This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867593, IEEE Transactions on Services Computing

Workflow Refactoring for Maximizing Concurrency and Block-Structuredness

Wei Song, *Member, IEEE,* Hans-Arno Jacobsen, *Senior Member, IEEE,*
S. C. Cheung, *Senior Member, IEEE,* Hongyu Liu, and Xiaoxing Ma, *Member, IEEE*

Abstract—In the era of Internet and big data, contemporary workflows become increasingly large in scale and complex in structure, introducing greater challenges for workflow modeling. Workflows are not with maximized concurrency and block-structuredness in terms of control flow, though languages supporting block-structuredness (e.g., BPEL) are employed. Existing workflow refactoring approaches mostly focus on maximizing concurrency according to dependences between activities, but do not consider the block-structuredness of the refactored workflow. It is easier to comprehend and analyze a workflow that is block-structured and to transform it into BPEL-like processes. In this paper, we aim at maximizing both concurrency and block-structuredness. Nevertheless, not all workflows can be refactored with a block-structured representation, and it is intractable to make sure that the refactored workflows are as block-structured as possible. We first define a well-formed dependence pattern of activities. The control flow among the activities in this pattern can be represented in block-structured forms with maximized concurrency. Then, we propose a greedy heuristics-based graph reduction approach to recursively find such patterns. In this way, the resulting workflow is with maximized concurrency and its block-structuredness approximates optimality. We show the effectiveness and efficiency of our approach with real-world scientific workflows.

Index Terms—Workflow refactoring, activity dependence, concurrency maximization, block-structuredness, synchronization links

1 Introduction

Workflows or processes are a series of inter-related activities targeted at achieving specific business functions. With the rapid advancement of cloud technology and the availability of ever more services, workflows are becoming one of the mainstream ways to construct software systems [1], [2], [3]. Meanwhile, service-based workflows grow increasingly large and their structure is becoming increasingly complex [1]. Modelling and changing workflows are a challenge [1], [3], [4]. Generally, workflows are expected to be free of both control flow errors (e.g., deadlocks) and data flow errors (e.g., activity input is undefined) [5], [6], [7], [8]. Maximizing concurrency in the control flow, with respect to the data flow, is also critical, and has recently received more attention in the literature [9], [10], [11], [12]. *Concurrency maximization* refers to the property that activities without dependences (including control dependences, data dependences [13]) should be executed in parallel. In this paper, we study BPEL-like workflows (workflows for short) [14]. However, even for the XML style, as pointed out in a recent empirical study [1], not all developers are familiar with good design practices for BPEL-like workflows. Thus, it is questionable whether real-world workflows are modeled with maximized concurrency.

While most control flow and data flow errors lead to failures of the workflow [5], [7], [15], a workflow does not necessarily fail if it is not designed with maximized concurrency. However, such non-maximized concurrency does affect the quality of workflows [16]; for example, the makespan of a workflow could increase in such situations. In the worst case scenario, non-maximized concurrency between communicating activities could reduce the success probability of a workflow (service) collaboration [10]. Take the workflows P_1 and P_2 in Fig. 1, for example: P_1 and P_2 share compatible interfaces (communicating activities) while P_1 (P_2) invokes (notation '!') the operation B (A) provided by P_2 (P_1) before receiving (notation '?') the invocation of its provided operation A (B) from P_2 (P_1). Assume that activities $?B$ and $!A$ in P_1 are independent, that is, P_1 is not designed with maximized concurrency because of the unnecessary `sequence` order between $?B$ and $!A$. This prevents P_1 from being composed with P_2 (and other similar workflows) to avoid a potential deadlock. However, if $?B$ and $!A$ are executed in parallel, then, P_1 and P_2 can be composed, thus, improving the success probability of the workflow collaboration. For the reasons identified above, non-maximized concurrency should be avoided.

Recently, a number of approaches to workflow refactoring have been proposed [17], [9], [10], [11], [16], [12]. For example, Wang *et al.* first transform BPEL workflows into automata then employ Petri nets synthesis [18] to obtain a workflow with maximized concurrency [11]. Jin *et al.* propose to leverage workflow mining (i.e., the α-algorithm [19]) for workflow refactoring such that activities without data dependences can be executed in parallel [12]. Though some of these approaches achieve maximized concurrency, it is

- *W. Song and H. Liu are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China, 210094. E-mail: wsong@njust.edu.cn.*
- *H.-A. Jacobsen is with the Middleware Systems Research Group, Technische Universität München, 85748 Garching, Germany, and University of Toronto, Toronto, ONM5S, Canada. E-mail: arno.jacobsen@msrg.org.*
- *S. C. Cheung is with the Department of Computer Science and Engineering, Hongkong University of Science and Technology, Hongkong, China. E-mail: scc@cse.ust.hk.*
- *X. Ma is with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China, 210023. E-mail: xxm@nju.edu.cn.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867593, IEEE Transactions on Services Computing

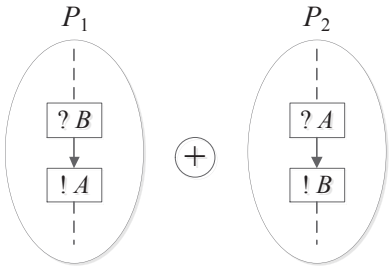


Fig. 1. An illustrating example showing maximized concurrency can improve the success probability of workflow collaboration.

not considered whether the refactored workflow is block-structured. Block-structuredness is a central workflow design principle [4], which means that for every node with multiple outgoing edges (i.e., a *split*), there exists a corresponding node with multiple incoming edges (i.e., a *join*) such that the workflow fragment between the split and the join forms a block [20]. If a workflow model block-structured, data flow analysis (e.g., data races) is made easier. Otherwise, users (e.g., workflow modelers) may have difficulty in understanding, analyzing, and testing its business logic. Furthermore, it is difficult for programmers to transform unstructured workflow models into popular block-structured workflow languages such as BPEL [14], [21]. For an approach which obtains the block-structured control flow (i.e., the approach in [11]), maximized concurrency is not ensured. One may argue that we can maximize concurrency and then block-structuring or reversely by combining these two existing approaches. Unfortunately, this solution is infeasible because the approach in [11] sacrifices concurrency for block-structuredness. In this paper, the control flow of a workflow is regarded as optimal if it ensures maximized concurrency and block-structuredness. To our best knowledge, our work is the first to consider both concurrency and block-structuredness maximization.

To address this bi-objective problem, we propose a novel workflow refactoring approach. For this, we first analyze activity dependences (e.g., control dependences, data dependences [13]) in the original workflow and capture them into a *workflow dependence graph* (WDG) from which we seek the optimal control flow of activities. We define a *well-formed* dependence pattern of activities whose control flow is block-structured with maximized concurrency. Specifically, the control flow can be specified using only sequential and parallel structures (Other structures are also considered but they are irrelevant to concurrency maximization). The WDG of a workflow is not always well-formed, which is caused by a minimum set of extra dependence edges. Should these edges be removed, the resultant graph becomes well-formed. By leveraging BPEL `links` to synchronize different branches (threads) of a parallel structure, the removed edges are transformed into `links` in the final optimized workflow. Those workflow with minimum `links` in the parallel structures are regarded as maximized block-structuredness [21]. Unfortunately, it is intractable to determine the minimum number of edges to transform. Thus, we present a greedy heuristic algorithm which harnesses the predecessors and successors of nodes (activities) to search well-formed patterns in the WDG. Although, our algorithm cannot always guarantee that the number of introduced `links` is minimal, we find that it is efficient in approximating an optimal control flow with minimal `links`.

We realize our approach and evaluate it based on a set of real-world scientific workflows, because they are sufficiently complex dependence graphs to evaluate the effectiveness and efficiency of our approach. The experimental results show that: (1) Our approach obtains BPEL-like workflows with maximized concurrency, i.e., activities without dependences are executed in parallel. (2) The obtained BPEL-like workflows contain few `links`. For the real-world DAGs (scientific workflows) employed in our experiment, the number of introduced `links` ranges from 0 to 8 and is 2.37, on average. (3) Our approach is efficient. The approach only takes 5.0 to 47.4 milliseconds (ms) to refactor scientific workflows with the number of nodes ranging from 7 to 85.

To sum up, this paper extends and improves our previous work [10] and makes the following new contributions:

1) We propose that *unoptimized sequences* (cf. Definition 7) can be regarded as another type of anti-pattern for workflow modelling and design, which are related to the interplay of both control flow and data flow, thus, complementing existing anti-patterns focusing either on control flow or data flow.
2) We define the notion of well-formed patterns in WDGs. The optimal control flow relations of activities in well-formed patterns are block-structured with maximized concurrency, which can be specified only with sequential and parallel structures.
3) Based on well-formed patterns and the semantics of `links`, we present a graph reduction approach, with the proof of its soundness, which guarantees obtaining the workflows with maximized concurrency and few `links` from irregular WDGs.
4) We implement our approach and experimentally evaluate it with a representative set of real-world scientific workflows; results demonstrate the effectiveness and efficiency of our approach.

The remainder of this article is organized as follows. Section 2 introduces some background information necessary for understanding this work. Section 3 formulates the research problem and provides an overview of our approach. Section 4 elaborates on our solution. Section 5 reports our experimental results. Section 6 reviews related work while Section 7 concludes the paper.

2 Preliminaries

Our refactoring approach is designed for BPEL [14] and workflow languages which also use `links` (or similar rules) to synchronize different branches in parallel structures. We use graph models (cf. Definition 1) instead of XML (BPEL) to represent workflows to improve readability. Since our refactoring only focuses on transforming unnecessary sequential structures into parallel structures, there is no need for the model to capture all syntax and semantics of BPEL. More specifically, commonly-used structured activities of BPEL [14], such as `sequence` (sequential structures), `flow`

(parallel structures), `if`, `switch`, `pick` (alternative structures), `while`, `repeatUntil` (iterative structures) and their nested forms can all be expressed by our workflow model (cf. Definition 1). `Links`, inspired by BPEL syntax, are used to synchronize different branches of parallel structures. In BPEL, `links` can be associated with transition conditions, which is not considered in this paper.

Definition 1 (Workflow Model). *A workflow is modeled as a four-tuple $W = (N, E, I, O)$ such that:*

- *$N = \{A_b, A_e\} \cup N_b \cup N_s$ is a set of nodes, where A_b, A_e represent the beginning and the ending of W, N_b is the set of basic activities, and N_s is a set of structured activities.*
- *$E \subseteq N \times N$ is a set of directed edges between nodes, representing the order between activities. An edge is called a link if it is between two nodes in different branches (threads) of a parallel structure, representing synchronization that reduces the parallelism.*
- *$I, O: N \rightarrow 2^D$ are functions assigning input and output parameters to activities (basic activities and decision activities) in N, where D is the set of data variables defined or used in W.*

In our notation for workflows, rounded rectangles, diamonds, and bars represent basic activities, decisions (the start of alternative and iterative structures), and the start (And-split) or the end (And-join) of parallel structures, respectively.

Different activities in a workflow can be inter-related by dependences. There are two common kinds of activity dependences in a workflow: *control dependence* and *data dependence*. Control dependence and data dependence are well-established notions in the field of programming language and software engineering [13], which is also the foundations of our refactoring approach. In the following, we formally introduce them (cf. Definitions 4 and 5) based on our workflow model. To define control dependence, we first introduce the concepts of *path* and *post-dominance*.

Definition 2 (Path). *A path ρ from A to B in a workflow $W = (N, E, I, O)$ is a sequence of one or more edges (V_1, V_2), (V_2, V_3), ..., (V_{n-1}, V_n) in W, where $V_i \in N$ $(1 \leq i \leq n)$, $V_1 = A$, and $V_n = B$.*

Definition 3 (Post-dominance). *In a workflow model W, an activity (node) A_i is post-dominated by another activity A_j if each directed path from A_i to the ending node A_e (not including A_i) contains A_j.*

Definition 4 (Control Dependence). *In a workflow model W, an activity A_j is control-dependent on another activity A_i iff there exists a directed path ρ from A_i to A_j such that any activity A_k in ρ (excluding A_i and A_j) is post-dominated by A_j, and A_i is not post-dominated by A_j.*

Data dependences are three kinds: true dependence, anti-dependence and output dependence [13], [22].

Definition 5 (Data (True) Dependence). *In a path ρ of workflow model W, an activity A_j is true- dependent on another activity A_i iff there is a variable $v \in I(A_j) \cap O(A_i)$, and in ρ, there is no A_k between A_i and A_j such that $v \in O(A_k)$.*

Anti-dependence and output dependence are defined similarly by replacing $v \in I(A_j) \cap O(A_i)$ with $v \in O(A_j) \cap I(A_i)$, and $v \in O(A_j) \cap O(A_i)$, respectively. Besides control dependence and data dependence, in BPEL, there is an *asyn-invocation dependence* between a one-way `invoke` activity and the `receive` activity responsible for receiving the result of the `invoke` [10]. All activity dependences (including implicit dependences in specific applicatons) in a workflow can be captured in its dependence graph.

Definition 6 (Workflow Dependence Graph (WDG) [22], [10]). *A workflow dependence graph of a workflow $W = (N, E, I, O)$ is a directed graph $WDG = (N', E')$, where*

- *$N' \subseteq N$ is a set of nodes representing activities of the workflow.*
- *$E' \subseteq N' \times N'$ is a set of directed edges, and an edge $<A_i, A_j> \in E'$ directed from A_i to A_j denotes an activity dependence (e.g., control dependence, data dependence) between the two activities denoted by A_i and A_j.*

For a control dependence edge $<A_i, A_j>$, if A_i is a decision activity representing an alternative structure (e.g., *if*, *switch*), $<A_i, A_j>$ is labelled either as "T" or as "F" depending on whether A_j occurs when the decision is true. If A_i is a decision activity representing an iterative structure (e.g., *while*, *repeatUntil*), $<A_i, A_j>$ is labelled as "T". If an activity A_j is not control-dependent on any activity, we assume that it is control-dependent on the starting activity A_b (also called "entry") of the workflow. In this way, A_e is always control-dependent on A_b, but we usually omit A_e in the WDG, because it has nothing to do with refactoring.

Fig. 2a illustrates the workflow model of a shipping service whose business logic can be expressed by BPEL: <sequence A_1, A_2, <A_3-while<sequence A_4, A_5, A_6/>/>/>. This workflow sends items in groups until the customer's order is fulfilled. From Fig. 2a, we can see that activities A_4 and A_5 are post-dominated by A_6, while A_3 is not. Thus, according to Definition 4, A_6 is control-dependent on A_3. Similarly, A_4 and A_5 are also control-dependent on A_3. Since activities A_1, A_2, and A_3 are not control-dependent on any activity, they are regarded as to be control-dependent on the beginning activity "Entry". The WDG of the workflow is shown in Fig. 2b where the solid lines labeled "T" denote control dependences, other solid lines data dependences, and the dashed lines loop-carried data dependences [13]. If we discard loop-carried data dependences and those caused by data races, the WDG turns into a DAG. As is discussed in our previous work [23], the discard facilitates the determination of control-flow relations of nodes in the WDG.

3 Problem and Approach Framework

In this section, we first formulate the research problem, and then present an overview of our approach.

3.1 Problem Formulation

To formulate the problem, we first define a novel workflow anti-pattern (cf. Definition 7). Similar to BPEL, we use s = <sequence A_1, A_2, ..., A_n/> to represent a sequence of activities A_1-A_n.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867593, IEEE Transactions on Services Computing

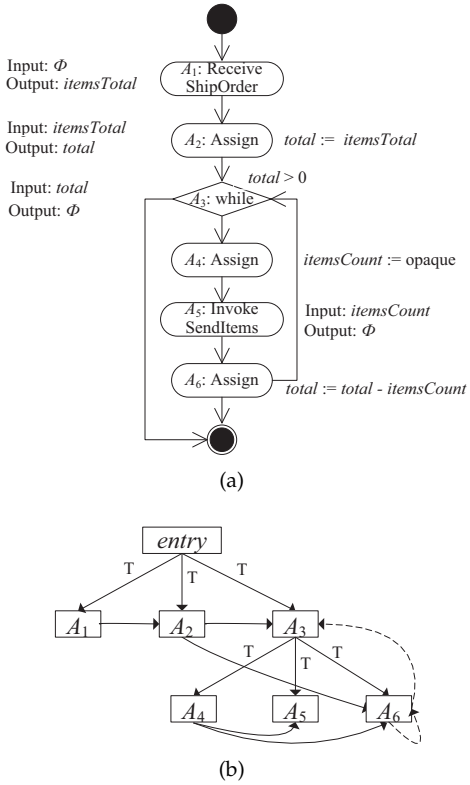


Fig. 2. (a) Workflow and (b) WDG of a shipping service.

Definition 7 (Unoptimized Sequence). *Given a* `sequence` s = <*sequence* A_1, A_2, ..., A_n/> *in a workflow* W, *for* $\forall\, i\ (1 \leq i < n)$, *if* $<A_i, A_{i+1}>$ *is an activity dependence, then* s *is a necessary* `sequence`. *Otherwise,* s *is an unoptimized* `sequence`, *i.e., it is unnecessary that all activities execute in a sequential order.*

The input workflow can have `flow`, but only unoptimized `sequence` (it can be nested in a `flow`) are considered. For example, the workflow depicted in Fig. 2a contains an unoptimized `sequence` s = <sequence A_4, A_5, A_6/> (this `sequence` is nested in an iterative structure *while*), as there is no dependence between activities A_5 and A_6 (cf. Fig. 2b). Informally, our goal is to automatically identify these unoptimized `sequences` and re-arrange those activities to make sure that activities without dependences are executed in parallel while the number of `links` introduced in parallel structures is minimum. More formally, the workflow refactoring problem is described as follows:

The refactoring problem is to optimize the control flow of a workflow W *to obtain a new workflow* W' *which shares the same WDG with* W *such that:*

- W' *involves no unoptimized* `sequences`.
- W' *is as block-structured as possible, i.e., with a minimum number of* `links` *in parallel structures (*`flows`*).*

3.2 Approach in a Nutshell

Fig. 3 illustrates the three stages of our approach:

1) *WDG construction*. Activity dependences (including control and data dependences) of the workflow are analyzed and captured in the WDG.

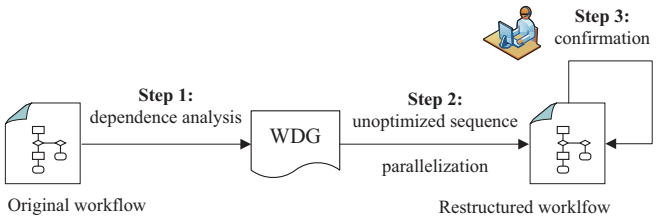


Fig. 3. Three stages of our approach.

2) *Unoptimized* `sequence` *identification & parallelization*. Unoptimized `sequences` are identified via the WDG. Activities with no direct or transitive dependences are arranged in parallel structures.
3) *User confirmation*. Since there may be implicit dependences (the activities need executing in sequence), each parallelization opportunity identified in Stage 2 is provided to users (e.g., workflow modelers and programmers) for final confirmation. An optimization opportunity is abandoned if it is not passed user confirmation.

Although our attention is focused on transforming `sequences` into `flows` (could be with `links`), our approach applies to all workflows that are defined in Definition 1. Since WDG can be obtained with existing approaches [13], [22] and the last stage is straightforward, we focus on the second stage in Section 4.

4 Refactoring Approach

To better understand our approach, we first examine the structure of the WDG. If only edges of control dependences are kept, the WDG degenerates into a tree with A_b ("Entry") as its root [13], [22]. In the tree, except the root, each inner node represents a decision activity, while each leaf node represents a basic activity. For instance, in Fig. 2b, the inner node A_3 denotes a decision activity *while*, while other nodes (except "Entry") denote basic activities.

Because of nested control flow, the refactoring is performed from the inner `sequence` to the outer one. For the workflow in Fig. 2a, the inner <sequence A_4, A_5, A_6/> is refactored first before the outer <sequence A_1, A_2, A_3/>. Next, we first present our approach to *local refactoring*, i.e., seeking the optimal control flow relations for the activities of the innermost block (`sequence`). Then, we show how to identify optimization opportunities across different blocks, referred to as *global refactoring*.

4.1 Local Refactoring

For local refactoring, we first consider a single-control WDG in which there is only one control node (decision activity or A_b). Other nodes in the single-control WDG can be partitioned into one, two or more classes according to the labels (e.g., "T","F") on the edges from the control node to these nodes. Nodes (activities) in different classes are mutually-exclusive, i.e., they cannot be executed together. The refactoring is applied to different classes of nodes independently. Although nodes in the same class can be

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867593, IEEE Transactions on Services Computing

related by edges of data dependences, asyn-invocation dependences, and implicit dependences, there is no need to differentiate them for refactoring.

4.1.1 Optimal Control Flow Graph

In graph theory, a sub-graph SG of $G = (N, E)$ is said to be *induced* by a node set SN ($SN \subset N$) if SG has all edges of G with both endpoints in SN. Now, we discuss how to seek the optimal control flow relations for the nodes in the same class based on the sub-graph (DAG) they induce.

A weakly connected component [24] is a maximal sub-graph of a directed graph such that if replacing all of its directed edges with undirected edges produces a connected (undirected) graph, that is, any two nodes, say, A_i and A_j, are reachable from each other. Only two cases exist: the sub-graph is a weakly connected component (WCC), or it involves more than one WCC. For the latter case, the control flow relation among these WCCs is specified as `flow`, as they are independent of one another. To determine the control flow relations of the nodes inside a WCC, we need to ensure that the determined control flow relations of the nodes are consistent with the dependences in the WCC such that concurrency degree can be maximized. This problem can be regarded as an extension to the problem of *topological sorting* [25], because all possible topological sorts correspond to the trace set of the obtained control flow graph (cf. Theoerem 3). Here, our goal is to obtain the optimal control flow graph (cf. Definition 8) from the WCC.

Definition 8 (Optimal Control Flow Graph). *Given a DAG $G = (N, E)$, the corresponding control flow graph G_{CF} derived from G is optimal if the following two conditions are satisfied:*

- *For any two nodes A_p and A_q, if $A_q(A_p)$ is reachable from $A_p(A_q)$ in G, the control flow between them in G_{CF} is `sequence`, and $A_p(A_q)$ precedes $A_q(A_p)$); otherwise, the control flow between them in G_{CF} is `flow`.*
- *G_{CF} is as block-structured as possible, i.e., minimum number of `links` is involved.*

The first condition of Definition 8 ensures concurrency is maximized, viz., no unoptimized `sequences` are involved. The second condition ensures maximized block-structuredness. Although our refactoring approach aims at this bi-objective optimization, as we will show in Section 4.1.3, it is difficult to guarantee the second condition. Fortunately, if the first condition is satisfied, the corresponding workflow shares the equivalent behavior with the refactored workflow satisfying both conditions [26], because they are trace-equivalent [27][1].

4.1.2 Graph Reduction

It is challenging to obtain the optimal control flow graphs because of the complex structure of WCCs. Here, we present a graph reduction approach to address this problem, which includes the following three steps:

1) *Obtain the transitive reduction of the WCC*. The transitive reduction[2] [28] preserves the reachability relations in the WCC, but has fewer dependence edges. This facilitates seeking the optimal control flow relations of nodes in the WCC. In the remainder of this paper, all mention of WCC refers to its transitive reductions.
2) *Search well-formed patterns (cf. Definition 10) for reduction*. In the WCC, control flow relations of well-formed patterns are block-structured and can be described with `sequence` and `flow`. After the control flow of a well-formed pattern is determined, we use a single node to replace the whole pattern in the WCC (cf. Reduction Rule 1) to facilitate identifying other well-formed patterns.
3) *Leverage the semantics of `links` for further reduction*. If the WCC is not reduced to a single node (e.g., the example in Fig. 6 in Section 4.1.3), the complete block-structuredness is impossible. In this case, we can remove minimum number of extra edges in order to obtain well-formed patterns whose control flow relations can be determined as the second step. After this, a `link` is introduced for each removed edge such that the two endpoints of the `link` are the same as those of the removed edge.

In the following, we explain Step 2 and Step 3. For convenience of describing the notion of well-formed pattern and the corresponding reduction rule, we first introduce the concepts of *preset* and *postset*.

Definition 9 (Preset, Postset). *In a directed graph $G = (N, E)$, for any node $A_p \in N$, the preset of A_p is defined as ${}^\bullet A_p = \{A_q | A_q \in N \wedge <A_q, A_p> \in E\}$ and the postset of A_p is defined as $A_p{}^\bullet = \{A_q | A_q \in N \wedge <A_p, A_q> \in E\}$. For a node set SN, the preset of SN is defined as ${}^\bullet SN = \{A_q | A_p \in SN \wedge A_q \in N \backslash SN \wedge <A_q, A_p> \in E\}$ and the postset of SN is defined as $SN^\bullet = \{A_q | A_p \in SN \wedge A_q \in N \backslash SN \wedge <A_p, A_q> \in E\}$.*

Definition 10 (Well-formed Pattern). *In a WCC $G = (N, E)$, a sub-graph SG induced by a node set $SN = SN_1 \cup SN_2 \cdots \cup SN_n$ is regarded to be a well-formed pattern if for any A_p, $A_q \in SN_i$ ($1 \leq i \leq n$), the following two conditions are satisfied:*

- *$<A_p, A_q> \notin E \wedge <A_q, A_p> \notin E$.*
- *${}^\bullet A_p = {}^\bullet A_q = SN_{i-1} \wedge A_p{}^\bullet = A_q{}^\bullet = SN_{i+1}$, where $SN_0, SN_{n+1} \subset N$, $SN_0 \cap SN = SN_{n+1} \cap SN = \emptyset$.*

Note that in Definition 10, SN_0 and SN_{n+1} are not subsets of SN, which indicates that the well-formed pattern is maximized already, which cannot be further enlarged.

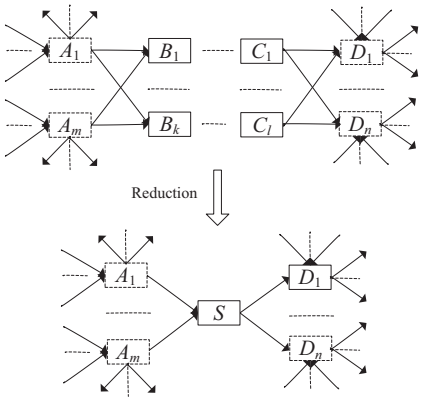
Reduction Rule 1. *In a WCC $G = (N, E)$, a well-formed pattern SG induced by a node set $SN = SN_1 \cup SN_2 \cdots \cup SN_n$ can be reduced into a single node S such that:*

- *${}^\bullet S = {}^\bullet SN_1 \wedge S^\bullet = SN_n{}^\bullet$.*
- *S = <sequence <flow SN_1 />, $\cdots$, <flow SN_n /> />, denoting that nodes in SN_i ($1 \leq i \leq n$) are in a parallel structure, and SN_1, ..., SN_n are in a sequential structure.*

1. A trace of G_{CF} is a linear extension of the *happens-before* relations in G_{CF}.

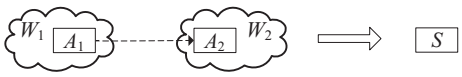
2. The transitive reduction of a directed graph D is another directed graph D' with the same vertices and as few edges as possible, such that if there is a (directed) path from vertex s to vertex t in D, then there is also such a path in the reduction D'.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867593, IEEE Transactions on Services Computing



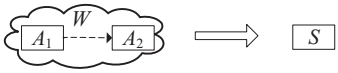
S = <sequence <flow B_1, ⋯, B_k />, ⋯ <flow C_1, ⋯, C_l /> />

Fig. 4. Reduction Rule 1.



S = <flow W_1, W_2 <link A_1 to A_2 /> />

(a)



S = <flow W <link A_1 to A_2 /> />

(b)

Fig. 5. Two possible scenarios to introduce `links`.

Fig. 4 illustrates Reduction Rule 1, where node sets SN_1, ⋯, SN_n = $\{B_1, \cdots, B_k\}$ ⋯, $\{C_1, \cdots, C_l\}$ and SN_0 = $\{A_1, \cdots, A_m\}$, SN_{n+1} = $\{D_1, \cdots, D_n\}$. We can see that the reduction from several WCCs to a `flow` node is a special case of Reduction Rule 1.

We can iteratively utilize this reduction rule to reduce the WCC until it becomes a single node or no well-formed patterns can be found. If it is the former case, we can obtain the optimal control flow relations of activities in the WCC from the single node. Otherwise, we need to combine Reduction Rule 1 with the semantics of `links` to seek the optimal control flow relations for not well-formed patterns. To this end, some edges in the resultant WCC need to be removed such that the WCC can become well-formed. Following the removal of some edges, the control flow relation obtained is more relaxed than what it should be. To reflect the restriction for removed edges on the control flow, for each removed edge, we introduce a `link` whose `source` and `target` are the same as those of the removed edge. Fig. 5a and Fig. 5b illustrate two possible scenarios to introduce `links`. In the former scenario, the `source` and the `target` of the introduced `link` are in different WCCs, whereas in the latter scenario, they are in the same WCC.

According to Definition 1 and the BPEL specification [14], `links` can only be contained in `flows`. Theorem 1 guarantees that our solution abides by this requirement.

Theorem 1. *The source activity and the target activity of an introduced `link` are in a parallel structure (i.e., a `flow`).*

Proof. We prove Theorem 1 by contradiction. According to our approach, a `link` in the resultant workflow corresponds to a removed edge in the WCC (dependence graph). Let $<A_p, A_q>$ be such a removed edge. Assume that activities A_p and A_q are not in a `flow`, and, thus, they must be in a `sequence`. According to Reduction Rule 1, there must exist a (transitive dependence) path from A_p to A_q in the WCC, say $\rho = <A_p, A_x> \cdots <A_y, A_q>$. However, the co-existence of ρ and $<A_p, A_q>$ contradicts the fact that WCC is a transitive reduction. Thus, Theorem 1 holds. □

Theorem 2. *Our graph reduction approach can obtain a control flow graph with maximized concurrency from a DAG.*

Proof. Let A_p and A_q be any two nodes in a DAG. If A_q is reachable from A_p in the DAG, or reversely, in the obtained control flow graph, they are either in a `sequence` according to Reduction Rule 1, or linked in a `flow` according to the semantics of `links`. In both situations, A_p and A_q is executed in sequence. If A_q and A_p are not reachable from each other in the DAG, they are either in the same WCC or in different WCCs. In either case, our approach ensures that they are executed in parallel. Thus, the first condition of Definition 8 is met, and thus Theorem 2 holds. □

The overall behavior of a workflow can be expressed by its set of complete traces (from the beginning to the end of the workflow). If at least one parallelization opportunity is confirmed by users, the trace set of the refactored workflow subsumes that of the original workflow. Theorem 3 demonstrates the relation between the trace set of the control flow graph with maximized concurrency and the set of topological sorts of the DAG (dependence graph).

Theorem 3. *A control flow graph derived from a DAG satisfies maximized concurrency if and only if its complete trace set equals the set of all possible topological sorts of the DAG.*

Proof. Assume that G_{CF} is a control flow graph derived from the DAG. Let S_1 be the set of all possible topological sorts of the DAG and S_2 the set of all complete traces of G_{CF}. We proceed by proving sufficiency and necessity.

Sufficiency. If $S_1 = S_2$, we show G_{CF} meets maximized concurrency (cf. Definition 8). For any two nodes A_p and A_q in the DAG,

1) If A_q (A_p) is reachable from A_p (A_q) in the DAG, for any topological sort σ in S_1, A_p (A_q) precedes A_q (A_p). Since $S_2 = S_1$, the control flow relation between A_p and A_q in G_{CF} is *sequence*.
2) Otherwise, there must exist at least two topological sorts σ_1 and σ_2 in S_1 such that A_p precedes A_q in σ_1 and A_q precedes A_p in σ_2. Since $S_2 = S_1$, the control flow relation between A_p and A_q in G_{CF} is `flow`.

Necessity. We prove $S_1 \subseteq S_2$ and $S_2 \subseteq S_1$.

1) Let σ be any topological sort in S_1, and A_i and A_{i+1} are any two adjacent activities in σ. This implies either $<A_i, A_{i+1}>$ is an edge of the DAG, or A_i and A_{i+1} are not reachable from each other. Since G_{CF} meets maximized concurrency, in either case, there must be a complete trace σ' of G_{CF} such that A_i and A_{i+1} are adjacent (A_i directly precedes A_{i+1}) in σ'. Owing to the arbitrariness of A_i and A_{i+1}, the order

between activities in σ is preserved in σ', i.e., $\sigma = \sigma'$. Hence, σ is a complete trace of G_{CF}, i.e., $S_1 \subseteq S_2$.

2) Let σ be an arbitrary complete trace in S_2, and A_i and A_{i+1} are any two adjacent activities in σ. In G_{CF}, the control flow relation between A_i and A_{i+1} is either *sequence* (A_i directly precedes A_{i+1}) or *flow*. Since G_{CF} meets maximized concurrency, for the former case, $<A_i, A_{i+1}>$ must be an edge in the DAG; for the latter case, neither A_i nor A_{i+1} can be reached from the other. In either case, there must exist a topological sort σ' of the DAG such that A_i and A_{i+1} are adjacent (A_i directly precedes A_{i+1}) in σ'. Owing to the arbitrariness of A_i and A_{i+1}, the order between activities in σ is preserved in σ', i.e., $\sigma = \sigma'$. Hence, σ is a topological sort of the DAG, i.e., $S_2 \subseteq S_1$.

To sum up, Theorem 3 is proven. □

If there are optimization opportunities (i.e., unoptimized `sequences`) in the original workflow, the restructured workflow derived from the DAG will involve more traces than the original workflow; that is, the trace set of the original workflow is a subset of that of the restructured workflow. This is of great significance for service (workflow) collaboration, because more behaviour (traces) implies more compatible partners in workflow composition [10].

4.1.3 Strategy for Introducing Links

Reduction Rule 1 and the semantics of `links` are complementary. On the one hand, if we incorporate all activities of a WCC into a `flow` structure and introduce a `link` for each edge in the WCC, maximized concurrency is achieved. Nonetheless, the obtained control flow graph is far from block-structured. On the other hand, Reduction Rule 1 can obtain the optimal control flow graph without using `links` provided that the WCC is well-formed. Due to the irregularity of WCC, we have to combine these two solutions together, and the goal is to introduce a minimum number of `links`. The well-formed pattern (cf. Definition 10) is called *series-parallel directed graph* (or *minimal vertex series-parallel directed graphs* in particular) in graph theory [29]. Thus, our goal equals to obtain the maximum series-parallel subgraph from a DAG. It is known that this problem is NP-hard for undirected graphs [30], [31], but whether the result is generalized to directed graphs is still an open problem [29].

One may come up with the following straightforward algorithm to introduce a minimum number of `links` for graph reduction. Given an arbitrary DAG $G = (N, E)$, its transitive reduction $G' = (N, E')$ can be obtained. First, the algorithm checks whether G' is a well-formed pattern. If yes, it terminates. Otherwise, it precedes by removing any one edge from G', and determines whether the resultant G' is well-formed. In the second step, there could be C_n^1 tries, where $n = |E'|$. For each of the C_n^1 attempts, the algorithm terminates when G' becomes well-formed. Otherwise, it goes on by removing any two edges from G', and determines whether G' is well-formed. In the third step, there could be C_n^2 attempts. The algorithm iterates in this way until G' becomes well-formed. When it terminates, the number of edges removed must be minimum, and thus a minimum number of `links` for graph reduction is guaranteed. However, the number of attempts in the worst case is $C_n^0 + C_n^1 + \cdots + C_n^n = 2^n$, which is intractable in practice.

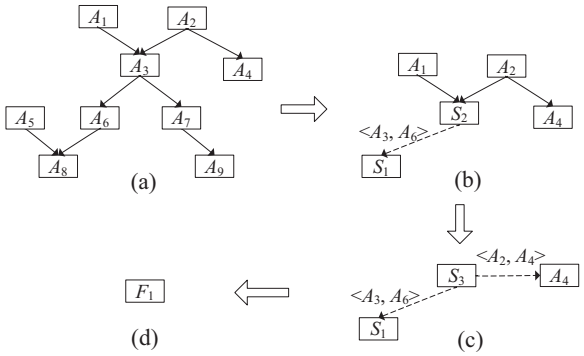
Algorithm 1 Local refactoring

Input: The DAG-based dependence graph (WCC) $G = (N, E)$
Output: The obtained control flow graph w.r.t the WCC

1: Obtain the preset ${}^\bullet A$ and postset $A^\bullet$ for each node A in the WCC;
2: Find nodes whose presets and postsets are all the same. These nodes can be combined and reduced to a single node (i.e., <flow/>). Find a sequence of nodes $A_1 A_2 \cdots A_n$ such that ${A_i}^\bullet = \{A_{i+1}\}$, ${}^\bullet A_{i+1} = \{A_i\}$, where $1 \leq i \leq n$-1. These nodes can be combined together and reduced to a single node (i.e., <sequence $A_1, A_2, \cdots, A_n$/>;
3: Find nodes whose presets (postsets) intersect. For each set (denoted as P) of these nodes, obtain the difference set (denoted as D) of the presets (postsets) of these nodes. If edges with one endpoint in P and the other in D are removed, a well-formed pattern can be found, which is regarded as a candidate for reduction;
4: Use a greedy heuristic for the graph reduction. That is, utilize Reduction Rule 1 to reduce the well-formed pattern (including the nodes in P) whose profit n/m is the largest, where m is the number of edges removed and n is the number of nodes that can be composed for reduction;
5: The locally optimal steps, i.e., steps 3 and 4, are iterated until the WCC is reduced into a single node.

With the above reasons, we propose a greedy-heuristic algorithm (Algorithm 1) to seek the optimal control flow graph. Algorithm 1 leverages the presets and postsets of nodes to find well-formed patterns. Note that we can optimize Step 3 in Algorithm 1 as follows. First, we identify activities whose presets or postsets are the same. Based on these sets of nodes, we can also obtain corresponding candidate subgraphs for the local optimization. If no activities meet this condition, then we use the operations in Step 3 to go on. The original operations in Step 3 are referred to strategy 1, and the optimization is referred to strategy 2. No matter which strategy is used in Step 3, our greedy heuristic does not guarantee to find the optimal solution (i.e., the minimal number of `links`), but rather yields solutions approximating the optimum in reasonable time. Strategy 2 does not guarantee introducing fewer `links` than Strategy 1 does, but it could be more efficient because it may generate fewer candidate subgraphs for the local decision. This will be validated in Section 5. The control flow relations obtained are always with maximized concurrency even if more than the minimum number of `links` are introduced.

Let us illustrate Algorithm 1 with an example. Fig. 6a shows a WCC representing a dependence graph. Our goal is to seek the optimal control flow relations of nodes (activities) in this WCC. Since the WCC is not well-formed, we need to remove some edges to obtain well-formed subgraphs whose control flow relations can be determined. According to Step 3 of Algorithm 1, we should find nodes whose presets or postsets intersect. For example, we can find a node set $\{A_5, A_6\}$, where nodes A_5 and A_6 share the same postset, i.e., ${A_5}^\bullet = {A_6}^\bullet = \{A_8\}$. Similarly, we can find all other candidate node sets: $\{A_1, A_2\}$, $\{A_3, A_4\}$, $\{A_6, A_7\}$. If node set $\{A_5, A_6\}$ is selected, the edge $<A_3, A_6>$ can be selected for removal because one of its endpoints is in $\{A_5,$



S_1=<sequence <flow A_5, A_6 />, A_8/>
S_2=<sequence A_3, A_7, A_9 />
S_3=<sequence <flow A_1, A_2 />, S_2 />
F_1= <flow S_1, S_3, A_4, <link A_2 to A_4 />, <link A_3 to A_6 /> />

Fig. 6. An example illustrating our greedy algorithm.

$A_6\}$ and the other endpoint is in ${}^\bullet A_6 \setminus {}^\bullet A_5$. If $<A_3, A_6>$ is removed, subgraphs induced by node sets $\{A_5, A_6, A_8\}$ and $\{A_3, A_7, A_9\}$ are both well-formed and the profit (i.e., (3+3)/1=6) is the largest. Therefore, we can utilize Reduction Rule 1 to reduce both subgraphs into two single nodes, S_1 and S_2 (cf. Fig. 6b). In Fig. 6b, the subgraph induced by node set $\{A_1, A_2, A_4, S_2\}$ is not well-formed. For this WCC, there are two candidate node sets $\{A_1, A_2\}$ and $\{S_2, A_4\}$. Either node set can be selected because the two node sets correspond to the same profit, i.e., (3+1)/1=4. We use a random selection in case of profit parity. If $\{A_1, A_2\}$ is selected, edge $<A_2, A_4>$ needs to be removed. Then, the subgraph induced by node set $\{A_1, A_2, S_2\}$ can be reduced to a single node S_3 (cf. Fig. 6c). As shown in Fig. 6c, since nodes S_1, S_3, A_4 are not connected, they can be incorporated into a *flow* activity F_1, and the two removed edges $<A_3, A_6>$ and $<A_2, A_4>$ become `links` in F_1 (cf. Fig. 6d). From the final node F_1, we can obtain the near-optimal control flow relations of activities A_1-A_9.

Finally, we analyze the time complexity of Algorithm 1. Before Algorithm 1 is applied, the transitive reduction of the dependence graph (DAG) is obtained, whose time cost is $O(|N|^3)$. For Algorithm 1, we intially obtain the preset and postset of each node in the dependence graph $G = (N, E)$. The time cost for this step is $O(|N|+|E|)$. Second, we search in G for the well-formed patterns by finding nodes whose preset and postset are the same. Once such a pattern is found, we reduce the pattern to a single node in the graph. The second step iterates until no such pattern can be found. The time complexity for this step is $O(|N|^2)$. Third, based on the presets and postsets of the remaining nodes, we utilize a greedy heuristic for selecting some edges to be removed in order to find well-formed patterns. The time complexity for the third step is $O(|E|\times|N|^2)$. Therefore, the total time complexity of Algorithm 1 is $O(|N|^2\times(|E|+|N|))$.

4.2 Global Refactoring

We extend our approach to restructure workflows whose WDGs have more than one control node (a.k.a. multi-controlled WDGs). Due to the nested structure of work-

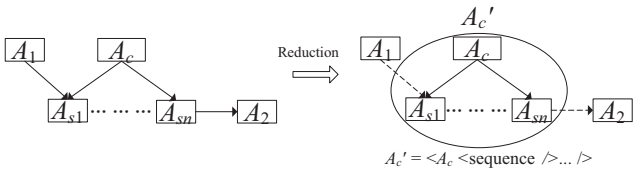


Fig. 7. Reduction Rule 2.

flows, global refactoring needs to identify optimization opportunities from the innermost `sequence` to the outermost one, as specified by Algorithm 2.

Algorithm 2 Global refactoring

Input: The WDG of the original workflow
Output: The restructured workflow
1: All control nodes (decision nodes and A_b) of the resultant WDG are identified and pushed into a stack in the level traversal order beginning from the root (A_b) of the sub-graph (only control dependence is considered) of the WDG;
2: A control node A_c is popped from the stack. From A_c, corresponding sub-workflow whose WDG is single-controlled is obtained. Thus, the sub-workflow can be refactored by Algorithm 1;
3: The corresponding single-control WDG is degenerated into its control node A_c (cf. Reduction Rule 2)
4: Steps 3 and 4 are iterated until the control node stack becomes empty.

Reduction Rule 2. *In a WDG $G = (N, E)$, $G_s = (N_s, E_s)$ is a single-control WDG induced by node set N_s. G_s degenerates to its control node A_c (also denoted as A_c') while G reduces to $G' = (N', E')$ such that:* $N' = N\setminus(N_s\setminus\{A_c\})$; $E' = E\setminus E_s\setminus\{<A,A_s>|A\in N\setminus N_s\wedge A_s\in N_s\setminus\{A_c\}\wedge<A,A_s>\in E\}\setminus\{<A_s,A>|A\in N\setminus N_s\wedge A_s\in N_s\setminus\{A_c\}\wedge<A_s,A>\in E\}$.

Fig. 7 illustrates Reduction Rule 2: The single-control WDG G_s degenerates to its control node A_c (A_c'), which facilitates the subsequent refactoring. Moreover, any edge with one endpoint in $N_s\setminus\{A_c\}$ and the other endpoint in $N\setminus N_s$ is "removed" (denoted by dashed directed edges). Such a "removed" edge will become into a *link* when its two endpoints are enclosed in a `flow` structure (cf. Theorem 1).

Our two reduction rules exhibit different functionalities. Reduction Rule 1 determines the block-structured control flow (`sequence`, `flow`) of activities in the dependence graph, whereas Reduction Rule 2 simplifies the WDG such that the problem of global refactoring is reduced to the one of local refactoring. We assume that the original BPEL-like workflows are block-structured and hence sound [32]. Since the `links` introduced cannot lead to errors, such as deadlocks, the soundness of the workflow is not compromised.

To refactor the process in Fig. 2a, we first traverse the sub-graph (tree in terms of control dependences) of the WDG in Fig. 2b following a level traversal order. After this step, the "Entry" node and the decision node A_3 are pushed into a stack. Secondly, A_3 is popped out from the stack, and Algorithm 1 is used to refactor <sequence A_4, A_5, A_6/> into <sequence A_4, <flow A_5, A_6/>/>. After this, the sub-graph controlled by A_3 is degenerated into a single node A_3' according to Reduction Rule 2. Next, the "Entry" node is popped out from the stack. However, <sequence A_1, A_2,

$A_3/>$ cannot be optimized any more. The final refactored workflow is <sequence A_1, A_2, <A_3-while<sequence A_4, <flow A_5, $A_6/>/>/>/>$, which is block-structured with maximized concurrency.

Similar to Theorem 2, we established that the restructured workflows obtained by our global refactoring algorithm are also with maximized concurrency. In addition, our greedy algorithm can approach to introducing minimum number of `links`. The time complexity of Algorithm 2 is the same as that of Algorithm 1, which is $O(|N|^2 \times (|E|+|N|))$.

5 Evaluation

In this section, we conduct an experimental evaluation to answer the following two research questions:

- **RQ1** - *Effectiveness*: How effective is our approach in maximizing concurrency and block-structuredness for workflow refactoring? That is, on the one hand, does it neither miss real optimization opportunities, nor introduce false ones? On the other hand, does it introduce few `links` into the refactored workflows?
- **RQ2** - *Efficiency*: How efficient is our refactoring approach (and other approaches) in maximizing concurrency and block-structuredness? Does our approach scale well in practice?

Our experiment was performed on a set of real-world scientific workflows by using a computer with 1.7 GHz CPU and 4 GB memory, running windows 8 and JDK 1.7.

5.1 Experimental Setup

Approach Implementation. In the experiment, we compare our approach with state-of-the-art refactoring approaches summarized in Table 1. We implement our approach in a prototype ProR, which can be found at: http://bit.ly/myProR. These three approaches do not require duplication of tasks. The approach BeehiveZ is based on a workflow mining technique, that is, the α-algorithm [19]. Note that the three approaches are quite different in the following respects. First, the workflow models used in these approaches are heterogeneous. More specifically, CASS uses directed graphs, BeehiveZ employs Petri nets, and the workflow model adopted by our approach is similar to UML activity diagrams [33]. Second, decision activities are only explicitly modeled in some of the aforesaid models. Thirdly, these approaches utilize distinct analysis techniques to obtain the activity dependences. Despite these differences, the last and the vital step in all these three approaches is similar: they all leverage the workflow dependence graph (or activity dependences) to obtain the refactored workflows. Hence, we only focus on the last step of each approach to obtain a fair comparison.

Data Set. Since the performance of our approach significantly depends on the complexity of the dependence graph, we use real-world scientific (Taverna) workflows from myExperiment[3] [34] for our evaluation. These scientific workflows tend to be more complex (not so strictly) with the increasing number of activities, which can discriminate different approaches. According to the number of activities (#Act) involved, we divide the workflows into three categories: simple (#Act $<$ 30), medium (30 $\leq$ #Act $<$ 50), and complex (50 $\leq$ #Act). We randomly select 10 subjects from each category and thus 30 scientific workflows are used. The 30 workflows are modeled with DAGs and each edge of the DAGs represents the dependence between two activities. Similar workflows are also used in [2], [35]. Based on the dependence relations in these DAGs, different refactoring approaches are utilized to seek the workflow models with the optimal control flow. This equals to transform the DAGs into BPEL-like workflows. The names (few are shortened for short), the number of activities (#Act), and the number of edges (#Edg) of the 30 workflows with references ("1"-"30") are summarized in Table 2. To facilitate comparison, we introduce two notions. Given a scientific workflow W, one, we utilize an ordered activity pair $<A_p, A_q>$ to show that activity A_q is reachable from activity A_p in W, and, two, we use an unordered activity pair $[A_p, A_q]$ to show that activities A_p and A_q are not reachable from each other in W. $<A_p, A_q>$ and $[A_p, A_q]$ are referred to as *reachability pair* and *unreachability pair*, respectively.

TABLE 1
Approaches Compared in the Experiments

Ref.	Refactoring approaches
CASS	The approach proposed in [9]
BeehiveZ	The approach proposed in [12]
ProR	Our graph reduction-based approach

Evaluation Criteria. Although workflow models (control flow graphs) obtained by different approaches are in different forms (directed graphs, Petri nets, UML activity diagrams), we can define general criteria to evaluate the obtained process models. Assume that W' is the workflow model obtained from a DAG by using an approach from Table 1. We use an ordered pair $<A_p, A_q>$ to show that activity A_p precedes activity A_q in W', and an unordered activity pair $[A_p, A_q]$ to show that activities A_p and A_q are concurrently executed in W'. Let S and F be the sets of ordered pairs and unordered pairs in W', and let R and U be the sets of reachability pairs and unreachability pairs in the DAG, respectively. The following criteria are used to evaluate different workflow refactoring approaches.

Criterion 1: *Checking Concurrency Maximization*. We adapt F-measure (i.e., F_1) of precision and recall [36] to measure to what extent a refactoring approach can achieve maximized concurrency, given by:

$$precision = (\mid S \cap R \mid + \mid F \cap U \mid)/ \mid S \cup F \mid \quad (1)$$

$$recall = (\mid S \cap R \mid + \mid F \cap U \mid)/ \mid R \cup U \mid \quad (2)$$

$$F_1 = 2 \times precision \times recall/(precision + recall) \quad (3)$$

The higher the value of the F-measure (F_1), the better the corresponding workflow refactoring approach. F_1 = 1 implies that the obtained workflow model is with maximized concurrency. $F_1 < 1$ indicates that the approach fails to obtain maximized concurrency.

3. http://www.myexperiment.org/workflows

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867593, IEEE Transactions on Services Computing

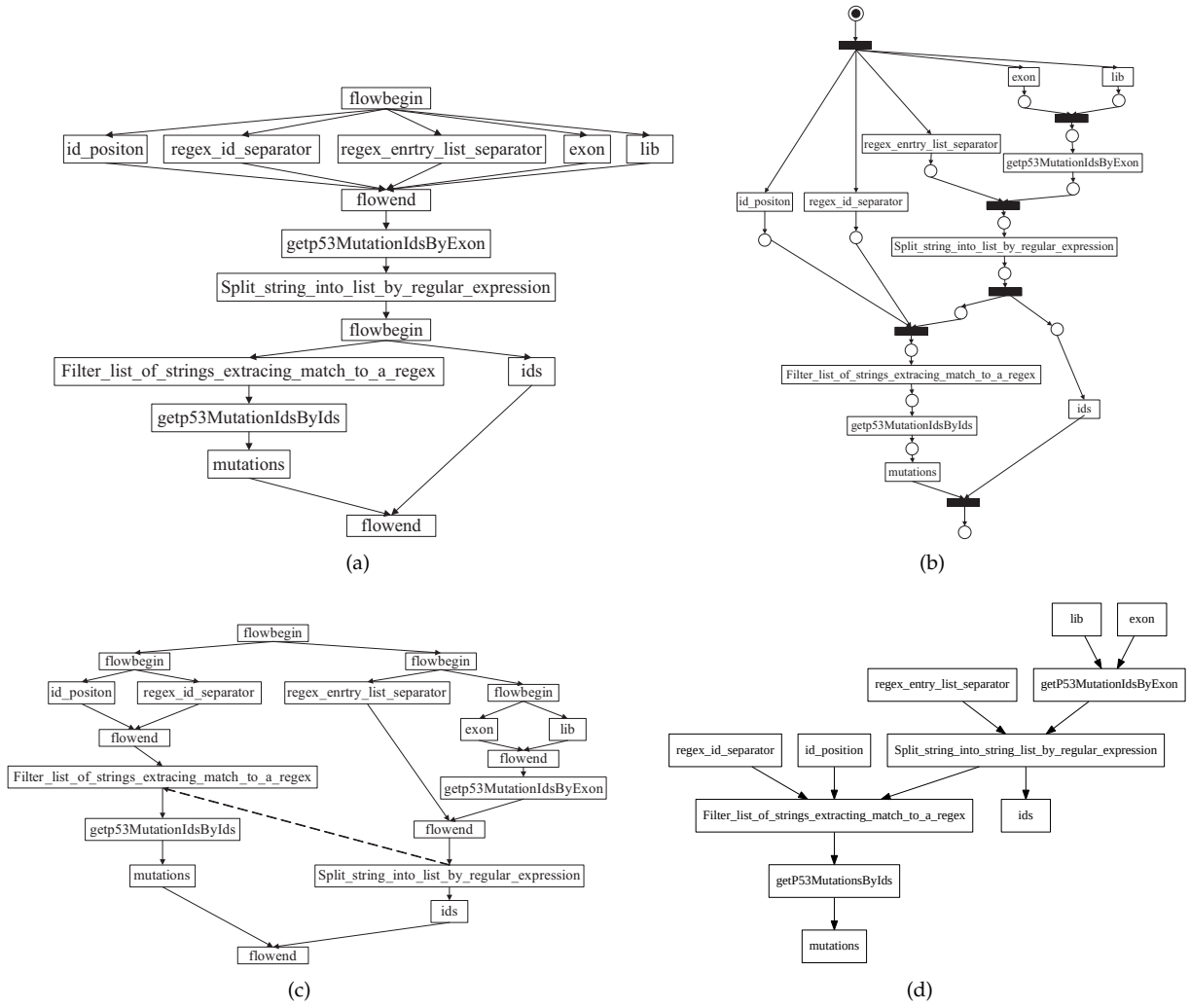


Fig. 8. Workflow models (control flow graphs) (a), (b), (c) transformed from (d) a real scientific workflow "Get TP53 By Exon" with CASS, BeehiveZ, and ProR, respectively.

Criterion 2: *Checking Block-structuredness*. From the DAGs, the obtained workflow models can only contain sequential structures and parallel structures. Hence, the obtained workflow model is block-structured if for each node representing the start (And-split) of a parallel structure, there is a corresponding node representing the end (And-join) of the parallel structure such that the workflow fragment between them forms a block. Hence, the necessary condition for the block-structuredness is that And-splits and And-joins are in pairs. Following this, if the number of And-splits is not equal to the number of And-joins, the obtained workflow model is not block-structured, because it is difficult to transform the model into BPEL. When a DAG cannot be transformed into a block-structured workflow, And-splits and And-joins also need to be in pairs such that a BPEL-like workflow with `links` can be obtained. In this case, the number of introduced `links` must be as small as possible. Unfortunately, the ground truth of the minimal `links` is unavailable.

Criterion 3: *Efficiency*. The runtime overhead (i.e., the average execution time) of the different refactoring approaches are also recorded to investigate the efficiency and scalability.

5.2 RQ1: Effectiveness

Before presenting the experimental results, we first use an example to show the advantage of our approach. Fig. 8a-c shows the workflow models (control flow graphs) transformed from the DAG (scientific workflow with reference "3" in our data set[4]) in Fig. 8d by approaches CASS, BeehiveZ, and ProR, respectively. It follows that: the workflow model in Fig. 8a is block-structured, but the maximized concurrency is not satisfied, because some activities independent in Fig. 8a are executed in sequence; the workflow model in Fig. 8b (Petri net) satisfies maximized concurrency but is not block-structured, because some And-split (flowstar) or And-join (flowend) are missing; the workflow model in Fig. 8c satisfies maximized concurrency and is as block-structured as possible, that is, each `flow` begins with an And-split and ends with an And-join, while only one `link` from activity *Split_string...* to activity *Filter_list...* is introduced. Note that the And-splits and And-joins in Fig. 8c are used for illustration, and they correspond to <flow> and </flow>, respectively, in the final BPEL-like file. Our

4. http://www.myexperiment.org/workflows/1013.html

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2867593, IEEE Transactions on Services Computing

TABLE 2
Experimental Results - RQ1: Effectiveness

Ref.	Name	#Act	#Edg	CASS			BeehiveZ			ProR			
				F_1	#split	#join	F_1	#split	#join	F_1	#split	#join	*#links*
1	Fetch today's xkcd comic	7	7	0.9	1	1	1	0	1	1	1	1	0
2	compare pubmed results ge	10	13	0.96	1	1	1	0	2	1	2	2	0
3	Get TP53 By Exon	11	10	0.87	2	2	1	1	3	1	5	5	1
4	Chemical2URIs (Version 1)	13	12	1	1	1	1	1	1	1	1	1	0
5	Simulate SBML ODEs	16	18	0.72	3	3	1	2	3	1	7	7	1
6	TTPish workflow for MASSyPup64	18	24	0.59	6	6	1	5	1	1	7	7	1
7	Online PubMed author search	21	27	0.99	1	1	1	0	2	1	2	2	0
8	EBI InterProScan for Taverna	24	25	0.96	2	2	1	3	5	1	1	1	0
9	NCBI Gi to Kegg Pathways	27	27	0.63	5	5	1	4	3	1	9	9	4
10	FLOSS Communication	28	33	0.89	5	5	1	4	5	1	9	9	2
11	GIPSY's galmod with VO	33	32	0.82	3	3	1	2	5	1	8	8	1
12	Author Citation Network	35	44	0.95	3	3	1	4	6	1	7	7	0
13	Mapping microarray data	37	46	0.73	6	6	1	5	8	1	9	9	2
14	Gene annotation pipeline	38	40	0.43	4	4	1	8	1	1	11	11	1
15	Download pathways	41	45	0.66	4	4	1	9	18	1	11	11	5
16	Identify strain and phylogenetic tree	43	39	0.64	3	3	1	12	8	1	10	10	2
17	Terms from collection of PDF files	44	43	0.71	3	3	1	10	13	1	8	8	1
18	ConVergenceTreeDiagnosticGeoKS	44	46	0.63	3	3	1	6	8	1	11	11	2
19	Entrez Gene to KEGG Pathway	47	50	0.79	7	7	1	9	5	1	8	8	4
20	casimir paper	49	53	0.54	4	4	1	3	1	1	4	4	0
21	retrive dpas data for all ics	52	55	0.61	4	4	1	4	8	1	18	18	5
22	Calculation of a rotation curve	54	59	0.66	5	5	1	19	15	1	15	15	5
23	Human Microarray Analysis	57	64	0.69	4	4	1	33	30	1	8	8	1
24	Lymphoma type prediction	58	76	0.79	5	5	1	19	7	1	8	8	4
25	BioAID ProteinDiscovery	59	61	0.74	5	5	1	21	31	1	18	18	5
26	Pathways and Gene annotations	61	65	0.7	8	8	1	12	7	1	11	11	4
27	Cow-Human Ortholog	62	66	0.65	6	6	1	40	36	1	11	11	3
28	Kinematical modelling of a galaxy	70	82	0.6	6	6	1	38	43	1	21	21	7
29	Nucleotide InterProScan	75	82	0.69	9	9	1	28	28	1	22	22	8
30	DataBiNS with Kegg ID	85	84	0.45	7	7	1	32	20	1	18	18	2

approach guarantees And-splits and And-joins are in pairs (cf. Section 4).

The experimental results on all the 30 scientific workflows based on Criteria 1 and 2 are summarized in Table 2. The first columns depicts the references 1-30 of the scientific workflows. The three sub-columns in the fifth and sixth columns report the F-measure (F_1), the number of AND-split (#split), and the number of AND-join (#join) of the workflow models obtained by the approach CASS and BeehiveZ, respectively. The first three sub-columns of the eighth column report F_1, #split, and #join of the workflow models obtained by our approach (ProR) and the fourth sub-column reports the corresponding number of `links` (#links) introduced in the obtained BPEL-like workflow models. Table 2 demonstrates:

CASS obtains block-structured workflow models from DAGs, because the AND-splits and the AND-joins are in pairs. However, F_1 of the workflow models obtained by CASS are below 1 (except scientific workflow "4"), which indicates that CASS fails to obtain the workflow models with maximized concurrency. In our experiment, we find that CASS does not introduce false optimization opportunities; i.e., it does not allow activities with direct and indirect dependences to be executed in parallel, but it may miss some real optimization opportunities, that is, activities without dependences are arranged to be executed in sequence. Surprisingly, for most well-formed DAGs (scientific workflows), i.e., "1", "2", "7", "8", "12", and "20", CASS fails to obtain workflow models with maximized concurrency. In summary, CASS sacrifices a number of optimization opportunities to make the workflow models block-structured.

F_1 of the workflow models obtained by BeehiveZ are all 1. This implies that BeehiveZ is able to derive from the DAGs the workflow models with maximized concurrency. However, for most cases, in the obtained workflow models, AND-splits and AND-joins are not in pairs. Hence, the obtained workflow models can hardly be expressed in BPEL. Therefore, we draw the conclusion that BeehiveZ cannot guarantee obtaining block-structured workflow models. In addition, since BeehiveZ is based on a well-known workflow mining technique (i.e., the α-algorithm) for workflow refactoring, our experimental results also demonstrate that it is not the main goal of workflow mining to produce block-structured workflow models but to produce workflow models that justify the respective behaviour exhibited in the event logs.

F_1 of the workflow models obtained by our approach (ProR) remains 1, and the AND-splits and the AND-joins in these workflow models are in pairs. In addition, the number of `links` introduced by ProR is small. Notably, we do not know whether the number of introduced`links` is minimal, because there is no oracle for this. Nevertheless, for the well-formed scientific workflows with labels "1", "2", "4", "7", "8", "12", and "20", the workflow models obtained by ProR involve no `links`; for the not well-formed scientific workflows with labels "3", "5", "6", "11", "14", "17", "23",

only one `link` is introduced by ProR, respectively. Thus, for at least 14 of the 30 scientific workflows, ProR obtains workflow models with maximized concurrency and as block-structured as possible. For the other scientific workflows, ProR also introduces few `links` and the average number of `links` is 2.37 for these 30 workflows. Thus, ProR approaches to the optimal solution. Since scientific workflows tend to become more complex with increasing number of activities, the average number of `links` introduced increases from simple, medium, to complex workflows.

5.3 RQ2: Efficiency

Table 3 summarizes the runtime overhead of different refactoring approaches, where the first column shows the reference to the selection of the 30 scientific workflows we use. The two sub-columns of the last column report the runtime overhead for the two strategies of the third step in Algorithm 1, where the second strategy (Strategy 2) is the optimized one. According to Table 3:

All three approaches scale well. The runtime overhead of different approaches does not always increase with increasing workflow (DAG) size, because in addition to DAG size, the runtime cost also depends on the structural complexity (i.e., irregular dependence relations) of the DAG. However, the average runtime overhead increases from simple, medium, to complex workflows.

Our approach ProR is faster than BeehiveZ but slower than CASS. This is because ProR aims at maximizing both concurrency and block-structuredness, while other approaches only focus on one aspect. BeehiveZ is slowest as it needs to derive direct succession relations from traces of the original workflow [12].

Strategy 2 of ProR is more efficient than Strategy 1, though they are equivalent in deriving the optimal control flow graph. This is because fewer candidate subgraphs need to be considered by Strategy 2 for graph reduction. No matter which strategy is employed, ProR scales well with increasing DAG size. For instance, for the DAG with label "30", although, it involves 85 activities and 84 edges, Strategy 1 and Strategy 2 of ProR take only 53.8 ms and 47.4 ms, respectively, to obtain the BPEL-like workflow models. The runtime overhead of ProR confirms the runtime complexity result of our approach.

The above experimental results demonstrate that ProR, not only guarantees obtaining workflow models with maximized concurrency and near-maximized block-structuredness, but also scales well in practice. Although other approaches may be slightly faster, the refactored workflows do either not exhibit maximized concurrency or are not as block-structured as possible.

TABLE 3
Experimental Results - RQ2: Efficiency

Ref.	CASS (ms)	BeehiveZ (ms)	ProR (ms)	
			Strategy 1	Strategy 2
1	3.2	10.6	7.0	5.0
2	3.2	18.1	4.2	4.0
3	3.6	19.5	7.4	5.4
4	3.1	13.8	7.0	5.4
5	6.6	20.5	10.4	9.4
6	7.2	24.1	15.6	12.6
7	5.2	21.6	10.2	9.2
8	7.2	22.8	12.4	11.4
9	4	31.1	17.2	15.6
10	10	22.9	24.6	16.0
11	3.2	28	9.6	6.2
12	3.8	23.9	9.4	6.2
13	10.2	37.4	25	22.2
14	5.4	24.6	18.4	14.0
15	3.4	46.8	37.8	30.4
16	4.4	32.2	22.8	16.0
17	4.2	32.6	17.8	13.0
18	5.2	36.8	22.4	16.2
19	5.2	57.6	37.2	31.0
20	3.2	29.6	15	14.6
21	15.8	74.1	58.4	54.8
22	9.2	65.8	45.2	37.4
23	4.2	34.4	22.8	14.8
24	10.6	65.1	40.2	34.8
25	15.6	73.8	55.6	41.6
26	11.8	71.8	37.4	36
27	9	64.8	48	39
28	16	84.3	65.8	53.2
29	17.8	85	67.2ms	57.8
30	10.8	78.8	53.8	47.4

6 Related Work

In this section, we review related studies on workflow anti-patterns [37], [5], [6], [7], [8], [15], workflow transformation [38], [39], [20], [40], [20], [41], [21], workflow refactoring [17], [16], [9], workflow enhancement based on workflow mining [42], [11], [12], [23], and draw a comparison between these studies and our work.

Workflow anti-patterns. We first review well-established workflow anti-patterns which are classified into two types [3]: control flow anti-patterns [37], [5], [6] and data flow anti-patterns [7], [8], [15]. Deadlocks and lack of synchronizations are two common kinds of control flow anti-patterns [37]. If an AND-join node is used to synchronize different branches of an XOR-split node, there will be a deadlock. If an XOR-join node is used to merge different threads of an AND-split node, the problem of lack of synchronization occurs because the XOR-join initializes more than once. Typical data flow anti-patterns includes missing input, redundant output, and lost output [7], [15]. Unoptimized `sequence` orders in executable workflows (e.g., BPEL processes) can be regarded as another type of anti-patterns which are related to the interplay of both the control flow and data flow [10].

Workflow transformation. Transforming unstructured workflow models into equivalent block-structured ones has been intensively studied [38], [39], [20], [40]. These studies focus on identifying kinds of unstructured workflows that can be transformed into structured equivalents. If this is not possible, maximal structuring of acyclic workflows is studied in [40]. However, the structuredness is achieved at the expense of duplicated nodes, where our approach does not require this. Besides, some researchers focus on more general situations, that is, transforming graph-based workflows into block-oriented workflows [20], [41], [21]. For example, Ouyang *et al.* study how to transform BPMN models to block-structured BPEL workflows [21]. Although existing studies closely relate to our work, there are three

main differences. First, since only control flow is considered and data flow is abstracted in these studies, they do not take concurrency maximization into account. Second, most of the work focuses on graph-based workflow models while our approach on BPEL-like models. Third, our approach focuses on deriving optimal control flow based on a dependence graph, whereas the above studies do not.

Workflow refactoring. A great deal of work focuses on improving the internal implementations of services without affecting the external observable behavior. For example, Ratkowski *et al.* in [17] propose a BPEL transformation approach to enhance the non-functional properties (e.g., performance, modifiability, granularity, and maintainability) of the original BPEL workflow but do not change its behaviour. Feng *et al.* leverage the data flow information to restructure service workflows [16]. Their goal is to improve the performance of service implementations while keeping the service protocol unchanged. Unfortunately, neither of these approaches discusses achieving maximized concurrency or block-structuredness. Ni *et al.* focus on detecting concurrency-relating problematic activity arrangements in BPEL workflows [9] and determine optimal control flow relations of a BPEL workflow. However, as we show in Section 5, their algorithm fails to maximizing concurrency.

Workflow enhancement based on workflow mining. Some recent work utilizes workflow mining (including Petri nets synthesis) techniques for workflow refactoring and enhancement. For instance, Wang *et al.* leverage the theory of regions and Petri nets synthesis to find the optimal representation of a service composition [11]. Jin *et al.* employ workflow discovery technique (i.e., the α-algorithm) to help reconstruct data-aware Petri nets [12]. The workflow mining approach presented in [23] can discover workflows from dependence-complete event logs. However, all these techniques can at most ensure maximized concurrency, while the block-structuredness is not considered. Leemans *et al.* propose a mining approach which can obtain block-structured workflow from a dependence graph (in terms of direct successorship relations) [42]. However, the process underlying the event log ought to be a tree structure. Our previous work [23] uses activity dependences for process discovery, which can derive a workflow model from the discovered dynamic dependence graph. However, this approach only considers local concurrency maximization while global concurrency maximization is not ensured.

7 Conclusions

In this paper, we have presented a refactoring approach to maximizing concurrency and block-structuredness for BPEL-like workflows. The main idea of our approach is to search in the workflow dependence graph for well-formed patterns whose control flow relations with maximized concurrency are block-structured. If a workflow cannot be fully block-structured, our approach uses synchronization `links` to make the refactored workflow near block-structured. Since the problem of introducing minimum `links` is intractable, we present a greedy algorithm to achieve efficiency. The experimental results on real-world scientific workflows demonstrate that our approach can efficiently obtain workflow models with maximized concurrency and few `links`. Our approach is also applicable to process discovery. Our future work will focus on other algorithms to introduce fewer `links`.

Acknowledgments

This work was supported in part by the National Key R&D Program of China under Grant No. 2017YFB1001801, the National Natural Science Foundation of China under Grant No. 61761136003, the Natural Science Foundation of Jiangsu Province under Grant No. BK20171427, the Deutsche Forschungsgemeinschaft (DFG) project under Grant No. JA 2441/2-1, NSERC, and the Alexander von Humboldt Foundation under Grant No. 5090551.

References

[1] M. Hertis and M. B. Juric, "An empirical analysis of business process execution language usage," *IEEE Trans. Software Eng.*, vol. 40, no. 8, pp. 738–757, 2014.
[2] W. Song, F. Chen, H.-A. Jacobsen, X. Xia, C. Ye, and X. Ma, "Scientific workflow mining in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2979–2992, 2017.
[3] W. Song and H.-A. Jacobsen, "Static and dynamic process change," *IEEE Trans. Services Computing*, vol. 11, no. 1, pp. 215–231, 2018.
[4] J. Mendling, H. A. Reijers, and W. M. P. van der Aalst, "Seven process modeling guidelines (7PMG)," *Information & Software Technology*, vol. 52, no. 2, pp. 127–136, 2010.
[5] J. Koehler and J. Vanhatalo, "Process anti-patterns: How to avoid the common traps of business process modeling," *IBM WebSphere Developer Technical Journal*, vol. 10, no. 2, p. 4, 2007.
[6] S. Roy, A. S. M. Sajeev, S. Bihary, and A. Ranjan, "An empirical study of error patterns in industrial business process models," *IEEE Trans. Services Computing*, vol. 7, no. 2, pp. 140–153, 2014.
[7] N. Trcka, W. M. P. van der Aalst, and N. Sidorova, "Data-flow anti-patterns: Discovering data-flow errors in workflows," in *Advanced Information Systems Engineering, 21st International Conference, CAiSE'09, Amsterdam, The Netherlands, June 8-12. Proceedings*. Berlin: Springer-Verlag, 2009, pp. 425–439.
[8] H. S. Meda, A. K. Sen, and A. Bagchi, "On detecting data flow errors in workflows," *J. Data and Information Quality*, vol. 2, no. 1, pp. 4:1–4:31, 2010.
[9] Y. Ni, L. Zhang, Z. J. Li, T. Xie, and H. Mei, "Detecting concurrency-related problematic activity arrangement in WS-BPEL programs," in *IEEE International Conference on Services Computing, SCC'11, Washington, DC, USA, 4-9 July*. Washington: IEEE Computer Society, 2011, pp. 209–217.
[10] W. Song, X. Ma, S. C. Cheung, H. Hu, Q. Yang, and J. Lü, "Refactoring and publishing WS-BPEL processes to obtain more partners," in *IEEE International Conference on Web Services, ICWS'11, Washington, DC, USA, July 4-9*. Washington: IEEE Computer Society, 2011, pp. 129–136.
[11] Y. Wang, A. Nazeem, and R. Swaminathan, "On the optimal Petri net representation for service composition," in *IEEE International Conference on Web Services, ICWS'11, Washington, DC, USA, July 4-9*. Washington: IEEE Computer Society, 2011, pp. 235–242.
[12] T. Jin, J. Wang, Y. Yang, L. Wen, and K. Li, "Refactor business process models with maximized parallelism," *IEEE Trans. Services Computing*, vol. 9, no. 3, pp. 456–468, 2016.
[13] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, no. 3, pp. 319–349, 1987.
[14] OASIS, "Web services business process execution language version 2.0," April 2007.
[15] W. Song, C. Zhang, and H.-A. Jacobsen, "An empirical study on data flow bugs in business processes," *IEEE Trans. Cloud Computing*, PrePrints, doi: 10.1109/TCC.2018.2844247.
[16] Z. Feng, R. Peng, K. He, and Z. He, "Service restructuring by choreography-driven equivalence," in *IEEE Ninth International Conference on Services Computing, SCC'12, Honolulu, HI, USA, June 24-29*. Washington: IEEE Computer Society, 2012, pp. 407–414.
[17] A. Ratkowski, A. Zalewski, and B. Piech, "Transformational design of business processes in BPEL language," *e-Informatica*, vol. 3, no. 1, pp. 103–117, 2009.

[18] W. Reisig, "The synthesis problem," *Trans. Petri Nets and Other Models of Concurrency*, vol. 7, pp. 300–313, 2013.

[19] W. M. P. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, 2004.

[20] J. Mendling, K. B. Lassen, and U. Zdun, "On the transformation of control flow between block-oriented and graph-oriented process modelling languages," *International Journal of Business Process Integration and Management*, vol. 3, no. 2, pp. 96–108, 2008.

[21] C. Ouyang, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Mendling, "From business process models to process-oriented software systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 19, no. 1, pp. 2:1–2:37, 2009.

[22] M. G. Nanda, S. Chandra, and V. Sarkar, "Decentralizing execution of composite web services," in *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'04, October 24-28, Vancouver, BC, Canada*. New York, NY: ACM Press, 2004, pp. 170–187.

[23] W. Song, H.-A. Jacobsen, C. Ye, and X. Ma, "Process discovery from dependence-complete event logs," *IEEE Trans. Services Computing*, vol. 9, no. 5, pp. 714–727, 2016.

[24] J. E. Hopcroft and R. E. Tarjan, "Efficient algorithms for graph manipulation [H] (algorithm 447)," *Commun. ACM*, vol. 16, no. 6, pp. 372–378, 1973.

[25] Y. L. Varol and D. Rotem, "An algorithm to generate all topological sorting arrangements," *Comput. J.*, vol. 24, no. 1, pp. 83–84, 1981.

[26] D. Grigori, J. C. Corrales, M. Bouzeghoub, and A. Gater, "Ranking BPEL processes for service discovery," *IEEE Trans. Services Computing*, vol. 3, no. 3, pp. 178–192, 2010.

[27] M. Weidlich, J. Mendling, and M. Weske, "Efficient consistency measurement based on behavioral profiles of process models," *IEEE Trans. Software Eng.*, vol. 37, no. 3, pp. 410–429, 2011.

[28] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM J. Comput.*, vol. 1, no. 2, pp. 131–137, 1972.

[29] M. Mitchell, "Creating minimal vertex series parallel graphs from directed acyclic graphs," in *Australasian Symposium on Information Visualisation, InVis.au, Christchurch, New Zealand, 23-24 January*, 2004, pp. 133–139.

[30] L. Cai and F. Maffray, "On the SPANNING *k*-tree problem," *Discrete Applied Mathematics*, vol. 44, no. 1-3, pp. 139–156, 1993.

[31] G. Călinescu, C. G. Fernandes, H. Kaul, and A. Zelikovsky, "Maximum series-parallel subgraph," *Algorithmica*, vol. 63, no. 1-2, pp. 137–157, 2012.

[32] A. Augusto, R. Conforti, M. Dumas, and M. L. Rosa, "Split miner: Discovering accurate and simple business process models from event logs," in *IEEE International Conference on Data Mining, ICDM'17, New Orleans, LA, USA, November 18-21*, 2017, pp. 1–10.

[33] G. Booch, J. E. Rumbaugh, and I. Jacobson, *The unified modeling language user guide - covers UML 2.0, Second Edition*. Addison-Wesley, 2005.

[34] D. D. Roure, C. A. Goble, and R. Stevens, "The design and realisation of the $\text{my}_{\text{experiment}}$ virtual research environment for social sharing of workflows," *Future Generation Comp. Syst.*, vol. 25, no. 5, pp. 561–567, 2009.

[35] Z. Zhou, Z. Cheng, L. Zhang, W. Gaaloul, and K. Ning, "Scientific workflow clustering and recommendation leveraging layer hierarchical analysis," *IEEE Trans. Services Computing*, vol. 11, no. 1, pp. 169–183, 2018.

[36] E. Alpaydin, *Introduction to machine learning*. London: MIT press, 2014.

[37] W. Sadiq and M. E. Orlowska, "Analyzing process models using graph reduction techniques," *Inf. Syst.*, vol. 25, no. 2, pp. 117–134, 2000.

[38] B. Kiepuszewski, A. H. M. ter Hofstede, and C. Bussler, "On structured workflow modelling," in *Advanced Information Systems Engineering, 12th International Conference CAiSE'00, Stockholm, Sweden, June 5-9, Proceedings*. Berlin: Springer-Verlag, 2000, pp. 431–445.

[39] R. Liu and A. Kumar, "An analysis and taxonomy of unstructured workflows," in *Business Process Management, 3rd International Conference, BPM'05, Nancy, France, September 5-8, Proceedings*. Berlin: Springer-Verlag, 2005, pp. 268–284.

[40] A. Polyvyanyy, L. García-Bañuelos, D. Fahland, and M. Weske, "Maximal structuring of acyclic process models," *Comput. J.*, vol. 57, no. 1, pp. 12–35, 2014.

[41] S. White, "Using BPMN to model a BPEL process," *BPTrends*, vol. 3, no. 3, pp. 1–18, 2005.

[42] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs - A constructive approach," in *Application and Theory of Petri Nets and Concurrency - 34th International Conference, PETRI NETS'13, Milan, Italy, June 24-28. Proceedings*. Berlin: Springer-Verlag, 2013, pp. 311–329.

Wei Song received the Ph.D. degree from Nanjing University, China, in 2010. He is an associate professor in the School of Computer Science and Engineering, Nanjing University of Science and Technology, China. He was a visiting scholar at Technische Universität München, Germany, from October to December, 2015, and August, 2016 to August, 2017, and was a visiting student at The Hong Kong University of Science and Technology from October, 2008 to February, 2009. His research interests include software engineering and methodology, program analysis and testing, services and cloud computing, and process mining and analysis. He was invited to the Schloss Dagstuhl Seminar "Integrating Process-Oriented and Event-Based Systems" held in August, 2016. He has published in premiere computer science journals such as *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Services Computing*, and *IEEE Transactions on Software Engineering*. He is a member of the IEEE.

Hans-Arno Jacobsen received the Ph.D. degree from Humboldt Universität in Germany. He engaged in postdoctoral research at INRIA near Paris, France, before moving to the University of Toronto in 2001. He is a professor of computer engineering and computer science and directs the activities of the Middleware Systems Research Group. He conducts research at the intersection of distributed systems and data management, with particular focus on middleware systems, event processing, and cyber-physical systems. In 2011, he received the Alexander von Humboldt-Professorship to engage in research at the Technische Universität München, Germany. He is an IEEE Senior Member.

S.C. Cheung received the Ph.D. degree in computing from the Imperial College London. In 1994, he joined The Hong Kong University of Science and Technology, where he is a full professor of computer science and engineering. He participates actively in program and organizing committees of major international software engineering conferences. He was the general chair of the 22nd ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE 2014). He was a director of the Hong Kong R & D Center for Logistics & Supply Chain Management Enabling Technologies. His research interests include program analysis, testing and debugging, big data software, cloud computing, internet of things, and mining software repository.

Hongyu Liu received the M.S. degree from Nanjing University of Science and Technology Nanjing, China, in 2016. She is now working at ZTEsoft. Her research interests include software engineering, services computing, and workflow management.

Xiaoxing Ma received the Ph.D. degree in computer science from Nanjing University, China, in 2003. He is a full professor with the State Key Laboratory for Novel Software Technology and the Department of Computer Science and Technology, Nanjing University, China. His research interests include self-adaptive software systems, cloud computing, software architecture, and middleware systems. He co-authored more than 60 peer-reviewed conference and journal papers, and has served as technical program committee member on various international conferences. He is a member of the IEEE and the ACM.