Process Mining by Measuring Process Block Similarity

Joonsoo Bae[1], James Caverlee[2], Ling Liu[2], Bill Rouse[2], Hua Yan[2]

[1] Dept of Industrial & Sys. Eng., Chonbuk National Univ., South Korea
jsbae@chonbuk.ac.kr
[2] College of Computing, Georgia Institute of Technology
{caverlee, lingliu, huayan}@cc.gatech.edu

Abstract. Mining, discovering, and integrating process-oriented services has attracted growing attention in the recent year. Workflow precedence graph and workflow block structures are two important factors for comparing and mining processes based on distance similarity measure. Some existing work has done on comparing workflow designs based on their precedence graphs. However, there lacks of standard distance metrics for comparing workflows that contain complex block structures such as parallel OR, parallel AND. In this paper we present a quantitative approach to modeling and capturing the similarity and dissimilarity between different workflow designs, focusing on similarity and dissimilarity between the block structures of different workflow designs. We derive the distance-based similarity measures by analyzing the workflow block structure of the participating workflow processes in four consecutive phases. We first convert each workflow dependency graph into a block tree by using our block detection algorithm. Second, we transform the block tree into a binary tree to provide a normalized reference structure for distance based similarity analysis. Third, we construct a binary branch vector by encoding the binary tree. Finally, we calculate the distance metric between two binary branch vectors. Our initial experience shows that this distance measure can be used as a quantitative and qualitative tool for understanding and detecting block structure similarity and dissimilarity between two workflow designs. It can be effectively combined with a workflow precedence based similarity analysis tool in process mining, process merging, and process clustering, and ultimately it can reduce or minimize the costs involved in design, analysis, and evolution of workflow systems.

Keywords: Process Mining, Block Structure, Similarity
Topics: Process Discovery
Submission Category: Regular Paper

1 Introduction

Business process management has continued to attract attentions of both academics and industry. With the increasing interest and wide deployment of BPML, we see a growing demand for efficient business process management architectures and technologies that support enterprise transformation [7]. Effective enterprise transformation refers to strategic business agility in terms of how efficiently an enterprise can respond to its competitors and how timely an enterprise can anticipate

new opportunities that may arise in the future. In the increasingly globalized economy, enterprises face complex challenges that require rapid and possibly continual transformations. As a result, more and more enterprises are focused on the strategic management of fundamental changes with respect to markets, products, and services [8]. Such transformation typically has a direct impact on the business processes of an enterprise.

Fundamental to enabling the transformation of an enterprise is the development of novel tools and techniques for transforming the business processes of an enterprise. In this paper, we present a critical component to the problem of process transformation from a business process management point-of-view. Workflow precedence graph and workflow block structures are two important factors for comparing and mining business processes based on distance similarity measures. Although some existing work has done in comparing workflow designs based on their precedence graphs [GT tech report], there lacks of formal distance metrics for comparing workflows that contain complex block structures such as parallel OR, parallel AND. In this paper, we present a novel process difference analysis method using distance measures between block structure of two business processes. We present a quantitative approach to derive the distance-based similarity measures in four consecutive phases. We first convert each workflow dependency graph into a block tree by using our block detection algorithm. Then, we transform the block tree into a binary tree to provide a normalized reference structure for distance based similarity analysis. In the third phase, we construct a binary branch vector by encoding the binary tree. Finally, we calculate the distance metric between two binary branch vectors. The proposed difference analysis method achieves three distinct goals. First, by analyzing the block structures of process models, we present a quantitative process similarity metric to determine the relative distance between two process designs in terms of their block structure similarity. This similarity analysis facilitates not only the comparison of existing process models with each other, but also provides the flexibility to adapt to changes in existing business workflow processes. Second, the proposed method is quick and flexible, which reduces the cost of both the analysis and design phases of web service processes. Third, the proposed method enables the flexible deployment of process mining, discovery, and integration – all key features that are necessary for effective transformation of an enterprise. We argue that the block structure based distance measure can be effectively combined with a workflow precedence based similarity analysis tool [4] in process mining, process merging, and process clustering, and ultimately it can help to reduce or minimize the costs involved in design, analysis, and evolution of workflow systems.

2. Process Definition Model

The business process reference model (process model for short in the rest of the paper) consists of business process definitions and the specification of workflows among the processes with respect to data flow, control flow, and operational views [9]. We define a business process in terms of business activity patterns. An activity pattern consists of objects, messages, message exchange constraints, preconditions and postconditions [10], and is designed to specify the service actions and execution dependencies of the business process. We consider two types of activity patterns –

elementary activity patterns and composite activity patterns [1, 5]. An elementary activity pattern is an atomic unit. A composite activity pattern consists of a one or more elementary activity patterns or other composite activity patterns.

We define a business workflow as a collection of business activities connected by data flow and control flow, where each represents a business process. We use data flow among processes to define the data dependencies among processes within a given business workflow. We use control flow to capture the operational structure of the business workflow service, including the process execution ordering, the transactional semantics and dependencies of the workflow.

Formally, each workflow service is specified in terms of process definitions. We can model each process definition using a directed graph, in which the nodes of the graph are activities. The process *dependency graph* captures information about how activities share information and how data flows from one activity to another. Due to the space constraint, in this paper we focus our discussion only on the dependency graph.

Definition 1 (Dependency Graph, *DG*) A dependency graph DG is defined by a binary tuple <*DN, DE*>, where

- $DN = \{nd_1, nd_2, ..., nd_n\}$ is a finite set of activity nodes where $n \geq 1$.
- $DE = \{e_1, e_2, ..., e_m\}$ is a set of edges, $m \geq 0$. Each edge is of the form $nd_i \rightarrow nd_j$. ■

Note that in the dependency graph formulation, self-edges are disallowed since edges are intended to denote data flow dependencies between different activities (nodes). Additionally, a dependency graph must be a connected graph. Unconnected nodes and isolated groups of nodes are disallowed in the graph, as isolated nodes or groups of nodes are considered a separate service process in our reference model. Two dependency graphs are said identical if the two graphs have the same set of nodes and the same set of edges.

Given two workflow processes and their respective dependency graphs, there are numerous ways these two graphs may differ. Typically, it makes more sense to compare only those graphs that have sufficient similarity in terms of their dependency graphs. Consider two extreme cases: one is when the two dependency graphs have the same set of nodes and the other is when there is no common node between two graphs. By assigning 1 for the first case and 0 for the latter case, we define a comparability measure that indicates the ratio of common nodes in two graphs. One way to measure the extent of comparability between two graphs is to use a user-controlled threshold, called δ-Comparability, which is set to be between 0 and 1. Because this value represents the ratio of common nodes over the union of all nodes in two graphs, the larger the value is, the greater degree of comparability between the two graphs. Note that δ value can not be 0 since δ = 0 means that there is no common node between two graphs, i.e., $DN_1 \cap DN_2 \neq \varnothing$.

Definition 2 (δ-Comparability of *DG*)

Let $DG_1 = (DN_1, DE_1)$ and $DG_2 = (DN_2, DE_2)$ be two dependency graphs, and δ be a user-defined control threshold. We say that DG_1 and DG_2 are *δ-comparable* if the condition $\frac{|DN_1 \cap DN_2|}{|DN_1 \cup DN_2|} \geq \delta$ holds, where $0 < \delta \leq 1$ ■

If we apply the δ-Comparability to the example graphs shown in Fig. 1 with δ=0.5, g^0 and f^1 are not comparable because there is no common node in the two graphs, and also g^0 and f^2 are not comparable because the number of common nodes is only one but the number of total nodes is 7, that is $\frac{|DN_1 \cap DN_2|}{|DN_1 \cup DN_2|} = \frac{1}{7} < 0.5$. On the other hand, g^0 and g^1 are δ-comparable because all of the nodes in both graphs are common nodes. g^0 and g^2 are δ-comparable because there are 3 common nodes and the total number of nodes is 5, thus the two graphs satisfy the δ-comparability condition $\frac{|DN_1 \cap DN_2|}{|DN_1 \cup DN_2|} = \frac{3}{5} \geq 0.5$ and δ = 0.5.

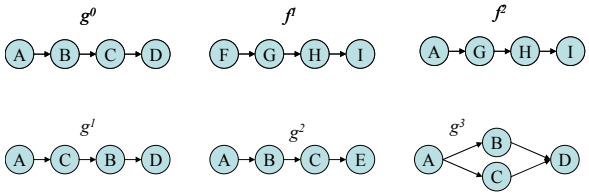


Fig. 1 Examples of δ-Comparability

3. Motivating Scenarios

Given the process reference model, we consider two motivating scenarios that benefit from the difference analysis methodology introduced in this paper. Consider a scenario where a company has maintained a warehouse of existing processes used in various business locations. *Process mining*[2, 3]of the process warehouse can help the enterprise to discover interesting associations or classifications among business processes running at different locations or branches of the company.

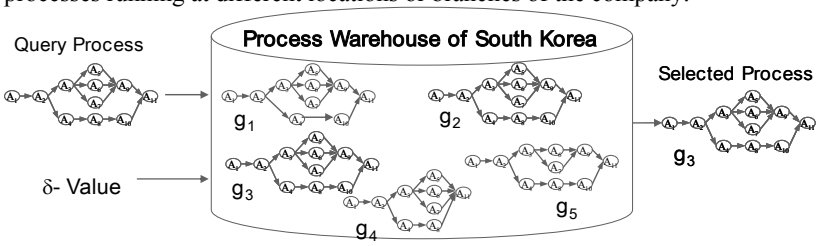


Fig. 2. Process mining example

In Fig. 2, we show a process warehouse that contains many types of processes (for example, g_1, g_2, g_3, g_4, g_5). A typical process mining scenario is the identification of

the processes most similar to a baseline process template in the process warehouse. Given a query process and a comparability threshold δ-value, the process mining will identify (g_3) the process that is most similar based on the comparability criterion. It is obvious that the concept of process similarity (or distance) is critical to the effectiveness of process mining.

4. Block Structure in Workflow

The first task in block structure based similarity analysis is to identify and extract structural patterns between two business processes. We assume that each business process and their process steps are described in terms of workflow and its activities in a precedence dependency graph. Our similarity comparison algorithm takes two workflow activity dependency graphs that satisfy δ-Comparability criteria as input and produces a block structure-based distance measure. Below we first introduce the concept of block types and then discuss the design of our block detection algorithm.

4.1 Block Types

If we use dependency graph in the process definition, the precedence relationships and existence of activities can be represented well. However, if there are splits in workflow (workflow denoting a parallel relationship), the dependency graph does not include the meaning completely. Thus, we need another representation method to measure parallel relationships. In this section the structure information that can be found in the dependency graph is used to define the distance measure between processes.

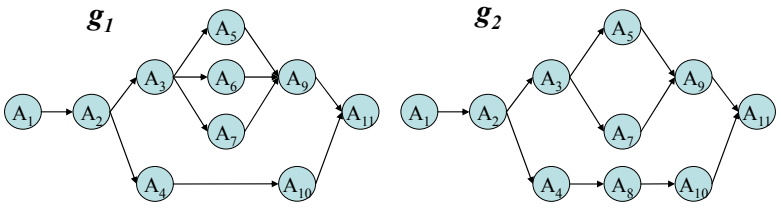


Fig. 3 Two example business processes represented in terms of workflow activities

Fig. 3 provides an example of a parallel relationship in the process definition, in which there are properties other than precedence relationship. One of other properties is parallel relationship and this parallel relationship comprises the structure of a process by using a nested relationship. In order to compare the structure properties, it is necessary to define how a process is composed of basic structures.

In order to represent the parallel relationship, the concept of block is introduced in this paper. A block is a unit of representation that can minimally specify the behavioral pattern of process flow. The behavioral patterns found in process models are classified into iterative, serial and parallel ones, each of which is illustrated in Fig. 4. Our discussion in this paper is confined to such networks that can be built by

combining those patterns. In this paper, the iterative block is not dealt with because it has less meaning in similarity of dependency and structure than serial and parallel. A serial pattern is shown in Fig. 4 (b). This pattern is simple in that it involves no iteration and has no split or merge in its task flow. But the serial block is related with the dependency measure which is discussed in the previous section. In this section, the parallel block is investigated in more detail.

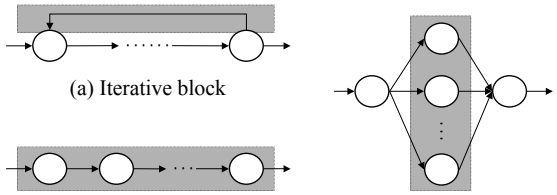


Fig. 4. Block types

A parallel pattern is such a flow that a node splits into two or more branches, the branches proceed in parallel, and merge into a node. Fig. 4 (c) is an illustration of this kind of pattern. The pattern is further subdivided into four types: AND-, XOR-, and SOR-parallel. Although all the parallel patterns are different in terms of their semantics, they have the same graphical structure. This is because the graphical objects of nodes and arcs deal only with the split-and-merge relations of tasks. The semantics distinguishing the parallel patterns are usually specified on the split or merge nodes.

4.2 Block Detection Algorithm

Since a component task of a process can be a nested process, the structure relation can be represented as a block structure. So in this paper, a block detection algorithm is used in order to generate blocks in the process and the generated blocks are used in the development of distance measure. The block detection algorithm[5] searches serial block and parallel block alternately and modifies the original network to construct a block tree from a process definition.

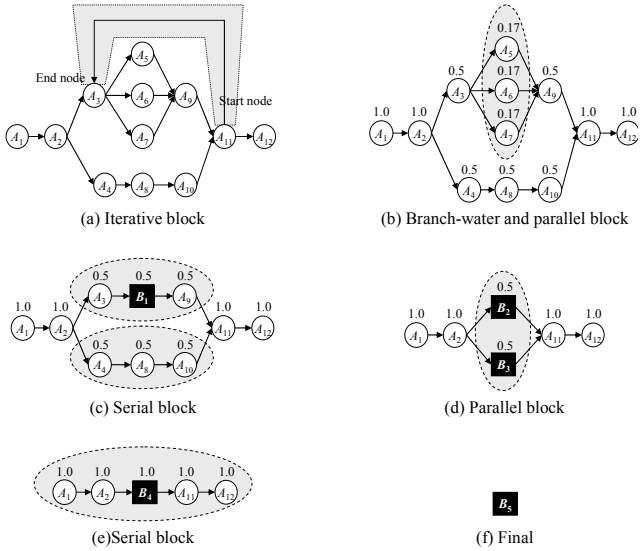


Fig. 5 Example of block generation

In reference [5], the block detection algorithm finds a cycle in the process network but these cycles are not used in the distance measure because the cycle does not affect the structure of a process. After serial blocks and parallel blocks are found alternately, this algorithm finally ends with a single node that means the uppermost block in the block tree. Fig. 5 provides an example of transforming a process network into block structure by using the block detection algorithm. In the example, the cycle block is removed at first, and then serial block, AND-parallel block, serial block, XOR-parallel block and serial block are generated. Since these blocks have hierarchy information, a tree can be made by combining the found blocks.

Definition 3 (Block Tree)

Let $DG_1 = (DN_1, DE_1)$ be the input and *T=(N, E, Root(T), label)* be the output of block detection algorithm respectively. The output is called block tree *N* is a finite set of nodes, which include the DN_1 and detected blocks. *E* is a binary relation on *N* where each pair *(u,v)* $\square$ *E* represents the parent-child relationship between two nodes *u, v* $\in$*N*. Node *u* is the parent of node *v* and *v* is one of the child nodes of *u*. Parent is nesting block and children are the nested components of the super block. There exists only one root note, denoted as *Root(T)* $\in$*N*, which has no parent. The root node is always serial block. Every other node of the tree has exactly one parent and it can be reached through a path of edges from the root. The nodes which have a common parent *u* (i.e., all the children of *u*) are siblings. |*T*| is the number of nodes in tree *T*, or the size of *T*. ■

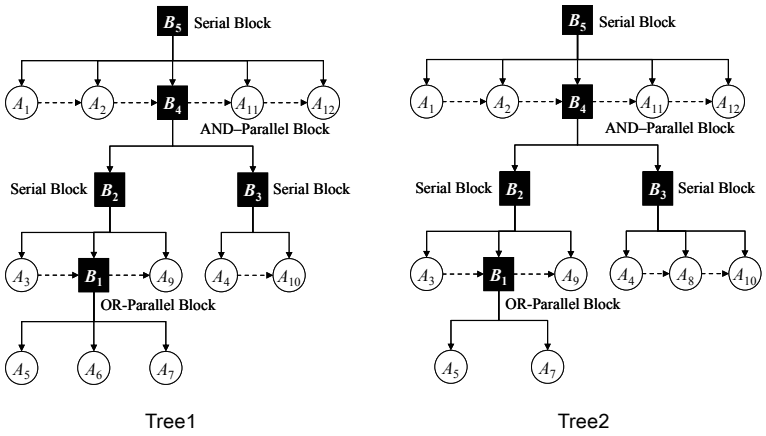


Fig. 6 Block Trees of Fig. 3

The block tree in Fig. 6 has a serial block B_5 as root node, which has 5 components, A_1, A_2, B_4, A_{11}, and A_{12}. These 5 components comprise first depth in the tree. Again a parallel block B_4 has two components, B_2 and B_3, which are serial blocks and second depth in the tree. At third depth, a serial block B_2 has three components, A_3, B_1, A_9 and a serial block B_3 has three components, A_4, A_8, A_{10}. Finally, three components A_5, A_6, A_7 are at the fourth depth as children of a parallel block B_1.

5. Structural Similarity in Tree Format

We have seen that block tree has two block types alternating serial and parallel as the depth increases in the previous section. So we can compare two block trees with the same block type (serial or parallel) in the same depth if we start from root node. We can define structural comparison by restricting the comparable depth in both block trees.

Definition 4 (Structural Comparison by Depth *d*)
If we have block trees $T_1=(N_1, E_1, Root(T_1), label_1)$, $T_2=(N_2, E_2, Root(T_2), label_2)$, the structural comparison by depth d is the block tree comparison from root node to depth d. ■

5.1 Structural Similarity Measure

The measure of similarity between two trees T_1 and T_2 has been well studied in combinatorial pattern matching. Most studies use edit distance to measure the dissimilarity between trees (notice that similarity computation is the dual problem of distance computation). [12]

5.2 Binary Tree Representation of Forests (or Trees)

Our proposed mapping of tree structures into a numeric vector space is based on the binary tree representation of rooted ordered labeled trees[11]:

Definition 5 (Binary Tree)
A binary tree consists of a finite set of nodes. It is:

1. an empty set. Or
2. a structure constructed by a root node, the left subtree and the right subtree of the root. Both subtrees are binary trees, too. ■

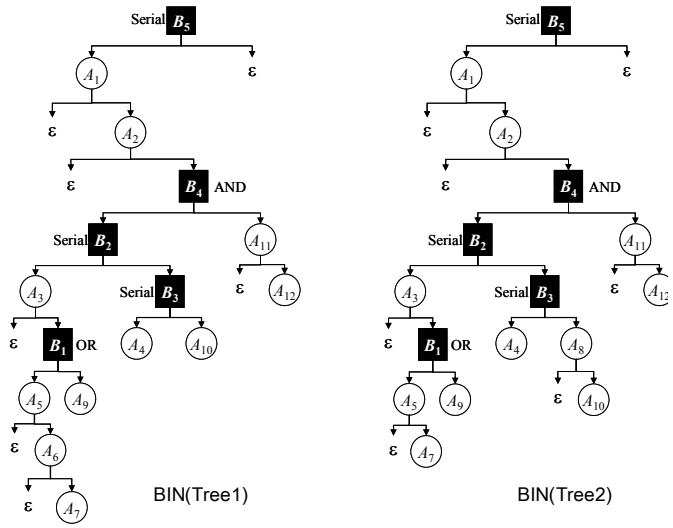


Fig. 6. Normalized Binary Tree Representation

There is a natural correspondence between forests and binary trees. The standard algorithm to transform a forest (or a tree) to its corresponding binary tree is through the left-child, right-sibling representation of the forest (tree):

(i) Link all the siblings in the tree with edges.

(ii) Delete all the edges between each node and its children in the tree except those edges which connect it with its first child.

Note that the transformation does not change the labels of vertices in the tree. We can transform T_1 and T_2 of Fig. 6 into $BIN(T_1)$ and $BIN(T_2)$ shown in Fig 7, respectively. The binary tree representation is denoted as $BIN(T) = (N, E_l, E_r, Root(T), label)$ in our paper.

A binary tree corresponding to a forest retains all the structure information of the forest. Particularly, in the binary tree representation, the original parent-child relationships between nodes, except the ones between each inner nodes and its first child, are removed. The removed parent-child relationships are replaced by the link edges between the original siblings. This property makes the transformed binary tree representation appropriate for highlighting the effect of the edit-based operations on original trees.

6. A Structural Similarity Measure

The key element of our algorithm is to transform rooted, ordered, labeled trees to a numeric multi-dimensional vector space equipped with the norm L1 distance. The mapping of a tree T to its numeric vector ensures that the features of the vector representation retain the structural information of the original tree. Furthermore, the tree-edit distance can be lower bounded by the L1 distance of the corresponding vectors. In this section, we present the transformation methods to get structural similarity measure.

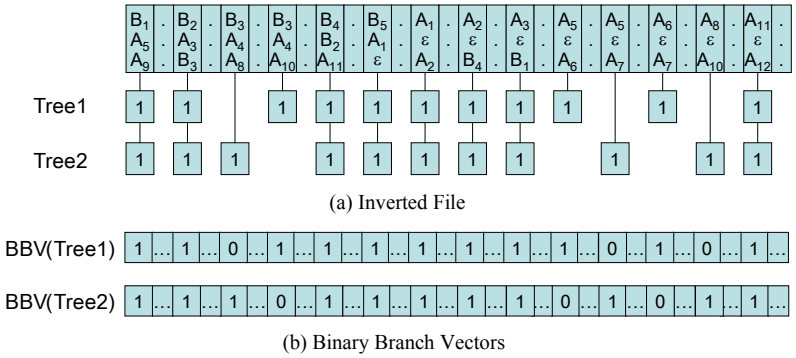
6.1 Vector Representation of Trees

To encode the structural information we normalize the transformed binary tree representation *BIN(T)* of *T*. In *BIN(T)*, for any node *u*, if *u* has no right (or left) child, we append a ε node (i.e., nodes labeled as ε do not exist in *T*) as *u*'s right (or left) child. Thus we make *T* a full binary tree in which all the original nodes have two children and all the leaves are labeled as ε (as in Fig. 7). The normalized binary tree representation is defined as $BIN(T) = (N \cup \{\varepsilon\}, E_l, E_r, Root(BIN(T)), label)$, where ε denotes the appended nodes as well as their labels. To simplify the notation, in this paper $u \in N$ represents the node as well as its label where no confusion arises. In order to quantify change detection in a binary tree, we define the binary branch on normalized binary trees:

Definition 6 (Binary Branch)
Binary branch (or branch for short) is the branch structure of one level in the binary tree. For a tree *T*, $\forall u \in N$ there is a binary branch *BiB(u)* in *BIN(T)* such that

$BiB(u) = (N_u, E_{u_l}, E_{u_r}, Root(T_u))$, where $N_u = \{u, u_1, u_2\}$ ($u \in N; u_i \in N \bigcup \{\varepsilon\}, i = 1,2$), $E_{u_l} = \{\langle u, u_1 \rangle_l\}$, $E_{u_r} = \{\langle u, u_2 \rangle_r\}$ and $Root(T_u) = u$ in the normalized *BIN(T)*. ■

Assume that the universe of binary branches *BiB()* of all trees in the dataset composes alphabet Γ and the symbols in the alphabet are sorted lexicographically on the string uu_1u_2.



(a) Inverted File

(b) Binary Branch Vectors

Fig. 7 Binary Branch Vector Representation

Definition 7 (Binary Branch Vector)
The binary branch vector *BBV(T)* of a tree *T* is a vector $(b_1, b_2, ..., b_{|\Gamma|})$, with each element b_i representing the number of occurrences of the ith binary branch in the tree. $|\Gamma|$ is the size of the binary branch space of the dataset. ■

We can first build an inverted file for all binary branches, as shown in Fig. 7 (a). An inverted file has two main parts: a vocabulary which stores all distinct values being indexed, and an inverted list for each distinct value which stores the identifiers of the records containing the value. The vocabulary here consists of all existing binary branches in the datasets. The inverted list of each component records the number of occurrences of it in the corresponding trees. The resulting vectors of our transformation for the block trees in Fig. 6 and the normalized binary trees in Fig. 7 are shown in Fig. 7 (b).

Based on the vector representation, we define a new distance of the tree structure as the L1 distance between the vector images of two trees:

Definition 8 (Structural Distance)
Let $BBV(T_1) = (b_1, b_2, ..., b_{|\Gamma|})$, $BBV(T_2) = (b'_1, b'_2, ..., b'_{|\Gamma|})$ be the binary branch vectors of trees T_1 and T_2 respectively. The structural distance of T_1 and T_2 is $BDist(T_1, T_2) = \sum_{i=1}^{|\Gamma|} | b_i - b'_i |$ ■

This structural distance has been proved that the distance properties met in reference [11]. And we can get the structural distance at depth 4 of Fig. 8 is 6.

In this part, we analyze the time and space complexities of our vector construction method. In order to calculate running time complexity, we consider each step of the algorithm. Assume that the size of the dataset, i.e., the total number of tree data objects, is |D|. For record T_i, there are $|T_i|$ nodes in it. The vocabulary of inverted file IFI is implemented by one hashing function. In order to traverse each node and insert the binary branch information of the current node into IFI, recursive function is called. Each time the new entries are appended at the end of the inverted list. So each update of IFI is of constant time complexity. Thus, the IFI construction is of linear complexity. As we store in IFI only the existing vocabulary of the dataset, the worst case is that all the nodes in the datasets have got different binary branches. Thus, the size of the vocabulary is at most $\sum_{i=1}^{|D|}|T_i|$. In addition, each node in each tree has one corresponding entry in the inverted list. In total, the space complexity of IFI is also $O(\sum_{i=1}^{|D|}|T_i|)$. To build the vector representation, the whole IFI has to be scanned once. So the time and space complexities of the whole vector construction algorithm are both $O(\sum_{i=1}^{|D|}|T_i|)$.

7. Conclusion and Future work

We have presented a structural difference analysis methodology between process definitions. Although there can be many difference attributes in process definitions, structural characteristics as well as dependency information are most important factors to discriminate processes. This paper focuses on the structural characteristic as a distance measure. We first convert each workflow dependency graph into block tree by using block detection algorithm. Second, the block tree is transformed into binary tree to make a binary branch. Third, binary branch vector is generated by encoding binary branch. Finally, we calculate the distance metric between the binary branch vectors. The proposed difference analysis method achieves three distinct goals. First, by analyzing the attributes of process models, we can present a quantitative process similarity metric to determine the relative distance between process models. This facilitates not only the comparison of existing process models with each other, but also provides the flexibility to adapt to changes in processes. Second, the proposed method is fast and flexible, which reduces the cost of both the analysis and design phases of complex web service processes. Third, the proposed method enables the flexible deployment of process mining, discovery, and integration – all desirable functionality that are critical for fully supporting the effective transformation of an enterprise. The next research issue is to integrate structural distance into dependency distance in process definition. And we are interested in developing a prototype system that provides efficient implementation of various similarity analysis methods, including the dependency distance metric presented in this paper.

Acknowledgments. The first author was supported by the Korea Research Foundation Grant (KRF-2004-003-D00477).

References

1. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros, "Workflow Patterns," Distributed and Parallel Databases, 14(3), pages 5-51, July 2003.
2. van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G. & Weijters. A.J.M.M. (2003). Workflow Mining: A Survey of Issues and Approaches, Data and Knowledge Engineering. 47(2), 237-267, 2003
3. van der Aalst, W.M.P., Weijters, A.J.M.M., & Maruster, L. (2004). Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering. 16(9), pp. 1128-1142, 2004
4. Bae, J., Caverlee J., Liu L., Rouse B., "Process Mining, Discovery, and Integration using Distance Measures," Technical Report GT-CSS-23-2006, Apr. 2006.
5. Bae, J., Bae, H., Kang, S., Kim, Y.: Automatic control of workflow process using ECA rules. IEEE Trans. on Knowledge and Data Engineering, vol.16, no.8 (2004) 1010-1023.
6. Bunke, H., Shearer, K.: A Graph Distance Metric based on the Maximal Common Subgraph. Pattern Recognition Letters, vol.19, issues 3-4, (1998) 255-259.
7. Hammouda, K.M., Kamel, M.S.: Efficient Phrase-Based Document Indexing for Web Document Clustering. IEEE Transactions on Knowledge and Data Engineering, vol.16, no.10 (2004) 1279-1296.
8. Rouse, W. B., "A Theory of Enterprise Transformation," Systems Engineering, vol. 8, no. 4, 2005.
9. Rush, R., Wallace, W.A., "Elicitation of knowledge from multiple experts using network inference," IEEE Transactions on Knowledge and Data Engineering, vol. 9, no. 5 (1997) 688-698.
10. WfMC, Workflow Management Coalition Workflow Standard Process Definition Interface -- XML Process Definition Language, Document Number WFMC-TC-1025 Version 1.13, September 7, 2005
11. Yang, R., Kalnis, P, Tung, A., "Similarity Evaluation on Tree-structured Data," ACM SIMOD 2005, June 14-16, 2005, pp. 754-765
12. Zhang, K., Shasha, D.: Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. SIAM Journal of Computing, vol.18, no.6 (1989) 1245-1262.