Mining Exception Instances to Facilitate Workflow Exception Handling

San-Yih Hwang[†] Sun-Fa Ho
Department of Information Management
National Sun Yat-Sen University
Kaohsiung, 80424 Taiwan
syhwang@mis.nsysu.edu.tw

Jian Tang[‡]
Department of Computer Science
Memorial University of Newfoundland
St. John's, Newfoundland, A1B 3X5 Canada
jian@cs.mun.ca

Abstract

The importance of exception handling within the context of workflow management has been widely recognized. While some exceptions are expected at design time and thus can be incorporated into the workflow design via some flexible mechanism, others are totally unexpected. Previous work in handling unexpected workflow exceptions focuses on the run-time support to, for example, allow the rollback of some already completed activities, validate the correctness of dynamic workflow change, and deploy a solution to handle exceptions. Authorized persons are responsible for deriving solutions to handle exceptions. In this paper, we propose a novel approach to facilitating users in proposing solutions for resolving a given exception. Specifically, our approach scans through the previous records in handling exceptions, looking for those that are close to the current exception. The ways in which those exceptions were handled serve as useful information in determining how to handle the current one. Several algorithms are proposed and evaluated through both theoretical analysis and synthetic data set.

1. Introduction

Workflow management systems (WFMSs) support the execution of business processes. Unlike manufacturing processes widely seen in today's factories, business processes are much more complicated because they are market-centered description with the aim to fulfill a business contract or satisfy customers' need [5]. A typical business process has to include the description of the constituent activities, their control/data flow, the potential participants, the organization model, the referenced data, etc. [15]. A WFMS separates the specification of workflows from their execution. It provides convenient and powerful means for specifying a business process in the hope that all requirements can be captured at design time and then enforced at runtime. However, it is not guaranteed that designers always do a perfect job in defining a workflow type that totally represents all properties of the underlying business process. Besides, since the notion of business processes is conceptually at a higher level, workflows have to adapt to the changing environment quickly, resulting in the executions that deviate from the predefined plan. That is, WFMSs have to handle exceptions. It has been observed that exceptions happen rather frequently in real working environments [12, 13]. This highlights the importance of exception handling in the context of workflow management.

The need for handling exceptions has been identified by several researchers and research projects (e.g., EXOTICA[1], METEOR[13], ADOME [2] ADEPT[12], and WAMO [3]). Exceptions could be expected, but as they occur rarely they are not incorporated into the standard workflow type. Instead, another flexible mechanism is adopted to support explicit modeling of these exceptions. This type of exceptions has been studied intensively within the context of extended transaction models (e.g., see [4, 1, 8]). An extended transaction model requires the workflow designer to specify some properties about the constituent activities and sub-workflows, such as compensatible, retriable, alternating activities, etc. When an exception occurs, the workflow system handles it according to the given attribute values of the involved activities/sub-workflows. A typical situation would be to rollback

† This research was supported by the Development of Software Object Reuse Technology Project of Institute for Information Industry and Sponsored by MOEA, R.O.C.
‡ Supported in part by the Natural Sciences and Engineering Research Council of Canada, individual operating grant OGP0041916.

activities to some point by executing the corresponding compensating activities in reverse execution order and take an alternating path to continue executing.

Other exceptions are unexpected due to the consequences of changes in the environment or something that just cannot be predicted at design time. Possible approaches to handling such exceptions include ignore, retry, partial rollback followed by forward execution, add some extra activities, delete some planned activities, or any change to the part of the workflow definition that is not executed yet [3, 14]. Researchers in ADEPT project also proposed an approach to evaluating the correctness of a workflow schema change [11, 12].

Previous work in handling unexpected workflow exceptions focuses on the run-time support to, for example, allow the rollback of some already completed activities [14], validate the correctness of dynamic workflow change [12], and deploy solutions to handle exceptions [3]. It is up to the authorized persons to decide an appropriate solution to handle a given exception. In [14], we propose to provide a query interface to facilitate users in browsing the related information about the current workflow instance. In this paper, we propose to further incorporate another module that scans through the previous records in handling exceptions, looking for those that are similar to the current exception. The handling approaches of these exceptions serve as good sources in designing the solution to resolve the current one. We describe the kind of information about exceptions, define the degree of similarity between two exceptions, and propose several algorithms for searching similar exceptions. We also present results from applying the algorithms to synthetic data sets to show their relative performance.

The rest of the paper is structured as follows. In Section 2, we describe the model adopted in this paper for handling exceptions. The information about exceptions that needs to be recorded in database is also discussed. In Section 3, we present three search algorithms. Section 4 compares the performance of these algorithms. We conclude the paper by summarizing the main results and identifying issues for further study.

2 The Model

Figure 1 depicts the reference architecture adopted in this paper. A workflow definition, which involves the constituent activities and their (control/data) interdependencies, is specified via some workflow definition tool and stored in the workflow database, WFDB. The workflow engine coordinates the workflow execution in such a way that all constraints, such as inter-activity dependencies, are enforced as specified. Note that an activity can be conducted by either a human or a computer program. The interaction between the performer of an activity and the workflow engine is enabled by either a human interaction agent for a human activity or an application agent for a computerized activity. An agent monitors the execution of its activity and reports its status to the workflow engine when necessary.

When an agent detects the occurrence of an exception to its associated activity, it reports to the workflow engine. To properly handle the exception, the workflow engine interacts with the component, **Exception Handler**, to determine a handling approach. Exceptions could be resolved by, e.g.:

1. rolling-back to some point to undone some accomplished activities and start execution from there,
2. modifying the definition of the part of workflow that has not been executed yet, or
3. a combination of both.

To determine a suitable approach for handling an exception, Exception Handler goes through a two step

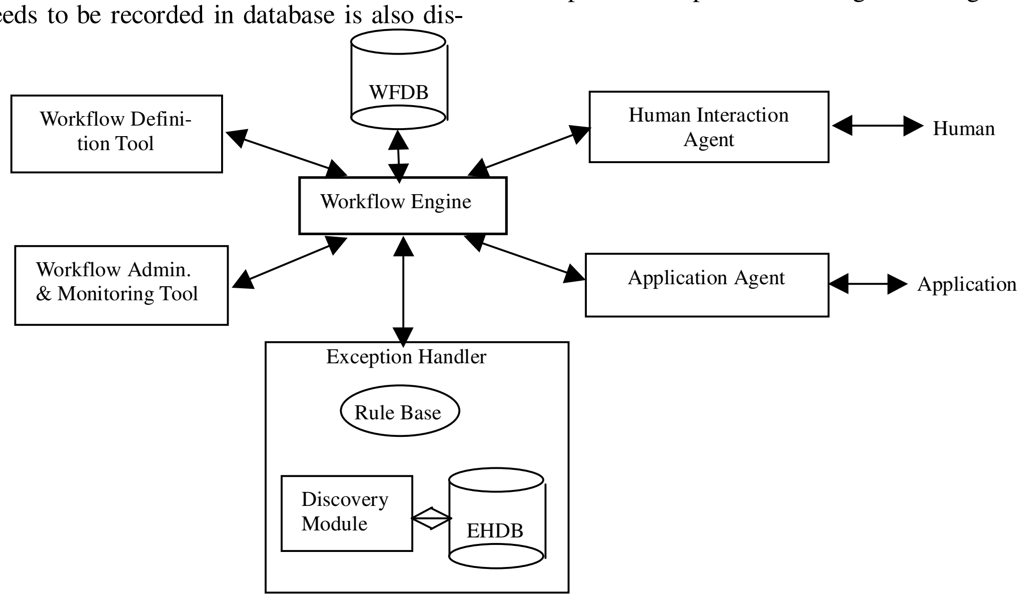


Figure 1. The Reference Architecture

procedure. First, it consults the rule base that contains a set of rules for handling exceptions. Each rule describes how to handle a specific exception type. That is, if the if-statement of some rule in the rule base matches the characteristics of the current exception, the then-statement will be used by the workflow engine to handle the exception accordingly. On the other hand, if none of the rules matches, the second step, conducted by Discovery Module, is taken. Assume a database, EHDB, contains the information about the previous experience in handling exceptions. The Discovery Module will try to find from EHDB previous exceptions that are similar to the current one. The approaches used to resolve those exceptions will be presented to the administrator as alternatives in handling the current exception. Once an alternative is chosen, possibly with some minor modification, it will be given to the workflow engine for implementation. This exception and its handling approach will be inserted into EHDB as a new record.

2.1 Records for Exceptions

To facilitate exception handling, our approach maintains the records about the past experience in handling exceptions. An exception can be characterized by the following attributes, among others:

1. *Status*: The status of the current workflow instance and its constituent activities. Possible status of a workflow instance includes initiated, running, suspended, terminated, active, and complete, while an activity may be in one of the following status: inactive, active, suspended, complete [15].
2. *Activity*: This dimension records the activity to which the exception occurs.
3. *Event*: It is the semantic information that describes the type of the exception. Events could be described by a set of user-defined keywords.
4. *When*: It denotes the time when the exception occurs.
5. *Who*: This attribute indicates the performer of the activity to which the exception occurs.

However, it may not be possible to find from these exception records the one whose attribute values are exactly the same as those of the current exception because, for example, two exceptions may not occur at the same time, their faulty activities may not be performed by the same person, etc. What we can best hope for is to identify the exception records whose attribute values are similar to those of the current one to some degree. To precisely define the concept of similarity, we incorporate the notion of concept hierarchies into our approach [6]. A concept hierarchy is a partial order of concepts, which indicates the general-to-specific ordering. Several of the attributes mentioned above can have their own concept hierarchies. For example, the nested workflow definition provides a natural means for defining the concept hierarchy for *Activity*. As the semantics are often organized into a taxonomy of concepts that are partial ordered [16], this taxonomy could serve as the concept hierarchy for *Event*. The organization hierarchy is an ideal candidate for the concept hierarchy of *Who*. Finally, *When* values can also have a concept hierarchy such as *Year-Month-Date-Time*.

Example 1. Consider an outpatient treatment workflow: shown in Figure 2. There is a sub-workflow *Diagnosis* in the definition that involves two sequential steps. The first step *Nurse's inquiry* is conducted by a nurse to collect some data about the patients, e.g., weight, height, and blood pressure. Then a doctor will do the second step *Doctor's diagnosis*. After the diagnosis, the patient may have to take some more tests and let the doctor read the result of the tests; or the doctor may just give out the prescription. Finally, the patient presents the prescription to the drug store and gets his dose.

Suppose a nurse Mary, when trying to perform *Nurse inquiry* activity for a patient, could not find this patient. The exception database stores a number of records about past exceptions, as shown in Table I.

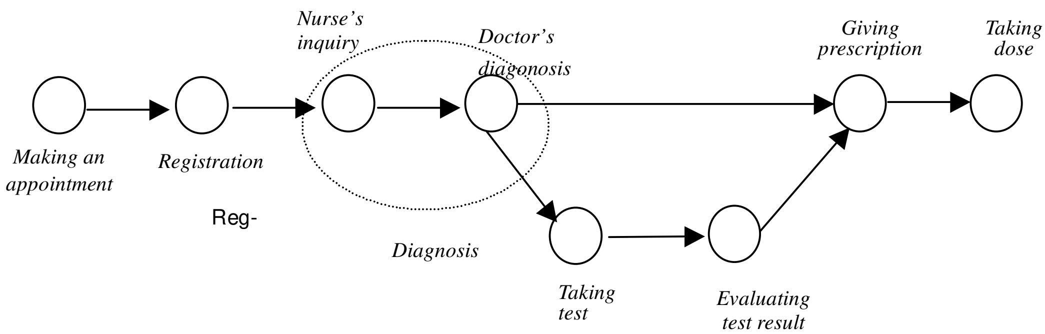


Figure 2. An Example Outpatient Treatement Workflow

Table I. A Sample Exception Database

When	Event	Who	Activity	Status	action
…	…	…	…	…	…
…	Cannot find patient	Ann (1)	Nurse's inquiry (0)	…	Skip and Fill in the data before *Giving prescription*
…	Cannot find patient	John (3)	Doctor's diagnosis (1)	…	abort
…	Cannot find patient	Tom (2)	Taking test (2)	…	pending
…	…	…	…	…	…

In the sample exception database, there are three records with the same event as the current exception, namely *Cannot find patient*. The first record describes an exception encountered by another nurse Ann when executing the same activity, and it was resolved by skipping this activity and prompting for required data entry before the prescription was decided (because the patient's data such as weight may play an important role in calculating the dose). The second exception was initiated by a doctor John when he tried to perform *Doctor's diagnosis* activity, and his approach was to abort the entire treating process. The third one was experienced by an examiner Tom when he intended to conduct some tests for the patient, and this activity was pended until the patient showed up. Now Mary focuses on *Who* and *Activity* attributes and would like to know how people handle exceptions with similar values on these attributes. Figure 3 depicts the concept hierarchies of both *Who* and *Activity*, where the number next to each level of a hierarchy is its level number.

The number in the parenthesis next to an attribute value in the exception database shown in Table I is the level number of the least common ancestor of this value and the attribute value of the current exception. It is clear that the first exception record, with the smallest parenthesis values in both Who and Activity, provides a good approach in handling the current exception.

□

Note that there may not be a unique concept hierarchy for each attribute. For example, There are two possible concept hierarchies for *When*, one characterized by weeks and the other by months. Exactly which concept hierarchy to use has to be decided by considering the significance of different concept hierarchies in this particular case.

The notion of finding similar records has been explored in various areas, e.g., k-nearest neighbors [10], classification [9], and case-based reasoning [7]. These prior work studied the data structures for organizing the existing records such that records similar to any given record can be found efficiently. However, the similarity functions they assume are different from that defined in our model. Besides, as will be shown in Section 3, our model allows users to dynamically specify the candidate attributes and their weights, which makes the preprocessing of existing records difficult, if not impossible. The following section describes the novel algorithms for identifying from records in the exception database the one with the smallest matching numbers (i.e., the numbers in the parentheses of Table 1) in the selected attributes.

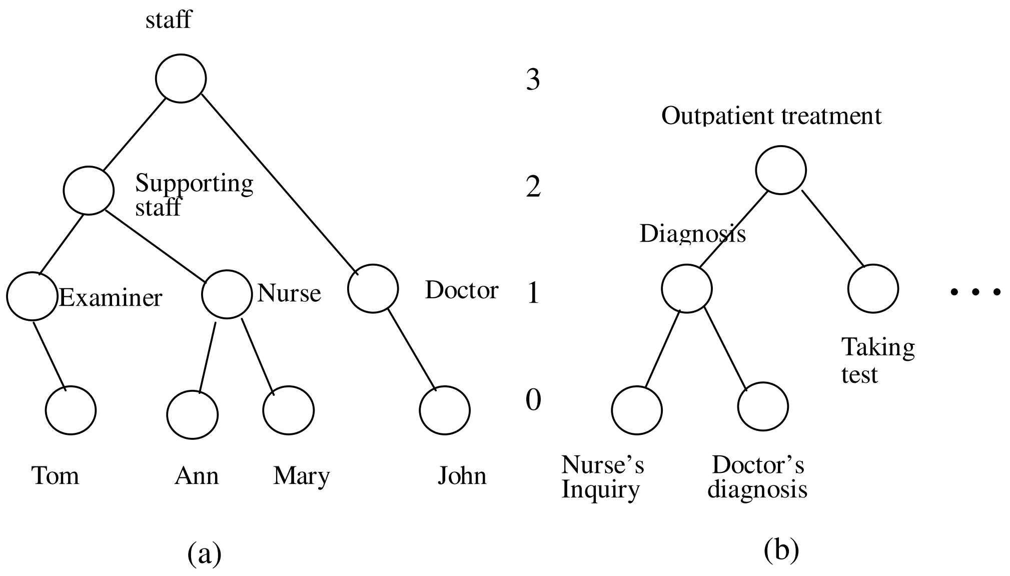


Figure 3 (a) Concept Hierarchy for *Who* (b) Concept Hierarchy for *Activity*

3.The Algorithms

Our goal in this section is to develop algorithms to allocate the exception records that have similar values on some attributes, called candidate attributes as specified by the user, with the current exception. To do so, the vague concept of similarity between exception records must be translated into a numeric measure.

Definition 1. Let k be the number of candidate attributes and $c_1, c_2, \ldots, c_k$ be the corresponding attribute values of the current exception. The matching tuple of an exception record $\langle a_1, a_2, \ldots, a_k \rangle$ is a k-tuple $\langle m_1, m_2, \ldots, m_k \rangle$, where m_i is the level number of the least common ancestor of a_i and c_i in the corresponding concept hierarchy, $1 \le i \le k$.

In Example 1, there are two candidate attributes: *Who* and *Activity*. The current exception is characterized as <Mary, Nurse’s inquiry>. The matching tuples of <Ann, Nurse’s inquiry>, <John, Doctor’s diagnosis>, and <Tom, Taking test> are <1, 0>, <3, 1>, and <2, 2> respectively.

Definition 2. A matching tuple $\langle m_1, m_2, \ldots, m_k \rangle$ is said to dominate another $\langle m_1', m_2', \ldots, m_k' \rangle$ if for each i, $1 \le i \le k$, $m_i \ge m_i'$.

We can see that both <3, 1> and <2, 2> dominate <1, 0>, while <3, 1> and <2, 2> do not dominate each other. It is clear that, given the current exception, if the matching tuple of an exception record *e1* dominates that of another *e2*, *e2* should be closer to the current exception than *e1*. However, if neither matching tuple dominates the other, we still need a way to tell which exception record is closer. We propose to adopt a weighting scheme that assigns a positive numeric weight to each candidate attribute. The weight of an attribute assigned by a user reflects her view on the importance of that attribute.

Definition 3. Given a k-tuple weight $\langle w_1, w_2, \ldots, w_k \rangle$, the weight of an exception record with matching tuple $\langle m_1, m_2, \ldots, m_k \rangle$ is $\sum_{i=1 \text{ to } k} w_i \times m_i$.

Discovery Module will take the current exception and a weight tuple as the input and discover from the exception database the record(s) with the lowest weight. In designing algorithms, we assume the size of the exception database is not very big, and all the records in the database can be placed in main memory[1]. We first present a sequential matching algorithm, abbreviated as SEQ-E, that traverses all the exception records once. For each record, its matching tuple is calculated. At the end of the traversal, the exception record with the minimum weight is identified. The algorithm is listed in Figure 4.

```
/* the current exception is <c1, c2, …, ck>. */
/* Li is the number of levels in the concept hierarchy of the i'th attribute. */
/* P[] contains the exception records. */
/* N is the number of records in EHDB. */
Function SEQ-E( ): EXCEPTION
Begin
Min_weight = Σi=1 to k wi×Li.;
For i=1 to N do
        Let <a1, a2, …, ak> be P[i];
For j = 1 to k do
For l = 0 to Li -1 do
If aj and cj meets at the l'th level in the concept hierarchy then
Begin
Mj = l;
Break;
                        Endif
                Endfor
        Endfor
        If Σi=1 to k wi×mi.< Min_weight then
        begin
Min_weight = Σi=1 to k wi×mi;
Closest_exception = P[i];
        Endfor
        Return Closet_exception;
End
```

Figure 4. The Pseudo Code of SEQ-E

Theorem 1. The complexity of SEQ-E is $O(\sum_{i=1 \text{ to } k} L_i \cdot N)$, where L_i is the number of levels of attribute i’s concept hierarchy and N the number of records in the exception database.

Proof. It is straightforward from the algorithm.

SEQ-E sequentially traverses every exception record in EHDB and, for each record, identifies its matching tuple. Another possible approach is to traverse all the possible concept level permutations in the ascending order of weight. Let $c_1, c_2, \ldots, c_k$ be the attributes values of the current exception. We say a concept level permutation $\langle l_1, l_2, \ldots, l_k \rangle$ is matchable if there exists an exception record $\langle a_1, a_2, \ldots, a_k \rangle$ in EHDB such that a_i and c_i meet at level l_i of the concept hierarchy of attribute i, $1 \le i \le k$. This traversal continues until the first matchable permutation is found. For example, suppose there are four candidate attributes (i.e., k=4), and the weight assigned to each attribute is equivalent. The traversal sequence on concept level permutation is <0, 0, 0, 0>, <0, 0, 0, 1>, <0, 0, 1, 0>, <0, 1, 0, 0>, <1, 0, 0, 0>, <0, 0, 0, 2>, …, $\langle L_1$-1, L_2-1, L_3-1, L_4-1$\rangle$. We call this approach SEQ-C, standing for SEQuential matching algorithm on Concept level permutations. However, this approach may consume even more time than SEQ-E as the average complexity is

[1] Due to space limitation, we do not present the work that drops this assumption. In fact, the proposed algorithms can be still applied with some modifications. Interested readers may contact the authors for the full paper.

$L_1 \times L_2 \times \ldots \times L_k \times N$, where L_i is the number of levels of attribute i's concept hierarchy and N the number of exception records in EHDB. To help reduce the time complexity, in the following we propose another algorithm that performs binary search on the sequence of concept level permutations, abbreviated as BIN-C.

Let C be the total number of concept level permutations, i.e., $C = L_1 \times L_2 \times \ldots \times L_k$. BIN-C conducts a binary search on the sequence of concept level permutations. Let $P[i]$ denote the i'th permutation in the sequence, $1 \leq i \leq C$. BIN-C first checks if $P[C/2]$ is matchable. If so, the permutations after $P[C/2]$ in the sequence are ignored, and the next permutation to check is $P[C/4]$. Otherwise, $P[3C/4]$ becomes the next permutation to check. This procedure continues until a matchable permutation $P[j]$ is found such that all the subsequent permutations $P[j/2]$, $P[3j/4]$, …, $P[j\ (2^{\log j}-1)/2^{\log j}]$ are not matchable.

In summary, BIN-C terminates at a matchable permutation such that some selected permutations with smaller weights are found non-matchable. The intuition behind BIN-C is that once such a permutation is found, the chance of the existence of any matchable permutation with smaller weight should be low. The algorithm for BIN-C is listed in Figure 5.

```
/* the current exception is <e1, e2, …, ek> */
/* Li is the number of levels in the concept hierarchy of the i'th attribute */
/* P[] contains all possible concept level permutations listed in ascending order of their weights */
Function BIN-C(K: INTEGER, exception: EXCEPTION): EXCEPTION
/* K is the upper limit in the permutation sequence, and "exception" is the one that matches the current exception at P[K] */
Begin
        Match = false;
        For i=1 to logK do
If Match(P[K(2^i-1)/2^i], exception) then
Begin
Match = true;
Break;
                Endif
        Endfor
        If Match = false then return exception;
        Else return BIN-C(K(2^i-1)/2^i)
END
```

Figure 5. The Pseudo Code of BIN-C

Theorem 2. The running time of BIN-C in the best case is $O(\log(\Pi_{i=1 \text{ to } k} L_i)\ N)$, where L_i is the number of levels of attribute i's concept hierarchy and N the number of exception records in EHDB.

Proof. The best case occurs when the following scenario occurs: once the first matchable permutation $P[k]$ is identified, it is found that all the following permutations $P[k(2^i-1)/2^i$, $1 \leq i \leq \log k$, are not matchable. Since there are $\Pi_{i=1 \text{ to } k} L_i$ permutations, the number of permutations visited in this case is $\log(\Pi_{i=1 \text{ to } k} L_i)$. Thus, the complexity is $O(\log(\Pi_{i=1 \text{ to } k} L_i)\ N)$.

Theorem 3. The running time of BIN-C in the worst case is $O(\Pi_{i=1 \text{ to } k} L_i\ N)$, where L_i is the number of levels of attribute i's concept hierarchy and N the number of exception records in EHDB.

Proof. The whole process of BIN-C can be viewed as a sequence of visited permutations. Some of the permutations are matchable while others are not. If there are quite a few (non-matchable) permutations between two adjacent matchable permutations in the sequence, the sequence tends to be longer as the two matchable permutations will not differ much in their weight. In the worst case, BIN-C has to visit all permutations and takes the same running time as SEQ-C: $O(\log(\Pi_{i=1 \text{ to } k} L_i)\ N)$.

The readers may have already noticed that the exception record returned by BIN-C may not be the optimal one. This is because BIN-C terminates at an exception record $P[k]$ only if $P[k]$ is matchable and all the following permutations $P[k(2^i-1)/2^i$, $1 \leq i \leq \log k$, are not matchable. However, this does not prevent the existence of some matchable exception record $P[j]$, where $j<k$. Besides, according to theorem 2 and 3, its execution time may or may not be better than SEQ-E. We have conducted some experiments to compare their relative performance, and the results will be shown in Section 4. Table II summarizes the running time of the three algorithms under various conditions.

Table II Running Time of SEQ-E, SEQ-C, and BIN-C

	SEQ-E	SEQ-C	BIN-C
Best Case	$O(\Sigma_{i=1 \text{ to } k} L_i \cdot N)$	$O(N)$	$O(\log(\Pi_{i=1 \text{ to } k} L_i)\ N)$
Worst Case		$O(\Pi_{i=1 \text{ to } k} L_i\ N)$	$O(\Pi_{i=1 \text{ to } k} L_i\ N)$
Average case		$O(\Pi_{i=1 \text{ to } k} L_i\ N)$	not clear

* N is the number of exceptions in EHDB

4. Performance Evaluation

We are still left with the problem of how the three algorithms peform under different operating environments. It can be seen from Table II that SEQ-E outperforms SEQ-C because both return the closest exception record(s), but SEQ-E runs faster than SEQ-C in average. However, there is no clear conclusion on the relative performance of SEQ-E and BIN-C. In this section, we present results of applying the algorithms to synthetic data sets. These results are intended to reveal the relative performance of the three algorithms in practical situations.

4.1. The parameter settings

Assume there are N exception records stored in EHDB, each of which contains values of B attributes. To simplify the performance study, we further assume that the concept hierarchy of every attribute is homogeneous in its structure. That is, each concept hierarchy is modeled to have *L* levels, and the degree of each non-leaf node is *D*. For the weighting scheme, we have exercised several combinations. In this paper we show two representative ones, namely *equal weighting* and *prioritized weighting*. The equal weighting scheme assigns the same weight to each attribute, and the prioritized weighting scheme prioritizes the attributes in a particular order. In our study, the weight tuple of the equal weighting scheme is <1,1, …, 1>. Without loss of generality, , let the priority sequence imposed by the prioritized weighting scheme be $w_1, w_2, \ldots, w_B$, where w_i has higher priority than w_j, $1 \leq i<j \leq B$. The weight tuple < $w_1, w_2, \ldots, w_B$ >of the prioritized weighting scheme is such that $w_B = 1$, and $w_i = w_{i+1}L$, $1 \leq i<B$. This assignment, for example, guarantees that the permutation <0, 1, 0, 0> has larger weight than <0, 0, *L*-1, *L*-1>. The exception records in EHDB and the current exception are randomly generated. These parameters and their settings are summarized in Table III.

Table III Parameters and Settings

Parameter	Meaning	Setting
N	Number of exception records in EHDB	10,000 … 100,000
B	Number of exception attributes	4
L	Number of levels in a concept hierarchy	6
D	Degree of each internal node in a concept hierarchy	5
W	Weighting scheme	Equal, Prioritized

4.2. The Experiments

We will not discuss SEQ-C in this subsection because it has been shown that SEQ-E always outperforms SEQ-C. Our focus here is to show how SEQ-E and BIN-C perform in a simulated environment, which was not clear from our previous theoretical analysis. Specifically, we would like to measure the *efficiency* and the *quality* of both algorithms. By efficiency, we refer to the running times of both algorithms. To measure the quality, we compare the concept level permutations obtained by both algorithms.

Figure 6(a) and 6(b) compare the efficiency of SEQ-E and BIN-C under equal and prioritized weighting schemes respectively. It can be seen that BIN-C constantly incurs less running time than SEQ-E in both schemes. We have also tried other weighting schemes, and the relative performance between SEQ-E and BIN-C remains approximately the same.

The exception record(s) returned by SEQ-E has the minimum weight, while those returned by BIN-C may be less optimal. To see how the exception record returned by BIN-C differs from that by SEQ-E in terms of weight, we conducted an experiment, whose results are shown in Figure 7. As expected, SEQ-E always returns the exception record with the smallest weight, which decreases as the number of records in EHDB increases. This is because we fix the structure of the concept hierarchy, and thus as the number of exceptions records increases, there is a higher chance to find a matching exception record with smaller weight. Besides, we can clearly see that the weight difference between the exception records returned by the two algorithms is very small. In several occasions, the results of BIN-C are as good as those of SEQ-E.

Overall, we conclude that BIN-C is an attractive approach under in-core environment in which it benefits quite a lot for efficiency and pays only a little price on the quality of the result. As the result returned by Discovery Module serves only as a reference to the administrator in deciding the proper approach to handling the current exception, small deviation from the optimal result should be acceptable in most cases.

5. Conclusions

Handling unexpected exceptions of workflows is a practical but difficult problem. In this paper, we proposed a model for handling workflow exceptions. This model provides a rule base that consists of a set of rules for handling exceptions. If none of the rules matches the current exception, a search on the previous experience in handling similar exceptions is conducted. Our focus in this paper has been on how an effective and efficient search can be performed. To do so, we have characterized exceptions, derived algorithms, namely SEQ-E, SEQ-C, and BIN-C, for finding past exceptions that are similar to the current one, and compared the relative performance of the three algorithms.

Through both theoretical analysis and experiments on synthetic data sets, we conclude that BIN-C is an attractive approach, especially when the number of past exception records is huge. BIN-C benefits quite a lot from execution time and sacrifices very little on the quality of the returned exception record(s).

Our future work includes deriving exception handling rules from the characteristics and the resolving approaches of past exceptions. The rule base in our model should not be a separate component functionally disjoint from the exception database. If an approach has been adopted to resolve many exceptions with similar features, rules could be derived to describe this scenario. These rules could be further confirmed and revised by the authorized persons and then become part of the standard procedure for handling exceptions in the future.

References

[1] Alonso, G., et al., "Failure Handling in Large Scale Workflow Management Systems," *IBM TR*, 1994.

[2] D. K.W. Chiu, K. Karlapalem, Q. Li, "Exception Handling in ADOME Workflow Management System," *Proc. of 2'nd Int'l. Conf. on Cooperative Information Systems*, 1997.

[3] J. Eder and W. Liebhart, "Contributions to Exception Handling in Workflow Systems," *EDBT Workshop on Workflow Management Systems*, Valencia, Spain, 1998.

[4] A. K. Elmagarmid, *Database Transaction Models for Advanced Applications*, Morgan Kaufmann Publishers, 1992.

[5] D. Georgakopoulos, M. Hornick, and A. Sheth," An Overview of Workflow Management: from Process Modeling to Workflow Automation Infrastructure," *Distributed and Parallel Databases*, 3(3), 1995

[6] J. Han, Y. Cai, and N. Cercone, "Knowledge Discovery in Databases: An Attribute-Oriented Approach," *Proc. of the 18th VLDB Conf.*, Vancouver, British Columbia, Canada 1992.

[7] J. Kolodner, *Case-Based Reasoning,* Morgan Kaufmann, 1993.

[8] Miller, J. A. , Palaniswami, D., Sheth, A.P., Kochut, K. J. , and Singh, H. , "WebWork: METEOR2's Web-Based Workflow Management System,", *Journal of Intelligent Information Systems*, 10(2), 1998.

[9] J. R. Quinlan, *Introduction to Decision Trees*, Machine Learning, 1, 1986.

[10] F. P. Preparata and M. I. Shamos, *Computation Geometry: An Introduction*, Springer-Verlag, 1985.

[11] M. Reichert, C. Hensinger, P. Dadam, "Supporting Adaptive Workflows in Advanced Application Environments," *EDBT workshop on Workflow Management System*, Valencia, Spain, 1998.

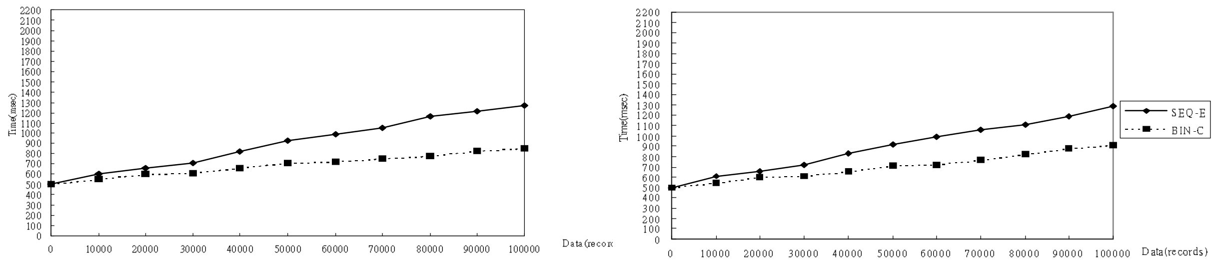
[12] M. Reichert and P. Dadam, "ADEPT—Supporting Dynamic Changes of Workflows without Losing Control," *Journal of Intelligent Information System*, 10(2), 1998.

[13] A. Sheth, K. J. Kochut, "Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems," *Proc. of NATO Advanced Study Institute on Workflow Management Systems and Interoperability*, Istanbul, Turkey, 1997.

[14] J. Tang, and S.-Y. Hwang, "A Scheme to Specify and Implement Ad-Hoc Recovery in Workflow," *Proc. of 6'th Int'l. Conf. On Extending Database Technologies*, Valencia, Spain, Springer LNCS 1377, 1998.

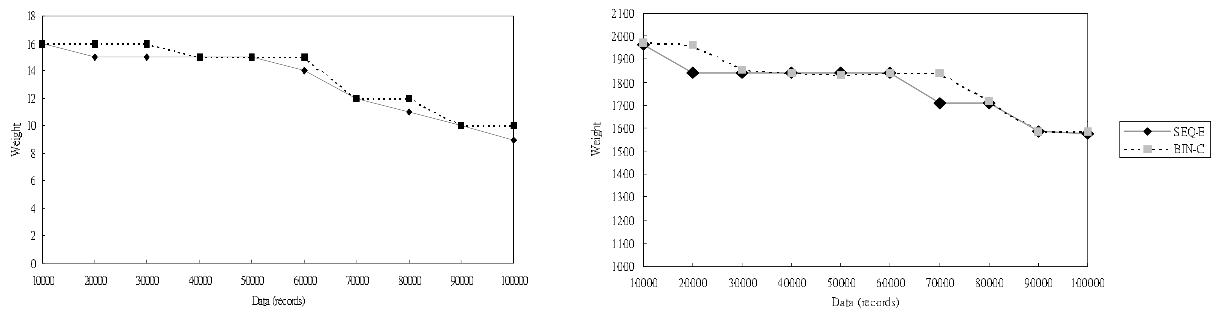
[15] Workflow Management Coalition, *The Workflow Reference Model*, 1994.

[16] P. H. Winston, *Artificial Intelligence*, 3'rd Ed., Addison-Wesley, 1993.



(a) (b)

Figure 6 (a) Running Time for Equal Weighting Scheme (b) Running Time for Prioritized Weighting Scheme



(a) (b)

Figure 7 (a) Weight of the Permutations for Equal Weighting Scheme (b) Weight of the Permutations for Prioritized Weighting Scheme in in-core environment