi-automatically analyze the processes and improve them by identifying areas that can increase effectiveness and efficiency, heavily relying on human knowledge and experience. Due to the differences in knowledge and experience among different experts, the diagnoses made by different experts may be different, leading to inconsistency in the diagnosis results. Specifically, the manual or semi-automatic methods will spend plenty of time and cost to improve the process, which is inefficient and the results cannot be guaranteed. In this paper, **Diff-BPI** automatically construct an improved service process based on the existing versions of a service process with better performance than the existing versions of this service process, which can increase the efficiency of service process improvement.

3 Preliminaries

This section presents a set of preliminaries that are important to set the stage for understanding **Diff-BPI**.

3.1 Service Process Modeling

A service process's structure can be modeled as a business process, described by a directed graph denoted as a tuple $P = (T, G, E)$, where

(1) $T = \{t_1, ..., t_n\}$ is the task node set.

(2) $G = \{g_1, ..., g_m\}$ represents a gateway node set including six types: "And-split", "And-join", "Xor-split", "Xor-join", "Loop-split", "Loop-join".

(3) E is a set of directed edges, where each edge connects two nodes $m, n \in T \cup G$.

There are four basic control flow patterns in a

service process:

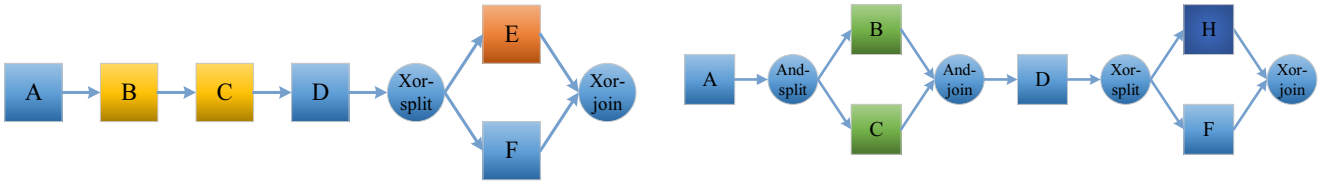
(1) *Sequential pattern*, where each task in it has exactly one incoming and one outgoing edge.

(2) *Parallel pattern*, which starts from "And-split" and ends at "And-join", all tasks in it can be parallelly executed.

(3) *Conditional pattern* that consists of multiple branches, which starts from "Xor-split" and ends at "Xor-join", only one branch is allowed to be executed.

(4) *Loop pattern*, which starts from "Loop-split" and ends at "Loop-join", the tasks in it can be repeatedly executed.

Figure 1 shows two return goods service processes modeled by two business processes. Taking "Service process 1" in Figure 1(a) as an example, its task node set is {A, B, C, D, E, F}, and its gateway node set is {Xor-split, Xor-join}. Besides, "Service process 1" contains one conditional pattern that starts from "Xor-split" and ends at "Xor-join", and either task E or task F can be executed.



(a) Service process 1 (b) Service process 2

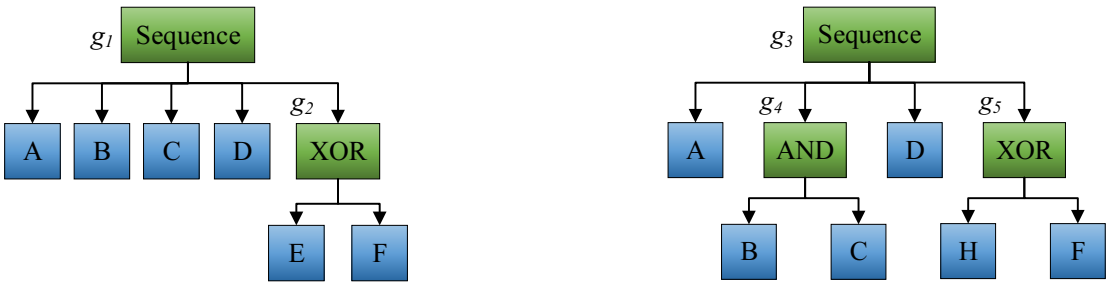
Figure 1. Two return goods service processes modeled by business processes, where *A*: apply for a return, *B*: confirm request, *C*: verify request, *D*: agree to return, *E*: courier pick up, *F*: self-drop off, *H*: post express

3.2 Task-based Process Structure Tree (TPST)

A service process can be decomposed into a hierarchy of sub-processes where each sub-process is a single-entry-single-exit fragment, and such a decomposition can be organized into a process structure tree (PST) [25]. In a PST, a leaf node represents an edge in its corresponding service process. To facilitate the difference detection, we use a variant of PST, i.e., task-based process structure tree (TPST) [26]. The features of a TPST are as follows:

- TPST has four types of gateway nodes: *Sequence*, *Loop*, *XOR*, and *AND*, corresponding to the sequential, loop, conditional, and parallel pattern in a service process.
- The leaf and non-leaf nodes in a TPST separately represent the service process's task nodes and control flow patterns.

Figure 2 shows two TPSTs transformed from two service processes in Figure 1. The leaf nodes colored blue and non-leaf nodes colored green in "TPST 1" and "TPST 2" are separately the task nodes and control flow patterns in two service processes. Taking "TPST 1" in Figure 2(a) as an example, the non-leaf node *Sequence* reveals that the highest abstraction level of "Service process 1" is a sequential pattern, *XOR* and its child nodes E and F correspond to a conditional pattern.



(a) TPST 1 (b) TPST 2

Figure 2. Two task-based process structure trees

3.3 Calculation of Capability Degree

The capability degree of an engineer is calculated based on the satisfaction degrees of service processes he has improved. The service satisfaction degree is rated by consumers using linguistic variables such as "low" and "good", which effectively represent the imprecise information. A linguistic variable can be quantified and extended to mathematical operations using triangle fuzzy numbers (TFNs) [27]. A TFN can be defined as $\tilde{A} = (a^l, a^m, a^r)$, where a^l and a^r denote the minimum value and the maximum value, showing that $\tilde{A}$ is ranged from a^l to a^r, and $a^m = (a^l + a^r) / 2$ is the highest possible value. For example, the linguistic variable set {"Excellent", "Good", "Normal", "Bad"} is used to measure the service satisfaction degree,

where each linguistic variable is quantified by a TFN with three scores (between 0-100): "Excellent" = (85, 92.5, 100), "Good" = (75, 79.5, 84), "Normal" = (60, 67, 74), "Bad" = (0, 29.5, 59).

Usually, the service process improvement requires multiple capabilities c_1, ..., c_n and corresponds to a service satisfaction degree SSD. Thus, the degrees of c_1, ..., c_n owned by the engineer who improves this service process are scored as SSD. As shown in Figure 3, "Service process 1" requires two capabilities "write" and "compute" to improve it and its service satisfaction degree is "Good". Hence the degrees of "write" and "compute" owned by the engineer are both scored as "Good". In addition, each capability degree of an engineer is equal to the average of all satisfaction degrees of service processes he has performed that include this capability. For example, the engineer A improved 3 service processes in the past shown in Figure 3 with satisfaction degrees "Good" = (75, 79.5, 84), "Normal" = (60, 67, 74) and "Excellent" = (85, 92.5, 100), and the capability sets required by these three service processes are separately {"write", "compute"}, {"design", "write"} and {"design", "compute"}. Thus, A's degrees of "write", "design" and "compute" are (67 + 79.5)/2 = 73.25, (67 + 92.5)/2 = 79.75, (79.5 + 92.5)/2 = 86, where 79.5, 67, 92.5 are separately the mean values of "Good", "Normal" and "Excellent". Since 73.25 belongs to (60, 74), the degree of "write" is "Normal". Similarly, the degrees of "design" and "compute" are "Good" and "Excellent", respectively.

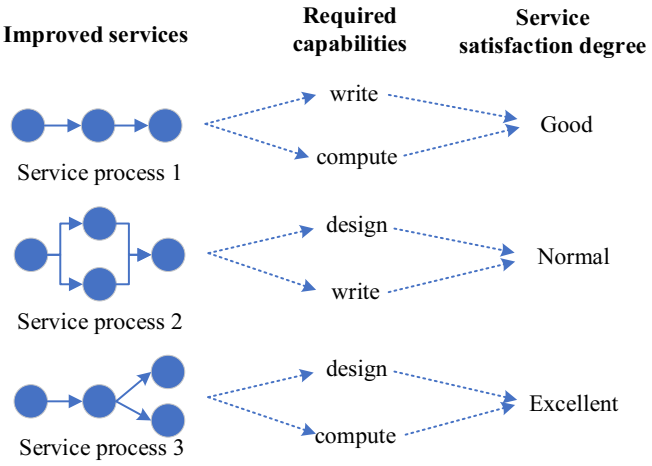


Figure 3. Three improved service processes with required capabilities and service satisfaction degree

Since how to get the capability required for improving a service process and owned by an engineer is not the focus of our work, we directly use the provided capability sets in this paper. Table 1 presents an example, meaning that two service processes in Figure 1 contain two improvements I_1 and I_2, and three engineers A, B, C can be selected to perform these two improvements. The capabilities required for the improvements and owned by the engineers are "read", "write" and "compute", and the capability degree corresponds to four TFNs: "Excellent", "Good", "Normal", and "Bad". Taking capability "read" as an example, the degree required for I_1 is "Normal", and the degree owned by B is "Excellent".

Table 1. Capability table

	I_1	I_2	A	B	C
Read	Normal	Excellent	Normal	Excellent	Good
Write	Bad	Good	Normal	Excellent	Good
Compute	Good	Normal	Good	Good	Normal

3.4 Cost and Execution Time

Different staff spends various execution times and monetary costs performing the tasks in a service process. Thus, an *Allocation Table* is used to record the execution time and monetary cost for each staff to perform the tasks. In the *Allocation Table*, each allocation can be defined as <*s*, *task*, *t*, *c*>, indicating that staff *s* is allocated for executing task *task* with execution time *t* and monetary cost *c*. As shown in Table 2, it records the allocation for staff and tasks. For example, (*a*, *A*, 4, 10) in the second row and second column shows that staff *a* spends 4 hours and $10 performing task *A*.

Table 2. Allocation table

	A	B	C	D	E	F	G	H
a	4,10		5,10			2,10		
b		3,20	2.5,20		5.5,20			
c	2,40			2.5,40		3,40		
d					3,30		2.5,30	3.5,30

3.5 Problem Statement

Given two service processes S_1 and S_2, **Diff-BPI** constructs all possible candidate improved service processes based on combinations of differences between S_1 and S_2, and selects the best improved service process S_{best}, with the following constraints: (1) The quality of improvement and the stability of time are separately greater than the given quality of improvement Q and stability rate S. (2) The longest execution time is not allowed to exceed the given time limit T. (3) The execution cost of S_{best} is minimal among all candidates as well as less than the costs of S_1 and S_2.

4 Diff-BPI Implementation

This section presents the details of implementing **Diff-BPI**. The main idea is to construct all possible candidate improved service processes, filter the candidates that do not meet the constraints, and select the best one with minimal cost. It consists of three phases: (1) *Difference Positioning*, (2) *Improved*

Service Process Construction and (3) *Best Improved Service Process Selection*.

4.1 Phase 1: Difference Positioning

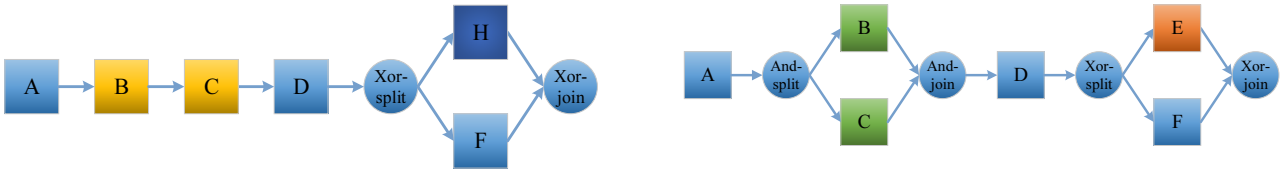
A difference means that two mapped parts in the same position of two service processes are different. Since the existing difference detection methods [7, 11-14] just display the compositions of differences and do not point out which control flow patterns a difference belongs to and in which location, we redefine a difference as $\{[p_1, node_1, pos_1], [p_2, node_2, pos_2]$. For each part $[p, node, pos]$, *node* represents the node set in this difference part, p is the control flow pattern that node belongs to. The positions of node in p are recorded in *pos*, the position of a node is 0 if p is unordered, i.e., p is a parallel or conditional pattern. To achieve this goal, we transform two service processes into their corresponding TPSTs. In this way, we traverse each TPST to determine the control flow pattern that a difference part belongs to and the positions of task nodes in each difference part.

Figure 2 shows two TPSTs transformed from two service processes in Figure 1. There are two differences between these two service processes: (1) the execution order of B and C in "Service process 1" is sequential, while this order becomes parallel in "Service process 2". (2) The conditional pattern in "Service process 1" consists of E and F, while it consists of H and F in "Service process 2". These two differences can be defined as difference$_1$ = $\{[g_1, \{B, C\}, \{2, 3\}], [g_4, \{B, C\}, \{0, 0\}]$, difference$_2$ = $\{[g_2, \{E\}, \{0\}], [g_5, \{H\}, \{0\}]$.

4.2 Phase 2: Improved Service Process Construction

We build all possible candidate improved service processes by connecting the same parts and the combinations of different parts between two service processes in this phase. First, the same parts in two service processes are fixed, and we connect every combination of different parts to these same parts to create all possible candidate improved service processes. In this way, we can construct 2^n-2 candidates if there are n differences between two service processes, where two input service processes are removed.

For example, there are two differences between "Service process 1" and "Service process 2" in Figure 1, corresponding to 2^2-2=2 combinations of differences. After separately connecting two process differences to the same parts in "Service process 1" and "Service process 2", we obtain two candidates "Improved Service process 1" and "Improved Service process 2" shown in Figure 4.



(a) Improved service process 1 (b) Improved service process 2

Figure 4. Two candidate improved service processes

$$md(I_i, A_j) = md(\tilde{I}_i, \tilde{A}_j) = \begin{cases} 1 - \dfrac{A_j^l - I_i^l + A_j^m - I_i^m + A_j^r - I_i^r}{100} \times \dfrac{1}{3}, \\ 0, otherwise \end{cases} \quad (1)$$

$$A_j^l > I_i^l, A_j^m > I_i^m, A_j^r > I_i^r$$

$$MD(I, A) = \begin{cases} \dfrac{1}{n}\sum_{i=1}^{n} md(I_i, A_j), \forall md(I_i, A_j) > 0 \\ 0, otherwise \end{cases} \quad (2)$$

4.3 Phase 3: Best Improved Service Process Selection

In this phase, we propose three metrics: quality of improvement (QoI), longest execution time (LET) and stability of time (SoT) to filter the candidate improved service processes and select the best one with minimal cost, which consists of five steps:

Step 1: Quality of improvement (QoI) calculation. In this step, we first calculate the matching degree in terms of a capability between an improvement and an engineer, then the matching degree in terms of all capabilities between each improvement and each engineering is computed. Finally, the QoI of a service process is calculated depends on the matching degrees between improvements and engineers.

The capability set required for an improvement is recorded as $I = (I_1, ..., I_n)$, and we use a TFN $\tilde{I}_i = (I_i^l, I_i^m, I_i^r)$ $(1 < i < n)$ to represent the degree of each capability I_i. The capability set owned by an engineer is $E = (E_1, ..., E_n)$, where the degree of each E_j is represented as a TFN $\tilde{E}_j = (E_j^l, E_j^m, E_j^r)$ $(1 < j < n)$. To select the most suitable engineer for an improvement, we propose a metric to calculate the

matching degree between improvement and engineer, including three parts: (1) matching degree in terms of a capability between an improvement and an engineer, (2) matching degree in terms of the capability set between each improvement and each engineer, and (3) the QoI of a service process.

(1) Matching degree in terms of a capability between improvement and engineer

In the first part, Equation (1) is used to calculate the matching degree between a capability required for an improvement I_i and a capability owned by an engineer E_j, where I_i and E_j are represented by two TFNs $\tilde{I}_i = (I_i^l, I_i^m, I_i^r)$ and $\tilde{E}_j = (E_j^l, E_j^m, E_j^r)$. There are two cases:

Case 1: The matching degree is 0 if E_j^l, E_j^m, E_j^r are separately lower than I_i^l, I_i^m, I_i^r.

Case 2: If E_j^l, E_j^m, E_j^r are separately larger than I_i^l, I_i^m, I_i^r, each value in $\tilde{E}_j$ and $\tilde{I}_i$ is first normalized by dividing the highest score of 100 and then the average distance of (E_j^l , I_i^l), (E_j^m , I_i^m) and (E_j^r , I_i^r) is calculated. Finally, the matching degree is one minus this average distance.

Table 3 presents the matching degrees between each improvement and each engineer in terms of a specific capability "Read", "Write" or "Compute" shown in Table 1. For example, the value in the second row and second column of Table 3 shows that the matching degree in terms of capability "Write" between I_1 and A is 0.625. According to Table 1, the capability degree required for I_1 is "Bad" = (0, 29.5, 59) and owned by A is "Normal" = (60, 67, 74). This is the second case of Equation (1), so the matching degree between I_1 and A is equal to $1-\frac{60-0+67-29.5+74-59}{100}\times\frac{1}{3}=0.625$. However, since the capability degree owned by engineer C is lower than the capability degree required for improvement I_2, the matching degree between I_2 and C in terms of "Read" is 0.

Table 3. Matching degrees of all pairs of capability degree

	(I_1,A)	(I_1,B)	(I_1,C)	(I_2,A)	(I_2,B)	(I_2,C)
Read	1	0.745	0.875	0	1	0
Write	0.625	0.37	0.5	0	0.87	1
Compute	1	1	0	0.875	0.875	1

(2) Matching degree in terms of the capability set

The matching degree between I = (I_1, ..., I_n) and E = (E_1, ..., E_n) is calculated by Equation (2), which also contains two cases.

Case 1: The matching degree between I and E is 0 if one or more matching degrees between I_i and E_i are 0.

Case 2: If any matching degree between I_i and E_i is larger than 0, i.e., $\forall md(I_i, E_j) > 0$, then the matching degree between I and E is the average of $md(I_1, E_1)$, ...,$md(I_n, E_n)$.

Table 4 lists the matching degrees between any improvement in {I_1, I_2} and any engineer in {A,B,C}. For example, the matching degree between I_1 and A is (1 + 0.625 + 1) / 3 = 0.88, while the matching degree between I_1 and C is 0 since their matching degree in terms of capability "compute" is 0.

Table 4. Matching degrees of all pairs of improvement and engineer

	Improvement I_1	Improvement I_2
Engineer A	**0.88**	0
Engineer B	0.71	**0.92**
Engineer C	0	0

(3) QoI of a service process

The QoI of an improvement I is the maximum matching degree among all matching degrees between I and an engineer who performs it, which is denoted as $MD(I, E)_{max}$. The QoI of a service process is the average matching degree of all its improvements, which can be calculated by Equation (3):

$$QoI(s) = \frac{1}{N}\sum_{i=1}^{N} MD(I, A)_{\max} \quad \textbf{(3)}$$

As shown in Table 4, the maximum matching degrees for I_1 and I_2, i.e., the QoIs of I_1 and I_2, are separately 0.88 and 0.92, which means that engineers A and B are assigned to perform them. Since "Improved service process 1" and "Improved service process 2" separately include improvement I_2 and I_1, their QoIs are 0.92 and 0.88, respectively.

Step 2: Independent path extraction. We introduce the independent path [28] to describe the typical behavior in a service process. An independent path is any path through a service process that introduces at least one new edge or node that is not included in any other independent path. We extract independent paths from a service process using the existing techniques [29]. For example, the independent paths of two improved service processes in Figure 4 are {ABCDH, ABCDF}and {ABCDE, ABCDF}, respectively.

Step 3: Longest execution time (LET) calculation. Since a candidate improved service process has multiple independent paths {ip_1, ..., ip_M}, we calculate the LET for each candidate based on its independent paths. The LET of a candidate is calculates as follows: for each task *task* in an independent path ip_j, the LET of task is selected according to the *Allocation Table* and the sum of each LET for all tasks in ip_j is recorded. The LET among all independent paths is selected as the LET of this candidate. For example, the independent paths of "Improved service process 1" and "Improved service process 2" in Figure 4 are separately {ABCDH, ABCDF} and {ABCDE, ABCDF}. The

allocation for tasks in these two candidate improved processes is shown in Table 2. For the independent path ABCDH, where the LETs of task A, B, C, D and H are 4h, 3h, 5h, 2.5h and 3.5h, so the LET of this independent path is 18h. The LET of the independent path ABCDF is 17.5h. Thus, the LET of "Improved service process 1" is max (18h, 17.5h) = 18h. Similarly, the LET of "Improved service process 2" is 18h.

Step 4: Stability of time (SoT) calculation. The independent path set of each candidate improved service process is $IP = \{ip_1, ..., ip_M\}$. We use a dynamic programming algorithm to compute all possible allocations $\{a_1, ..., a_N\}$ for staff and task in terms of M independent paths. Each allocation a_i ($1 < i < N$) produces an execution time t_i and a cost c_i. In this way, we obtain an execution time set $T = \{t_1, ..., t_N\}$ and a cost set $C = \{c_1, ..., c_n\}$ from each candidate improved service process. The SoT of T recorded as $\Phi(T)$ can be calculated as follows:

$$\Phi(T) = \frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \frac{avg(T)}{avg(T) + |t_i - t_j|} \quad \textbf{(4)}$$

where $avg(T)$ represents the average execution time of T, and the main idea of Equation (4) is to calculate the average fluctuation of T by comparing $avg(T)$ with the fluctuation of each pair (t_i,t_j). The allocation in terms of staff and task for two candidates in Figure 4 is shown in Table 2, and all possible of allocations, execution time and execution cost for an independent path are described in Table 5. According to Equation (4), the SoTs of "Improved service process 1" and "Improved service process 2" are separately 0.88 and 0.87.

Table 5. Allocations, execution time and cost for independent paths

Independent Path	Allocation	Execution Time	Execution Cost
ABCDH	abacd	18	110
	abbcd	15.5	120
	cbacd	16	140
	cbbcd	13.5	150
ABCDF	abaca	16.5	90
	acacc	17.5	120
	abbca	14	100
	abbcc	15	130
	cbaca	14.5	120
	cbacc	15.5	150
	cbbca	12	130
	cbbcc	13	160
ABCDE	abacb	20	100
	abacd	17.5	110
	abbcb	17.5	110
	abbcd	15	120
	cbacb	18	130
	cbacd	15.5	140
	cbbcb	15.5	140
	cbbcd	13	150

Step 5: Filter strategy design and best improved service process selection. In this step, we filter the candidate improved service processes that are not meet the constrains and select the best one from the rest candidates. First, the service processes that their QoIs are less than the given QoI Q will not be built. For example, only "Improved service process 1" in Figure 4 is constructed if we set Q to 0.9. Then, the candidates that their LETs exceed the given time limit T are filtered. For example, the LET of "Improved service process 1" in Figure 4 is 18h, so "Improved service process 1" is left if we set T to 20h. Next, we compute the SoTs of the rest candidates, where the ones that their SoTs are less than the given SoT S are filtered. For example, "Improved service process 1" is still not filtered if we set S to 0.8. Since the SoT calculation involves expensive computations of the allocation for staff and tasks concerning the candidates' independent paths, the first step of this filter strategy can significantly decrease the number of candidate, saving the execution time of SoT calculation for filtered candidates.

For the remaining candidates, we compare their average execution costs and select the one with minimal cost less than the costs of the existing two versions. For example, the average cost of "Improved service process 1" is \$126.7, and the costs of the existing two service processes "Service process 1" and "Service process 2" in Figure 1 are \$125 and \$126.7. Thus, we fail to build the improved process, because the constraint (3) in Problem Statement is not met.

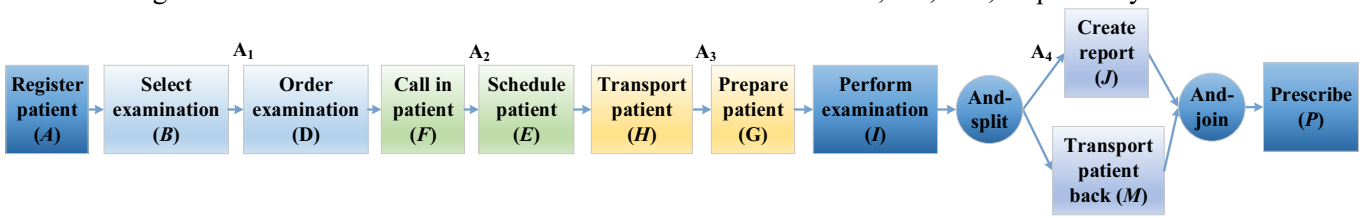
5 Experimental Evaluation

This section provides the case study and efficiency evaluation of **Diff-BPI**. All experiments were evaluated on a machine with Intel(R) Core(TM) i7-6650U CPU @ 2.20GHZ 2.21 GHz, running JDK 1.8 and Windows 10. We use the existing method **Pro-Diff** [14] to detect differences between two business processes modeled service processes. **Pro-Diff** first summarizes two groups of difference patterns from the node level and control flow level: (1) node edit difference patterns, and (2) control flow difference patterns. Then, **Pro-Diff** finds out the difference patterns between two business processes at different levels of abstraction. In this experiment, we use the difference patterns at the lowest level of abstraction.

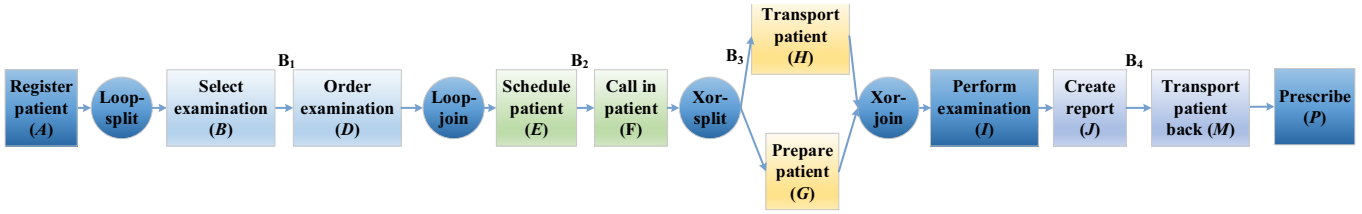
5.1 Case Study

This section discusses how to apply **Diff-BPI** to improve the service process in a real-life scenario. The service processes used in this case study are conducted based on a mixture of real and synthetic data sets, where the real part comes from the process models used in the health care domain [30]. We choose one service process from the real part and make some

modifications to it, i.e., remove/insert some nodes and edges from/to this service process, to obtain two different versions, as shown in Figure 5. The goals of V_1 and V_2 are the same, i.e., a physical examination for the patients. Two TPSTs converted from V_1 and V_2 are shown in Figure 6. Table 6 shows the allocation of medical staff for V_1 and V_2, where the units of time and cost are separately "minute" and "$". There are three kinds of staff: staff *a* is receptionist, staff *b* and *c* are nurses, staff *d* and e are doctors. We set the given quality of improvement Q, stability rate S, and time limit T to 0.94, 0.9, 100, respectively.

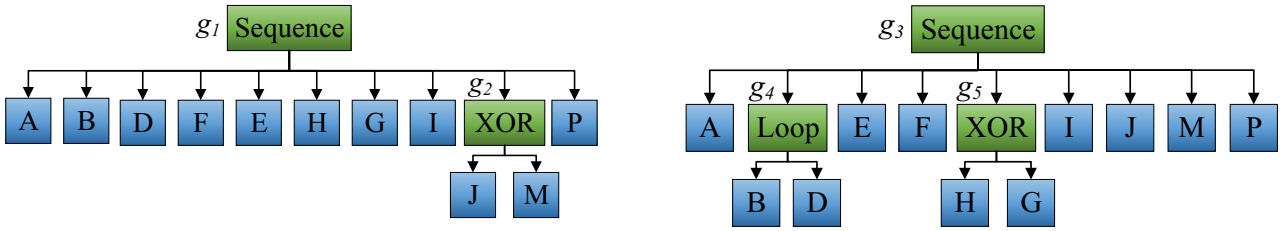


(a) One version of a service V_1



(b) The other version of a service process V_2

Figure 5. Two versions of a service process



(a) TPST of V_1 (b) TPST of V_2

Figure 6. TPSTs of two service processes in case study

Table 6. Allocation of case study

	A	B	D	E	F	H	G	I	J	M	P
a	5,10	1,3	1,2								
b				10,10		15,25	13,50			16,20	
c				5,20	2,4	20,30					
d								20,40	10,50		
e									5,60		15,30

There are four differences between V_1 and V_2: (A_1, B_1) = {[g_1, {B,D}, {2,3}], [g_4, {B,D}, {1,2}], (A_2, B_2)= {[g_1, {F,E}, {4,5}], [g_3, {E,F}, {3,4}], (A_3, B_3) = {[g_1, {H,G}, {6,7}], [g_5, {H,G}, {0,0}], (A_4, B_4) = {[g_2, {J,M}, {0,0}], [g_3, {J,M}, {7,8}], which are separately colored light blue, green, yellow and purple. These four differences correspond to four improvements: I_1, I_2, I_3, I_4, and the description of these four differences are as follows:

(1) The patient can only select and order the examination once in V_1, while the patient can select and order examination multiple times in V_2.

(2) The patient is first called in and then scheduled in V_1, while the orders of these two tasks are interchanged in V_2.

(3) After the patient is called in and scheduled, this patient will be transported and prepared in V_1, while this patient will be either transported or prepared in V_2.

(4) Creating a report and transporting the patient back in V_1 can be performed simultaneously while sequentially executing these two tasks in V_2.

Table 7 shows the details of the capabilities required for the improvements and owned by the engineers. According to the matching degrees between all pairs of

improvement and engineer shown in Table 8, improvements (abbreviated as Imp.) I_1, I_2, I_3, I_4 are separately assigned to engineers D, A, C, D. This result shows that the proposed matching degree metric can assign the most suitable engineer to perform the improvements. For I_2, engineers A, B, C, D can improve it, while A is selected with the least waste of capability, and other engineers with higher skill levels can be assigned to perform the improvements with higher capability requirements. Based on these four differences, we can construct 14 candidate improved service processes P_1 - P_{14}.

Table 7. Capability required for improvements in case study

Capability	Imp. I_1	Imp. I_2	Imp. I_3	Imp. I_4	Engineer A	Engineer B	Engineer C	Engineer D	Engineer E
Analysis	Excellent	Bad	Good	Good	Normal	Excellent	Excellent	Excellent	Normal
Compute	Good	Normal	Good	Good	Normal	Excellent	Good	Good	Bad
Design	Excellent	Normal	Good	Excellent	Normal	Excellent	Good	Excellent	Good

Table 8. Matching degrees between improvements and engineers

	I_1	I_2	I_3	I_4
A	0	**0.875**	0	0
B	0.96	0.62	0.87	0.91
C	0	0.71	**0.96**	0
D	**1**	0.67	0.91	**0.96**
E	0	0	0	0

Table 9 shows the details of 14 candidate improved service processes, including the combination of differences, the QoI, LET, SoT and execution cost. We set the execution time of the loop pattern in V_2 to 2, i.e., the examination can be selected and ordered twice. We first filter the candidates with QoIs less than 0.93, i.e., P_3, P_8, P_{10} and P_{14} will not be built. Since the SoTs of all rest candidates are larger than 0.9, no candidate in the remaining is filtered. Then, the candidates that their LETs more than 100 are filtered, so P_1, P_2 are left, and other candidates are filtered. Finally, we calculate the average cost of the independent paths for these three candidates. Since they have the same average cost of \$214, anyone of them can be selected as the optimal one. The average costs of V_1 and V_2 are \$256.5 and \$219, so the improved service process has a lower cost than any of them. For example, P_2 is chosen as the best improved service process, as shown in Figure 7. In this way, we construct an improved service process with better performance than the existing ones.

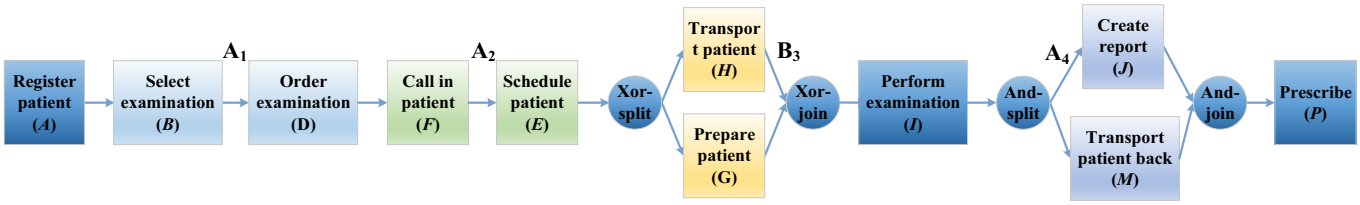


Figure 7. The best improved service process P_2

Table 9. Difference combinations, QoI and LET of each improved service process in case study

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}
Combination	A_1,A_2, B_3,B_4	A_1,A_2, B_3,A_4	A_1,B_2, B_3,B_4	B_1,A_2, B_3,A_4	B_1,B_2, A_3,A_4	B_1,B_2, A_3,B_4	B_1,A_2, A_3,B_4	A_1,B_2, A_3,B_4	A_1,A_2, A_3,B_4	A_1,B_2, A_3,A_4	B_1,B_2, B_3,A_4	B_1,A_2, B_3,B_4	B_1,A_2, A_3,A_4	A_1,B_2, B_3,A_4
QoI	0.96	0.96	0.93	0.98	0.94	0.95	0.98	0.92	0.96	0.88	0.95	0.96	1	0.92
LET	100	100	-	102	115	115	115	-	113	-	102	102	115	-
SoT	0.943	0.943	-	-	-	-	-	-	-	-	-	-	-	-
Cost	214	214	-	-	-	-	-	-	-	-	-	-	-	-

5.2 Efficiency Evaluation

In this section, we first evaluate the efficiency of **Diff-BPI** in terms of the running time for a pair of service processes, where the difference detection and independent path extraction are implemented based on [14] and [29]. Then, the running time in terms of comparing a given version of a service process with other versions in a repository is studied. The experiments in this section are based on a mixture of real and synthetic data sets, where the real parts come from an existing data set of IBM [31]. The synthetic part is some modifications made on the real parts for the sake of evaluations, i.e., remove/insert some nodes and some edges from/to the models. In this way, we build four data sets as follows:

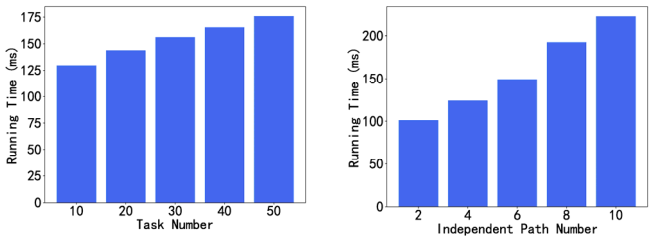
(1) Six business process modeled service processes T_1 - T_6 with the same number of independent path and difference, and their task numbers are 30, 10, 20, 30, 40, 50, respectively.

(2) A service process repository consists of 30 service processes.

(3) Five service processes T_7 - T_{11} have the same number of task and difference, while their independent path numbers are 2, 4, 6, 8, 10, respectively.

(4) Six service processes T_{12} - T_{17} with the same number of task and independent path, the numbers of difference between each service process in T_{13} - T_{17} and T_{12} are ranged from 2, 3, 4, 5, 6.

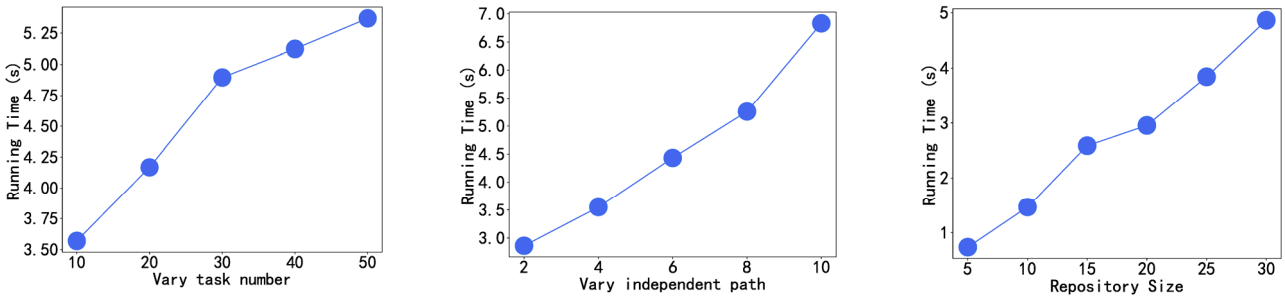
Figure 8(a) applies the first data set to study the impact of task number on the running time of **Diff-BPI**, where T_2 - T_6 are separately compared with T_1. The numbers of difference between each compared pair of service processes are the same. Figure 8(b) uses T_1 and the third data set to evaluate the impact of independent path number on the running time of **Diff-BPI**, where T_7 - T_{11} are compared with T_1 one by one. From Figure 8(a) and Figure 8(b) we can see that the running time is slightly growing as the task number is getting larger, while the running time is dramatically growing with the independent path number becomes larger. The reason behind this is that more task nodes cause more time for processing the task nodes. The more independent path in a service process, the more time for allocating staff to tasks include, leading to more time spent in longest execution time and stability of time calculation.



(a) Vary task number (b) Vary independent path

Figure 8. Efficiency evaluation results for two service

In Figure 9(a), T_2 - T_6 with different task numbers are used to compare with the service process repository. Figure 9(b) applies T_7 - T_{11} with different independent paths to compare with the service process repository. In Figure 9(c), 5, 10, 15, 20, 25, 30 service processes in the service process repository are separately compared with T_1. Figure 9(a) and Figure 9(b) shows that the total running time spent by **Diff-BPI** increases approximately linearly as the number of task nodes or independent paths goes up. We can observe that the change of independent path number is more sensitive than the change of task number in terms of running time. This is because the more independent paths lead to more time spent in longest execution time and stability of time calculation.



(a) Vary task number (b) Vary independent path (c) Vary repository size

Figure 9. Efficiency evaluation results based on the repository

Figure 10 result between with filter and without filter compares the results with and without filter strategy using the fourth data set, where the impact of numbers of difference between two service processes on the running time of **Diff-BPI** is studied. We can observe that the execution time increases nearly exponentially concerning the difference number between a couple of service processes before using the filter strategy. It is because that we calculate all metrics for every candidate, and the optimal one is selected from all candidates, which is time-consuming. While once we use the filter strategy, the execution time significantly decreases. For example, the execution time of using filter decreases by approximately 20% compared with no filter strategy when the difference number between two versions of a service process is more than 3. The reason behind this is that the filter strategy can significantly decrease the number of candidate improved service processes. Notably, the running time for LET and SoT calculation about the filtered candidates is saved. The results show that the filter strategy saves time for building the improved service process.

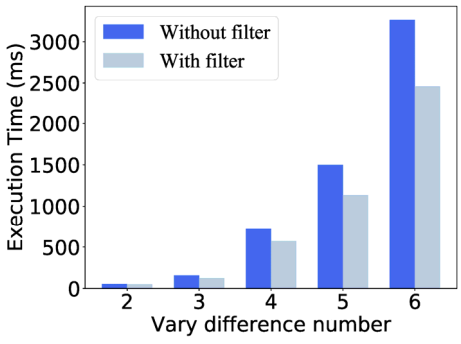


Figure 10. Comparison result between with filter and without filter

6 Conclusion

This paper aims for the consumer to receive the best service process from the service vendors. Therefore, this paper proposes **Diff-BPI**, a difference detection based BPI method for improving service processes. **Diff-BPI** can construct an improved service process with better performance than its existing two versions. A case study is conducted on a pair of service processes in a real-life scenario to show the practical use of **Diff-BPI**, and the efficiency evaluation reveals the effectiveness and extensibility. The limitation of this paper is that the proposed method is too simple to satisfy the real requirements. In the future, we will consider more kinds of differences between two service processes and more real BPI requirements to deal with practical problems. Besides, the reinforcement learning methods will be introduced to schedule the service process resource with resource constraints.

Acknowledgments

This research was supported by National Key Research & Development Program of China (No. 2018 YFB1402802), National Natural Science Foundation of China (No. 6210070439) and China Postdoctoral Science Foundation (No. 2019M660145).

References

[1] H. Gao, L. Kuang, Y. Yin, B. Guo, K. Dou, Mining Consuming Behaviors with Temporal Evolution for Personalized Recommendation in Mobile Marketing Apps, *ACM/Springer Mobile Networks and Applications*, Vol. 25, No. 4, pp. 1233-1248, August, 2020.

[2] L. Li, D. Tan, T. Wang, Z. Yin, X. Fan, R. Wang, Multiphase coupling mechanism of free surface vortex and the vibration-based sensing method, *Energy*, Vol. 216, Article No. 119136, February, 2021.

[3] H. Gao, W. Huang, Y. Duan, The Cloud-edge-based Dynamic Reconfiguration to Service Workflow for Mobile Ecommerce Environments: A QoS Prediction Perspective, *ACM Transactions on Internet Technology*, Vol. 21, No. 1, Article No. 6, February, 2021.

[4] H. Gao, C. Liu, Y. Li, X. Yang, V2VR: Reliable Hybrid-Network-Oriented V2V Data Transmission and Routing Considering RSUs and Connectivity Probability, *IEEE Transactions on Intelligent Transportation Systems*, pp. 1-14, April, 2020, DOI: 10.1109/TITS.2020.2983835

[5] Y. Yin, Z. Cao, Y. Xu, H. Gao, R. Li, Z. Mai, QoS Prediction for Service Recommendation with Features Learning in Mobile Edge Computing Environment, *IEEE Transactions on Cognitive Communications and Networking*, Vol. 6, No. 4, pp. 1136-1145, December, 2020.

[6] H. Gao, X. Qin, R. J. D. Barroso, W. Hussain, Y. Xu, Y. Yin, Collaborative Learning-based Industrial IoT API Recommendation for Software-defined Devices: The Implicit Knowledge Discovery Perspective, *IEEE Transactions on Emerging Topics in Computational Intelligence*, pp. 1-11, September, 2020, DOI: 10.1109/TETCI.2020.3023155

[7] J. Wang, B. Cao, J. Fan, T. Dong, FB-Diff: A feature based difference detection algorithm for process models, *IEEE International Conference on Web Services*, Honolulu, HI, USA, 2017, pp. 604-611.

[8] C. Liu, L. Cheng, Q. Zeng, L. Wen, Formal Modeling and Discovery of Hierarchical Business Processes: A Petri Net Based Approach, *IEEE Transactions on Systems, Man and Cybernetics: Systems*, in press, 2021.

[9] A. Yousfi, K. Batoulis, M. Weske, Achieving business process improvement via ubiquitous decision-aware business processes, *ACM Transactions on Internet Technology*, Vol. 19, No. 1, pp. 1-19, March, 2019.

[10] J. Ghattas, P. Soffer, M. Peleg, Improving business process decision making based on past experience, *Decision Support Systems*, Vol. 59, pp. 93-107, March, 2014.

[11] C. Liu, Q. Zeng, L. Cheng, H. Duan, J. Cheng, Measuring Similarity for Data-aware Business Processes, *IEEE Transactions on Automation Science and Engineering*, pp. 1-13, January, 2021. DOI: 10.1109/TASE.2021.3049772

[12] N. V. Beest, M. Dumas, L. Garcia-Banuelos, M. L. Rosa, Log Delta Analysis: Interpretable Differencing of Business Process Event Logs, in: H. Motahari-Nezhad, J. Recker, M. Weidlich (Eds.), *Business Process Management. BPM 2016*, Vol. 9253, Springer, Cham, 2015, pp. 386-405.

[13] J. Wang, J. Lu, B. Cao, J. Fan, D. Tan, KS-Diff: A key structure based difference detection method for process models, *IEEE International Conference on Web Services*, Milan, Italy, 2019, pp. 408-412.

[14] B. Cao, J. Wang, J. Fan, S. Deng, J. Yang, W. Zhao, J. Yin, M. Zhou, Pro-Diff: A process difference detection method based on hierarchical decomposition, *IEEE Transactions on Services Computing*, pp. 1-14, November, 2019. DOI: 10.1109/TSC.2019.2953853

[15] I. Beerepoot, I. van de Weerd, H. A. Reijers, Business process improvement activities: differences in organizational size, culture, and resources, *International Conference on Business Process Management*, Vienna, Austria, 2019, pp. 402-418.

[16] C. U. Pyon, J. Y. Woo, S. C. Park, Service improvement by business process management using customer complaints in financial service industry, *Expert Systems with Applications*, Vol. 38, No. 4, pp. 3267-3279, April, 2011.

[17] M. Attong, T. Metz, *Change or die: The business process improvement manual*, Productivity Press, 2017.

[18] T. Jin, J. Wang, Y. Yang, L. Wen, K. Li, Refactor business process models with maximized parallelism, *IEEE Transactions on Services Computing*, Vol. 9, No. 3, pp. 456-468, May-June, 2016.

[19] P. Griesberger, S. Leist, G. Zellner, Analysis of techniques for business process improvement, *European Conference on Information Systems*, Helsinki, Finland, 2011, pp. 1-13.

[20] A. Yousfi, A. De Freitas, A. K. Dey, R. Saidi, The use of

ubiquitous computing for business process improvement, *IEEE Transactions on Services Computing*, Vol. 9, No. 4, pp. 621-632, July-August, 2016.

[21] R. Seethamraju, O. Marjanovic, Role of process knowledge in business process improvement methodology: a case study, *Business Process Management Journal*, Vol. 15, No. 6, pp. 920-936, November, 2009.

[22] M. P. Sallos, E. Yoruk, A. Garcia-Perez, A business process improvement framework for knowledge-intensive entrepreneurial ventures, *The Journal of Technology Transfer*, Vol. 42, No. 2, pp. 354-373, April, 2017.

[23] D. Iren, H. A. Reijers, Leveraging business process improvement with natural language processing and organizational semantic knowledge, *International Conference on Software and System Process*, Paris France, 2017, pp. 100-108.

[24] S. Satyal, I. Weber, H. Y. Paik, C. D. Ciccio, J. Mendling, Shadow Testing for Business Process Improvement, OTM Confederated International Conferences "On the Move to Meaningful Internet Systems", Valletta, Malta, 2018, pp. 153-171.

[25] J. Vanhatalo, H. Volzer, J. Koehler, The refined process structure tree, *Data & Knowledge Engineering*, Vol. 68, No. 9, pp. 793-818, September, 2009.

[26] J. Fan, J. Wang, W. An, B. Cao, T. Dong, Detecting difference between process models based on the refined process structure tree, *Mobile Information Systems*, Vol. 2017, pp. 1-17, March, 2017.

[27] M. G. Voskoglou, Application of fuzzy numbers to assessment of human skills, *International journal of fuzzy system applications*, Vol. 6, No. 3, pp. 59-73, July, 2017.

[28] J. Wang, D. Tan, B. Cao, J. Fan, S. Deep, Independent path-based process recommendation algorithm for improving biomedical process modelling, *Electronics Letters*, Vol. 56, No. 11, pp. 531-533, May, 2020.

[29] B. Cao, F. Hong, J. Wang, J. Fan, M. Lv, Workflow difference detection based on basis paths, *Engineering Applications of Artificial Intelligence*, Vol. 81, pp. 420-427, May, 2019.

[30] C. Li, M. Reichert, A. Wombacher, The minadept clustering approach for discovering reference process models out of process variants, *International Journal of Cooperative Information Systems*, Vol. 19, No. 3-4, pp. 159-203, September & December, 2010.

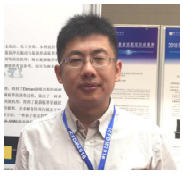
[31] D. Fahland, C. Favre, B. Jobstmann, J. Koehler, N. Lohmann, H. Volzer, K. Wolf, Instantaneous soundness checking of industrial business process models, *International Conference on Business Process Management*, Ulm, Germany, 2009, pp. 278-293.

Biographies

Jia-Xing Wang received the Ph.D. degree in control science and engineering from Zhejiang University of Technology, China, in 2019. She is now a postdoc in the College of Computer Science and Technology at Zhejiang University of Technology. Her research interest is business process management.

Si-Bin Gao received the master's degree in College of Computer Science from Zhejiang University of Technology, China, in 2017. He is now an algorithm engineer in the Cethik Group Co., Ltd. His research interest is artificial intelligence and deep learning.

Cong-Er Yuan is the undergraduate student in the College of Computer Science and Technology at Zhejiang University of Technology. His major is software engineering.



Da-Peng Tan received Ph.D. degree from the College of Mechanical and Energy Engineering, Zhejiang University, China, in 2008. He is now a Professor of College of Mechanical and Energy Engineering. He performed research in the areas of embedded system technology, advanced manufacturing technology, and metallurgy process automatic control. His research interests include real-time intelligent networks and industrial process monitoring and diagnosis.

Jing Fan received her B.S., M.S. and Ph.D. degree in Computer Science from Zhejiang University, China in 1990, 1993 and 2003. She is now a Professor of School of Computer Science and Technology, and Director of Institute of Computer Software at Zhejiang University of Technology, China. She is a Director of China Computer Federation (CCF), and Member of CCF Technical Committee on Service Computing. Her current research interest includes Service Computing, Software Middleware.