Supporting On-the-fly Provenance Tracking in Stream Processing Systems 1
Watsawee Sansrimahachai, Luc Moreau, and Mark J. Weal

A Job Hunting Matching System for University Students 12
Taku Jiromaru and Tokuro Matsuo

Actionable MOOCs through Course Certifying Agency Framework 18
Yeong-Tae Song, Chris Connoly, and Yongik Yoon

Measurement and Development Cost Implications of Component Commonality
in Software Product Line Engineering 27
Hamad Alsawalqah, Sungwon Kang, Bashar Al-Shboul, and Ji hyun Lee

A Polynomial-Time Algorithm for Checking the Equivalence for Real-Time
Deterministic Restricted One-Counter Transducers Which Accept by Final State 45
Mitsuo Wakatsuki, Etsuji Tomita, and Tetsuro Nishino

Distributed-Shogi-System Akara 2010 and its Demonstration 55
Kunihito Hoki, Tomoyuki Kaneko, Daisaku Yokoyama, Takuya Obata, Hiroshi Yamashita, Yoshimasa Tsuruoka, and Takeshi Ito

The International Association for Computer & Information Science (ACIS)
www.acisinternational.org acis@acisinternational.org
Supporting On-the-fly Provenance Tracking in Stream Processing Systems

Watsawee Sansrimahachai  
School of Science and Technology  
University of the Thai Chamber of Commerce, Thailand  
Email: watsawee@yahoo.com

Luc Moreau  
Electronics and Computer Science  
University of Southampton, UK  
Email: L.Moreau@ecs.soton.ac.uk

Mark J. Weal  
Electronics and Computer Science  
University of Southampton, UK  
Email: mjw@ecs.soton.ac.uk

Abstract

A new class of data management systems that operate on high-volume streaming data is becoming increasingly important. As this kind of systems has to process unpredictable streaming data in real-time and deliver instantaneous responses, it becomes very difficult to precisely validate stream processing results in timely manner, verify stream computation that took place and investigate processing steps used to generate result data. Therefore, a mechanism that can precisely track provenance of data streams at execution time is crucial for confidence in the results produced by this kind of systems. This paper presents a novel on-the-fly stream provenance tracking mechanism that enables a collection of provenance queries to be performed dynamically without requiring provenance information to be stored persistently. The experimental results indicate that the impact of provenance collection on system performance is relatively small (7% overhead observed). In addition, our provenance solution offers low-latency processing (about 0.3 ms per additional component) with reasonable memory consumption.

Keywords: Provenance, Data Streams, On-the-fly Provenance Tracking.

1. Introduction

Major changes in daily life have been caused by recent advancements in micro-sensor and wireless communication technologies. The functionality and usability of sensor technologies enable several kinds of sensors to be deployed in a wide variety of environments. Since the price of these devices is becoming cheaper, sensors are increasingly considered as commodity products that anyone can afford to buy. With the growth of sensor technologies, the time may come in the near future that every real-world object is tracked by several kinds of sensors which report its status or location in real-time or near real-time. This will lead to a significant increase of wide range environment monitoring, control and also data management applications that operate over streaming data with high-volume and real-time processing requirements.

There are a number of significant requirements that these kinds of systems need to satisfy [13]. One of the most important requirement is that a stream processing system needs to process data streams in real-time or near real-time, and provide an instantaneous response to support a precise and on time decision. Another significant requirement concerns dynamic and low latency processing. A stream processing system must be able to process stream events on-the-fly without any requirement to store them. With these requirements, the support for real-time provenance tracking in this kind of systems is extremely important. This functionality allows users to validate result data in timely manner, verify execution that took place in data stream generation and diagnose processing steps that are potentially the cause of errors in real-time.

Imagine that in a radioactivity leak incident in a nuclear submarine (such as [12]), emergency services operators rely on a real-time mapping (GIS) application in order to manage and control the disaster. The information displayed on the GIS application is submitted by several sensors located near the scene of the incident. The sensor measurements received by the GIS system are also forwarded to an early warning system where predictions of a possible ‘dirty bomb’ event are made in real-time. Because of extreme weather conditions, some sensors were damaged and continuously submit their faulty measurements into the GIS system. At some point, an operator has received a report indicating that there is an explosion in the nuclear reactor. The operator questions why this explosion was not automatically detected by the early warning system and why the level of radioactive material shown on the map display was not classified as being potentially dangerous radioactive intensity levels. A stream-specific provenance functionality that can be operated dynamically in real-time would allow the operator to validate stream processing results (e.g. predicted events) in a timely manner, trace back incorrect information to its origin before particular critical decisions are made, and also continuously verify sequence of stream processing steps used to produce those results.

This situation illustrates a common problem found in existing stream processing systems and more particularly those used for supporting on-time and critical decision making. We derive a strong requirement for dynamically tracing the stream events that caused a given output stream event without storing provenance information. Such capability is referred as on-the-fly stream provenance tracking.

To address such a provenance challenge in stream processing systems, we introduce a novel on-the-fly provenance tracking mechanism. The key concept is that we
exploit a provenance service - a central component of the stream provenance system - as a stream component. Provenance assertions - assertions pertaining to provenance recorded by each stream operation for individual stream elements - streamed to the provenance service are processed dynamically without any requirement to store them persistently. We extend the persistent provenance mechanism for streams presented in our previous work [11] by introducing the idea of property propagation. We do not propagate provenance-related properties through the processing flow of a stream processing system because this requires the modification of the internal processing of stream operations; instead properties are propagated inside the provenance service using accumulators. By utilizing a new version of stream ancestor functions - reverse mapping functions used to express dependencies between input and output elements of stream operations, properties can be propagated and provenance query results produced as a stream.

The use of our on-the-fly provenance tracking mechanism offers two major advantages. Firstly, because provenance assertions are not required to be stored persistently, the storage problem caused by storing provenance information for high-volume stream events can be solved. Secondly, as the query results are generated as a stream, this allows for provenance systems to offer instantaneous query results to their users.

This paper makes the following key contributions:

- It presents an on-the-fly provenance tracking mechanism for stream systems based on the idea of using a provenance service as a stream component.
- It defines a set of stream ancestor functions designed to work with the on-the-fly provenance mechanism.
- It identifies the important characteristics of this on-the-fly provenance tracking mechanism.
- It proposes a novel on-the-fly provenance tracking algorithm for stream processing systems.
- It presents the performance characteristics of our provenance solution for streams.

The rest of this paper is organized as follows. Section 2 reviews previous work in the area of provenance and stream processing systems. Section 3 introduces an on-the-fly provenance tracking mechanism for streams. Section 4 presents an on-the-fly provenance tracking algorithm. Section 5 demonstrates the experimental evaluation in terms of the impact of provenance recording, the memory consumption and the time latency. Finally, Section 6 presents conclusions.

2. Related work

Considerable research efforts have been made by the database community to address the provenance problem. Woodruff and Stonebraker [16] proposed a technique called weak inversion used to regenerate input data items that produced an output. The drawback is that the answer returned by this function is not guaranteed to be perfectly accurate. Buneman et al. [1] draw a distinction between two types of provenance: “why-provenance” and “where-provenance”. Why-provenance determines what tuples in the source database contributed to an output data item. Where-provenance, on the other hand, identifies locations in the source database from which the data item was extracted. Based on these types of provenance, a technique that propagates annotation according to propagation rules [2] has been proposed. Green et al. [6] define provenance semirings as a formal way of understanding “how-provenance” which describes how the input data leads to the existence of the output data. Annotations in the form of variables are propagated to form polynomials with integer coefficients for the output tuples.

In the context of scientific systems, Taverna [10] provides support for provenance tracking to allow scientists to understand how results from experiments were obtained. Provenance information is collected by recording metadata and intermediate results during workflow enactment. Another system, PASOA [7], built an infrastructure for reasoning over provenance in service-oriented architectures. By recording assertions comprising interaction messages and causal relationships between messages, provenance of data products can be captured. Our provenance solution extends the PASOA provenance mechanism. In our solution, a stream operation is treated as a “grey box” [4] and provenance is collected based on input-output dependencies of the operation. However, because PASOA needs to store all dependencies and intermediate data objects, the amount of information recorded can potentially cause a storage burden problem.

In the area of distributed stream processing, Vijayakumar et al. [15] propose a coarse-grained provenance model that identifies dependencies between streams or sets of stream elements as the smallest unit for which provenance is collected. However, the level of granularity for capturing provenance in this model is not detailed enough to identify dependencies among individual stream elements. The Time-Value-Centric model [9] (TVC) is a finer-grained provenance solution that provides the ability to express dependencies for individual stream elements by utilizing input-output dependencies described in terms of three primitive invariants: time, value and sequence. Nevertheless, this model still has some limitations, since it can fail to identify precisely input elements used in the production of an output. Another limitation is because all intermediate stream elements need to be stored for querying; this potentially leads to a storage problem when dealing with high volume streams. Another similar model [8] has the same limitation as TVC as well.

To sum up, the approach presented in this paper improves over the state-of-the-art in multiple ways. It defines a fine-grained notion of provenance for streams similar to why-provenance, which can explain the presence of individual stream elements. It identifies a set of stream operations for which on-the-fly provenance tracking can be computed. It
applies an eager approach [14], since it propagates provenance related information at runtime to obtain provenance tracking results in real-time. Furthermore, it eliminates the storage problem caused by storing provenance information.

3. On-the-fly provenance tracking mechanism

To design an on-the-fly provenance tracking mechanism, we make the following assumptions describing stream processing systems that our mechanism is intended to support.

- A stream processing system is represented as a set of interconnected nodes, with each node representing a stream operation. Input stream events flow through a directed graph of stream operations and finally, streams of output events are presented to applications that subscribe to receive results.
- A data stream consists of a sequence of time ordered stream events and each individual event is composed of an event key - a unique reference of an individual stream event - and a content of stream event (data).
- Each individual stream element is associated with one or more provenance-related properties. A property in our context is a piece of information describing a quality, characteristic or attribute that belongs to an individual stream element.

To overview our stream provenance system, we present a provenance architecture for stream processing systems in Figure 1. In this paper we focus on the non-persistent subsystem. An on-the-fly provenance tracking service plays a central role in this subsystem. It is designed to be utilized as a stream component that executes continuous queries over continuous data streams. During an initialization period, users who manage a client application need to subscribe to receive query results from the service. Provenance assertions are recorded as a stream by each stream operation for each individual stream element. The provenance assertions are not generated at a single time, but instead their generation is interleaved continuously with execution. As provenance assertions are being received by the on-the-fly provenance tracking service, provenance queries are performed automatically and continuously over the streams of provenance assertions. It is important to note that the generation of on-the-fly provenance queries is based on configuration parameters and stream topology information specified during system registration time. Such information is stored in a compact provenance store - a compact version of the provenance store used only for storing static provenance information. Finally, provenance tracking results are generated and streamed back to the subscriber application.

3.1 Provenance data model

We now present the provenance data model that underpins on-the-fly provenance tracking in stream processing systems. As presented in Figure 2, the information contained in the provenance model can be divided into two parts. Static information is the provenance related information gathered during initialization time of a stream system. Static information includes stream topology information (stream operations, types of operations and input-output streams), configuration parameters and metadata. The other part - Dynamic information - is the provenance related information collected during execution time. Dynamic information includes a set of provenance assertions recorded as streams by stream operations during execution time. It is necessary to note that the static provenance information is only the part of our provenance model required to be stored in a compact provenance store. No persistent storage space is required for the dynamic provenance information due to the fact that we compute provenance-related properties on-the-fly over streams of provenance assertions (dynamic provenance information) without any requirement to store them persistently.
3.2 Basic concept of on-the-fly provenance tracking

We express dependencies between input and output elements of stream operations by means of a stream ancestor function (SAF) that is defined for each stream operation [11]. The key idea of SAF is that for a given reference to an output element, it lists references to input elements involved in the production of that output. The SAF does not work directly with individual stream elements, but instead it operates on a representation of each individual stream element - provenance assertion - that is recorded as streams by each stream operation. We assume that every assertion contains an event key, which consists of a timestamp, a sequence number, a stream identifier and a delay time. The event key serves as a unique reference for identifying each provenance assertion of an individual stream element in a stream. By composing SAFs for all stream operations in a stream processing system, all the intermediate stream elements involved in the processing of a particular output can be exactly identified.

We extend the concept of the SAF in our persistent provenance tracking mechanism [11] by adding stream-specific techniques tailored to address on-the-fly provenance tracking. Our key concept is inspired by the idea of property propagation. We assume that each individual stream element contains a provenance-related property in its accumulator - a field used to accumulate a list of provenance-related properties. For each stream operation, properties are computed and propagated automatically from input streams (ancestor streams) to output streams (descendant streams) based on input-output dependencies between stream elements. Each intermediate result produced by the processing of provenance-related properties is accumulated and temporarily stored in the accumulator field on intermediate stream elements. The processing of property propagation is performed continuously until reaching the final stream operation of a stream processing flow. Figure 3 illustrates an example of our property propagation approach in which properties are propagated through a stream processing flow.

However, it is important to note that we do not try to propagate properties within a stream processing system layer because this would require the modification of the internal processing of stream processing operations, but instead properties are computed and propagated through the use of provenance assertions inside a provenance service.

An example process of our on-the-fly provenance tracking inside the provenance service is shown in Figure 4. The execution begins with the provenance service receiving streams of provenance assertions (AS) generated by stream operations in a stream system. Each assertion is detected by an assertion separation unit (ASU) - a component used to detect provenance assertions for a particular stream and direct them to a SAF that they are associated with. In the case that arrival assertions are provenance assertions of the first-input event - stream events that enter a stream system from data sources and are first processed by stream operations, it will be sent to a property computing unit (PCU) first in order to extract properties. After that, each individual provenance assertion is computed by its associated SAF. The SAF utilized in our on-the-fly provenance tracking is the new version of the original SAF - called a property stream ancestor function (PSAF). It receives stream elements from both an assertion stream (AS) and result streams (RS) - the output from the previous step of property propagation - as an input and then it produces an output element belonging to a property stream (PS). Not only is the PSAF used to identify the ancestors of a particular provenance assertion, it is also used to extract properties from the ancestor provenance assertions (elements from RS). The output generated from the PSAF is fed into a PCU in order to compute property propagation. Once the property propagation is processed, an output provenance assertion containing a new property is produced as an element of a result stream (RS). This process of on-the-fly provenance tracking is executed continuously until it reaches the final PSAF that executes on provenance assertions of the final output stream.
3.3 Primitive stream processing operations

We now present the primitive stream operations supported by on-the-fly provenance tracking. The goal of this presentation is to demonstrate how each output element of a stream operation is produced in terms of its input elements. Based on these operations, we then present a list of specifications of property stream ancestor functions (PSAFs) in the subsequent section. Note that all stream operations and PSAFs are defined by using Event Processing Language (EPL) (sometime called StreamSQL [13]) - an SQL-like language extended to handle event streams. We use EPL provided by the Esper stream engine to illustrate how continuous queries are formulated in stream operations and how provenance-related properties can be computed and propagated. By using EPL, we believe that a concise and clear definition of PSAFs can be explained.

Our primitive stream operations include windowed operations that operate on sets of consecutive events from a stream at a time and operations that operate on a single event at a time. These operations are recognized as common operations developed in several stream projects [3] [5].

- **Map(F,sid)**: A map operation is a stream operation that operates on a single stream element at a time. The operation applies an input function \( F \) to the content of every element of an input stream.
- **Filter(P,sid)**: A filter operation screens events that satisfy a predicate \( P \) in an input stream.
- **TW(w,sid)**: A time window is a data window where the extent of the window is defined in terms of time interval. At any point in time, the time window generates an output event from the most recent input events over a given time period \( w \).
- **LW(l,sid)**: A length window is a data window where the extent of the window is defined in terms of the number of events. At any point in time, a length window covers the most recent \( N \) events (the size of window - 1) of a stream.
- **JoinTW(w1,w2,sid)**: A time window join pairs stream events from two input streams. Stream events from two time-based windows are combined and output events are produced according to a join condition. The parameters consists of the duration of time windows \( w1, w2 \).
- **JoinLW(l1,l2,sid)**: Similar to the time-window join, a length-window join pairs events from two input streams. The difference between these two operations is the length-window join operates on stream events from two tuple-based windows. The required parameters consist of the size of windows \( l1, l2 \).

The definitions of the primitive stream operations are described in Table 1. In this context, a stream id \( (sid) \) for every operation is the ID of an output stream. Note that the filter, time-window join and length-window join operations use a sequence number \( (sn) \) as an internal variable which is generated according to a number of output events.

Furthermore, every stream operation presented in the table execute stream event of the form Event(key(t,n,s,d),data).

3.4 Property stream ancestor functions

In this section, the stream ancestor functions (SAFs) of our on-the-fly provenance mechanism are described. We call this version of SAFs “Property stream ancestor functions” (PSAFs) because one of the most significant tasks for these SAFs is to extract provenance-related properties from ancestor provenance assertions. The definitions of PSAFs are presented in Table 2.

Every PSAF utilizes internal values of its associated stream operation as input parameters at registration time. For example, the PSAF for a sliding time-window \( (TW_{pad}) \) utilizes a duration of the time window \( w \) as a function parameter. Each PSAF takes a provenance assertion \( (AS) \) and the elements of a result stream \( (RS) \) - a stream generated from the previous property propagation step - as an input and generates an element of a property stream \( (PS) \) containing all provenance-related properties needed for further processing. Note that, for the PSAFs for binary stream operations (e.g. join\( TW \)), two result streams \( (RS1 \) and \( RS2 \)) are utilized as an input.

In addition, the \( PExtract \) function is used by each PSAF as a crucial internal function to extract provenance-related properties from elements in a result stream \( (RS) \). In the PSAFs for windowed operations \( (TW, LW, JoinTW and JoinLW) \), the \( PExtract \) function is parameterized by the size of a data window and a delay time for stream processing, in order to define the extent of a past data window at the time that the stream operation produced the output. This extent of a past data window is then used to identify the ancestor provenance assertions (the elements in \( RS \)) and also to extract provenance-related properties from them. For example, the definition of the PSAF for a sliding time-window \( (TW_{pad}) \) indicates that the \( PExtract \) function extracts properties from elements in \( RS \) by creating the extent of a past time-window where the interval of the window is between \( t - d \) (upper bound) and \( t - d - w \) (lower bound). Aside from windowed operations, there are operations that operate on a single element at a time such as Map and Filter operations. The PSAFs for these operations straightforwardly pass a timestamp and a delay time for stream processing to the \( PExtract \) function, which identifies an ancestor element from \( RS \) and finally extract provenance-related properties from that ancestor element. For example, the definition of the PSAF for a map operation \( (Map_{pad}) \) indicates that properties are extracted from an element in \( RS \) that was timestamped with a value \( t - d \).

4. An on-the-fly provenance tracking algorithm

We now present an on-the-fly provenance tracking algorithm that executes provenance queries continuously over streams of provenance assertions.
### Table 1: The definitions of primitive stream processing operation

<table>
<thead>
<tr>
<th>Stream operations</th>
<th>Definition - input: Event(key(t,n,s,d).data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map(F, sid)</td>
<td>(* fn : (‘a → ’b) * INT → ‘a EVENT list → ‘b EVENT list *)&lt;br&gt;insert into resultEvent(timestamp, seqNo, streamID, delay, eventData)&lt;br&gt;select current_timestamp, n, sid, (current_timestamp - t), F(data) from Event</td>
</tr>
<tr>
<td>Filter(P, sid)</td>
<td>(* fn : (‘a → bool) * INT → ‘a EVENT list → ‘a EVENT list *)&lt;br&gt;insert into resultEvent(timestamp, seqNo, streamID, delay, eventData)&lt;br&gt;select current_timestamp, n, sid, (current_timestamp - t), data from Event where P(data)</td>
</tr>
<tr>
<td>TW(w, sid)</td>
<td>(* fn : TIME * INT → ‘a EVENT list → ‘a EVENT list *)&lt;br&gt;insert into resultEvent(timestamp, seqNo, streamID, delay, eventData)&lt;br&gt;select current_timestamp, n, sid, (current_timestamp - t), list(data) from Event where win : time(w)</td>
</tr>
<tr>
<td>LW(l, sid)</td>
<td>(* fn : INT * INT → ‘a EVENT list → ‘a EVENT list *)&lt;br&gt;insert into resultEvent(timestamp, seqNo, streamID, delay, eventData)&lt;br&gt;select current_timestamp, n, sid, (current_timestamp - t), list(data) from Event where win : length(l)</td>
</tr>
<tr>
<td>JoinTW(w1, w2, sid)</td>
<td>(* fn : TIME * TIME * INT → ‘a EVENT list → ‘a EVENT list → ‘a EVENT list *)&lt;br&gt;insert into resultEvent(timestamp, seqNo, streamID, delay, eventData)&lt;br&gt;select current_timestamp, sn, d, (current_timestamp - MaxTime(e1.t, e2.t)), J(e1.data, e2.data) from Event1:win : time(w1) as e1, Event2:win : time(w2) as e2</td>
</tr>
<tr>
<td>JoinLW(l1, l2, sid)</td>
<td>(* fn : INT * INT * INT → ‘a EVENT list → ‘a EVENT list → ‘a EVENT list *)&lt;br&gt;insert into resultEvent(timestamp, seqNo, streamID, delay, eventData)&lt;br&gt;select current_timestamp, sn, d, (current_timestamp - MaxTime(e1.t, e2.t)), J(e1.data, e2.data) from Event1:win : length(l1) as e1, Event2:win : length(l2) as e2</td>
</tr>
</tbody>
</table>

### Table 2: The definitions of property stream ancestor functions (PSAFs)

<table>
<thead>
<tr>
<th>PSAFs</th>
<th>Definition – input: AS/RS - Assertion(key(t,n,s,d),prop)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map_psaF ()</td>
<td>(* fn : ‘a ASSERTION list → ‘b ASSERTION → ‘a ASSERTION *)&lt;br&gt;insert into PS(timestamp, seqNo, streamID, delay, property)&lt;br&gt;select t, n, s, d, PExtract.compute((select * from RS), t – d) from AS</td>
</tr>
<tr>
<td>Filter_psaF ()</td>
<td>(* fn : ‘a ASSERTION list → ‘b ASSERTION → ‘a ASSERTION *)&lt;br&gt;insert into PS(timestamp, seqNo, streamID, delay, property)&lt;br&gt;select t, n, s, d, PExtract.compute((select * from RS), t – d) from AS</td>
</tr>
<tr>
<td>TW_psaF (w)</td>
<td>(* fn : TIME * ‘a ASSERTION list → ‘b ASSERTION → ‘a ASSERTION *)&lt;br&gt;insert into PS(timestamp, seqNo, streamID, delay, property)&lt;br&gt;select t, n, s, d, PExtract.compute((select * from RS), t – d, t – d – w) from AS</td>
</tr>
<tr>
<td>LW_psaF (l)</td>
<td>(* fn : INT * ‘a ASSERTION list → ‘b ASSERTION → ‘a ASSERTION *)&lt;br&gt;insert into PS(timestamp, seqNo, streamID, delay, property)&lt;br&gt;select t, n, s, d, PExtract.compute((select * from RS), n, n – l + 1) from AS</td>
</tr>
<tr>
<td>JoinTW_psaF(w1,w2)</td>
<td>(* fn : TIME * TIME → ‘a ASSERTION list → ‘a ASSERTION list → ‘b ASSERTION → ‘a ASSERTION *)&lt;br&gt;insert into PS(timestamp, seqNo, streamID, delay, property)&lt;br&gt;select t, n, s, d, PExtract.compute((select * from RS1), t – d, t – d – w1)</td>
</tr>
<tr>
<td>JoinLW_psaF(l1,l2)</td>
<td>(* fn : INT * INT → ‘a ASSERTION list → ‘a ASSERTION list → ‘b ASSERTION → ‘a ASSERTION *)&lt;br&gt;insert into PS(timestamp, seqNo, streamID, delay, property)&lt;br&gt;select t, n, s, d, PExtract.compute((select * from RS1), l1, t – d)</td>
</tr>
</tbody>
</table>
The main concept of the algorithm is that it utilizes stream topology information and stream operation parameters stored in a compact provenance store to automatically create the internal processing of on-the-fly provenance queries. To this end, it uses function composition to dynamically perform provenance tracking. The algorithm takes a stream of provenance assertions generated during a stream system's execution as an input and produces provenance tracking results, which are provenance assertions containing provenance-related properties. The algorithm for on-the-fly provenance tracking is presented in Figure 5.

As shown in Figure 5, the algorithm consists of two main functions: OTFquery and composePSAF. The OTFquery function is the entry-point function that takes a list of provenance assertions (paList), a list of target stream IDs (stream IDs used to terminate the query execution - sidList) as input parameters and returns provenance tracking results (resultList - a list of provenance assertions containing provenance-related properties). The other function - composePSAF - is the recursive function that contains the business logic of the algorithm. The process of the function begins by receiving parameters passed by the OTFquery function. For each provenance assertion (pa), it is first processed by an assertion separation unit (ASU) to create an extra field for storing property information. Then, in the case that input provenance assertions belong to the first-input stream - a stream that enters a stream system from data sources and are first processed by stream operations, they are executed by a property computing unit (PCU) in order to extract properties. After that, for each assertion, it will be processed by its associated PSAF (using the executePSAF function) and PCU. Every output of property computing (pa2 - an element of a result stream RS) is inserted to a data buffer (bufferList). Finally, the composePSAF function is called recursively until no provenance assertions remain in the provenance assertion list (paList).

To support the dynamic execution of on-the-fly provenance tracking, a number of intermediate provenance assertions needs to be temporarily stored in a data buffer. This processing technique potentially increases the amount of memory consumed by our stream provenance system. However, the size of the data buffer can be controlled by using Dequeue function. The function utilizes operation parameters - especially a data window size for each stream operation - to eliminate assertions that are no longer needed for further processing. Therefore, we expect the amount of memory consumed by our system to depend on the types of stream operations and the size of the data window utilized in a stream system. We discuss about the memory consumption for our provenance solution in the section 5.

5. Experimental evaluation

In this section, we evaluate the implementation of our on-the-fly provenance tracking mechanism. Our evaluation is conducted across three different aspects, including the provenance recording impact, the memory consumption for a provenance service and the time latency for on-the-fly provenance tracking. For all experiments, the experimental setup was as follows: Our provenance service and a stream processing system were hosted on a Linux PC with 1.60GHz Intel Xeon Quad Core CPU and 4 GB memory. To store auxiliary information (e.g. topology information and operation parameters) supporting the dynamic execution of on-the-fly provenance tracking, our implementation used MySQL 5 as a database backend. All our application components were implemented in Java. In addition, we use JMS as our messaging infrastructure with ActiveMQ 5 as the implementation choice.

5.1 Provenance recording impact

In this evaluation, system throughput - the number of stream events processed by a stream system over a given interval of time - is used as our performance indicator. In each set of experiments, we first measured the system throughput of a stream system that does not record provenance information. Then, we measured the system throughput of a stream system that records provenance information for different modes of the provenance service including 1) Just receive provenance
assertions (without any provenance processing), 2) Store provenance assertions into a provenance store and 3) Perform on-the-fly provenance queries. In addition, because different stream-based applications have different time delays for processing depending on the complexity of the computation performed, therefore three different time delays for stream processing - no delay, 1ms and 2ms - were considered as significant parameters in the experiments.

Figure 6(a) displays the system throughputs of our implementations with no time delays in processing, as a number of stream components increases. The figure shows a significant drop-off in the throughput of the implementation that does not record provenance information from the maximum throughput (23,000 messages/second). Similar but significantly lower trends in system throughput were observed for the other implementations that record provenance information. At the same number of stream components, the throughput decreases more than 50% for all implementations compared to the case of “no provenance” implementation. We determined that this degradation is due to the introduction of provenance recording functionality which doubles the number of data streams maintained by the message broker software.

Figure 6(b) and 6(c) demonstrate the system throughputs of our implementations that increase processing time delays (1ms and 2ms respectively). In Figure 6(b), all system throughputs significantly drop from that in “no processing delay” experiment (shown in Figure 6(a)) as expected due to the introduction of time delay 1ms. The system throughput of all our implementations gradually decreases when a number of stream components increases. In addition, as shown in Figure 6(c), there are almost no significant difference between the system throughputs of “no provenance” implementation and that of the on-the-fly provenance tracking implementation. The more the processing delay increases the more the processing overhead for provenance tracking is reduced. As a result, we conclude that the processing overheads caused by our provenance solution can be largely ignored when the time delay of a stream system is large.

Furthermore, considering the percentage of the processing overheads incurred by our provenance solution, the average processing overheads for the “no processing delay” experiment are excessively high - about 75% for the on-the-fly provenance approach. However, as the presence of processing time delays, the processing overheads are significantly reduced to be about 7%. Therefore, with the experimental results, we can establish that the impact of provenance recording is relatively small or more particularly it generally does not have a significant effect on the normal processing of stream processing systems.

5.2 Memory consumption for a provenance service
This evaluation aims to examine the effect of on-the-fly provenance processing on the normal processing of a provenance service, particularly with respect to memory consumption. In each set of experiments, we first measured the actual memory space used by the provenance service when provenance assertions are just collected but not processed. The result of this experiment is utilized as a performance baseline which indicates the normal amount of memory consumed. Then, we measured the actual memory space used by the provenance service operating the on-the-fly provenance query mode. To investigate the change of the memory consumption when a stream processing system is scaled up, the number of stream components was increased from 2 up to 10. Two different stream operations are considered: Map and Time-window operations.

The memory consumption of a provenance service can be estimated by using a straightforward mathematical formula. Table 3 lists the symbols used in our memory prediction formula. For on-the-fly provenance tracking, the amount of memory space consumed can be calculated as follows:

\[ MS = \text{Initial memory} + EP + (Q_{EP} \times AS) + (c \times w \times AS) \]

According to the memory prediction formula, the initial memory is defined as the amount of memory consumed by the provenance service when provenance assertions are just received but not processed. The last term - \((c \times w \times AS)\) - describes the memory size of assertions temporarily stored in the internal data buffers of a stream engine for all stream operations. In addition, we can further utilize the memory prediction formulas to predict the marginal cost of memory space consumed. In this context, the marginal cost is the amount of memory space required for adding an additional stream component to a stream system. The marginal cost is calculated by the following equations:

\[ MC = \frac{MS - MS \text{ (for a single stream component)}}{(c-1)} \]

Figures 7(a) and 7(b) present the actual memory space consumed by our provenance service for the map operations and the time window operations experiments respectively. The memory size for the on-the-fly provenance approach is slightly higher in Figure 7(a), and significantly higher in Figure 7(b) as the number of stream components increases, compared to that for the baseline (Just receive provenance assertions). We determine that this increase of memory size is due to the processing of on-the-fly provenance tracking that needs to store provenance assertions in internal memory. In addition, considering that when the type of operations is changed from the map operations (Figure 7(a)) to the time window operations (Figure 7(b)), the memory size consumed for the on-the-fly provenance approach goes up considerably. This can be explained, because, for the time window experiment, about 1000 provenance assertions are required to be stored in the memory for each internal assertion stream of the stream engine during execution time (the size of data window used is 1 seconds and the stream rate is an event per millisecond). When the number of stream components increases, the number of internal assertion streams increases as well. The greater the number of assertions needed to be stored in memory for processing, the greater the memory space consumed by our provenance solution.

Moreover, as shown in Figures 7(c), the overall trend for the marginal cost of memory space consumed remains stable for both the map operation and the time-window experiments as the number of components increases. The result also shows that the marginal cost is relatively low compared to the total memory consumption. For example, the average marginal cost in the map operation experiment is slightly less than 0.5 MB and that in the time window experiment is about 1.8 MB. Therefore, we conclude that the memory consumption for a provenance service may vary based on the types of stream operations and the size of data windows used in a stream system, but the marginal cost of memory space consumed for our provenance solution is relatively low and reasonable.

### 5.3 Latency for on-the-fly provenance tracking

We now examine the runtime overheads for our provenance solution by measuring the time latency for on-the-fly provenance tracking. In this context, the latency is defined as the time difference between a stream processing system produces a result and a corresponding provenance-related property is computed. In this experiment, three different chains of stream operations are considered: map operations, time-windows of size 10ms and time-windows of size 100ms. The reason behind the use of different operations is because for some operations (e.g. time windows), more than one properties have to be computed. This potentially results in delayed query results.

Figure 8 displays the average time latency of the implementation of our on-the-fly provenance solution. In the figure, as the number of components increases, the latency for the chain of map operations remains stable. On the other hand, the time latency for both chains of time-
window operations increases significantly. The explanation is that for windowed operations, internal data buffers of a stream engine are used to store intermediate provenance assertions during execution. This results in an increase of the average time latency 0.3 ms per additional component. Therefore, with a relatively low latency per additional component, we can establish that

Figure 8. Average latency for on-the-fly provenance query

our provenance solution offers low-latency processing and our solution can provide provenance tracking results in real-time.

6. Conclusion

In this paper, we have presented a novel on-the-fly provenance tracking mechanism, which performs provenance queries dynamically over streams of provenance assertions without requiring the assertions to be stored persistently. We have discussed the important characteristics of on-the-fly provenance tracking and also defined the specifications of property stream ancestor functions (PSAFs) designed specifically to work with on-the-fly provenance tracking. To demonstrate how our provenance mechanism works in practice, we have presented an algorithm for on-the-fly provenance tracking. The experimental evaluation demonstrated that our provenance solution enables the on-the-fly provenance tracking problem in stream processing systems to be addressed with acceptable performance and reasonable overheads. According to our experimental results, a 7% overhead is observed as the impact of provenance recording on system performance. Moreover, it is shown that our on-the-fly provenance approach offers low-latency processing (average latency: 0.3 ms per additional component) with reasonable memory consumption.

References


Watsawee Sansirimahachai is a member of Systems and Networking group (SysNet), in Computer Science Department, School of Science and Technology at University of the Thai Chamber of Commerce. His research interest is in the area of parallel and distributed processing with a special emphasis on data stream processing.

Luc Moreau is a Professor of Computer Science, in the Web and Internet Science group (WAIS), in School of Electronics and Computer Science (ECS) at the University of Southampton. His research interest is in provenance, and the broad area of distributed systems (grids, clouds, the Web, Web
Mark Weal is a member of the Web and Internet Science Group (WAIS), in School of Electronics and Computer Science (ECS) at the University of Southampton. His research interest is in the area of information systems, from hypermedia systems for eLearning through to information infrastructures for multi-user pervasive experiences.
A Job Hunting Matching System for University Students

Taku Jiromaru\textsuperscript{a}
Graduate School of Science & Engineering
Yamagata University, Japan

Tokuro Matsuo\textsuperscript{b}
Advanced Institute of Industrial Technology, Japan

Abstract

Many private universities have a problem on the students’ job hunting in Japan. When students get a job, they need to take the employment paper test. However, there are no effective support methods of job-hunting examination for low test-scored students. In this paper, we propose an education management support system for paper test in job-hunting. Our proposed system is used by instructor in the course. Each class contains multiple sets of lecture and exercise. The system searches for an appropriate exercise question for each student based on their knowledge and experiences. When students take an identical exercise question, traditional education methods have a problem like that students lose motivation to continue to learn because the exercise question is not fitted to each learner. On the other hands, using this system, we confirm that keeping students’ learning motivation and increasing students that want to study next time.

Keywords: Profiling, edge profiling, path profiling, performance, control flow graph.

1. Introduction

In this paper, we propose the system to effectively support out-sourcing job-hunting lecture in university. This system is the new LMS (Learning Management System) in order to enhance operating efficiency and an education effect which use computer.

In recent years, in Japan, companies carefully choose and select new college graduates. Companies often use a written-examination to select good people, while the college students cannot pass the written examination because their bases scholastic ability is dropped. Therefore the university's employment-rate maybe is decreasing for having students of low scholastic ability.

By the way, some private universities in Japan administrations are deteriorating year after year. Therefore, they have a limit of costs which spend except on regular lecture [1].

Consequently, we propose the new LMS to improve the problems effectively, and we explain it. This system is not using LMS in self-study but uses LMS in lecture, and it can supply appropriate question for student of individual. Moreover, offer of the question of the level optimal for a student is possible for this system. Thereby, the system purposes to increase self-efficacy of students, and keeping or increasing for student motivation of learning. Self-efficacy is said by Albert Bandura, it means “the belief in one’s capabilities to organize and execute the courses of action required to manage prospective situations.”[2]

2. PRELIMINARY CONSIDERATION

2.1 Employment support in higher education

In recent year, many companies think to hold down the cost of employment activity and employment, because stagnation and new college graduates are not aggressive in Japan. To employ them regularly, company may not teach business skill for them, because making employee’s age are not evenly in company. That makes disadvantage for company to see long-range.

Consequently, to employ new college graduates is a small number of employment and companies need the way to hold down the cost of employment method. Therefore, companies are occurring to fail applicants by written-examination. In recent year, a written-examination’s importance is increasing that many companies use Web-test before employment explanation meeting to select the college students.

Consequently, college students became a low-achievement makes low-employment-rate year after year. Fig.1 says that university-going-rate in Japan.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig1.png}
\caption{University-going-rate in Japan}
\end{figure}

\begin{quote}
\textsuperscript{1} 4-3-16 Jonan, Yonezawa City,Yamagata, 992-8510, Japan
jiro@om-edu.jp
\textsuperscript{2} 7-10-40 Higashihoi, Shinagawa-ku, Tokyo, 140-0011, Japan
dr@tokuro.net
\end{quote}
University-going-rate is increasing year after year, after 2005, Japanese high-school-students more one of half goes to university. Trow says university-going-rate over the 50% that is said “Universal-Access”, to keep college-student’s quality that is high-level is hardly at job-hunting at university of low-selection [3].

Many scholars are taking various researches to solve the problem. [5] However, the problem is not researches enough in Japan. [4] To increase university-going-rate makes low-academic students, thereby few universities take remedial-education by low-score students. [6] The contents of a course contain basis-achievement in elementary school and junior high school.

2.2 Existing Problem Solving

In order to conquer the above situation, to keep motivations for students and to realize the educational support for each student's level are necessity. Moreover, that educational support is desirable to hold down cost.

To thinks from the field of cost, to hold down the cost, using of e-learning which only use a student or active-learning which teaches and occurs are mentioned. These teaching methods are effective to students of having a self-regulated learning [7] [8]. Self-regulated learning is defined from” the learner in which the student controls his study positively when the student raises motivation and knowledge.” [9]. In this research, the student which does not have self-regulated learning is marked. Therefore, the above ways are not effective.

2.3 The reason of wrong answer or expression

Percentage of questions answered correctly is one of the way to decide difficulty of each question and IRT(Item Response theory), used percentage of questions answered correctly, is one of useful theory that appraise ability of each learner. IRT evaluation system has the potential for understanding each question’s difficulty. But, it is not useful for questions that are too difficult for each learner [10].

We forecast that there is more than one reason. So, we corrected the reason of wrong answers or expression to 15 people. They are university students. The way of correcting the reasons shows Fig 2. In Fig 2, A is teacher and B is learner.

One learner tries to solve problem, and one teacher watch learner’s action. If learner is wrong answer or expression, teacher tells answer or expression is wrong. And teacher asks learner why learner is wrong. At this time teacher see not to make learner to feel psychological pressure.

If learner has no idea, teacher advance an idea to learner. If learner has an idea, teacher and learner entertain that the idea is right. If both teacher and learner think that the idea is right, teacher records the idea.

This way has possibilities not to be able to find idea. In this research, we don’t have such case. But, if we have such case, more than one teacher entertains the idea.

Fig 2: Correcting reason of wrong answer or expression

We corrected 368 ideas, and identify 4 types of these ideas like table 1.

<table>
<thead>
<tr>
<th>Reason</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Lack of understanding</td>
<td></td>
</tr>
<tr>
<td>a) words</td>
<td>How to use “and”, “or” and “so”</td>
</tr>
<tr>
<td>b) number or sign</td>
<td>Mean of sign</td>
</tr>
<tr>
<td>c) mathematical formula</td>
<td>Mathematical explanation</td>
</tr>
<tr>
<td>d) concept</td>
<td>Understanding of Unit Metaphysical</td>
</tr>
<tr>
<td>2. Circumstances of learner</td>
<td></td>
</tr>
<tr>
<td>a) inner</td>
<td>Change in physiological phenomenon Physical sign</td>
</tr>
<tr>
<td>b) outer</td>
<td>Change in circumstance Small calculation sheet</td>
</tr>
<tr>
<td>3. Miss</td>
<td></td>
</tr>
<tr>
<td>a) Miscalculation</td>
<td>Miss by mental calculation</td>
</tr>
<tr>
<td>b) Copy miss</td>
<td>Handwriting is a mess</td>
</tr>
<tr>
<td>c) Cognitive distortions</td>
<td>Meaning shift while solving the problem</td>
</tr>
<tr>
<td>4. Lack of logical thinking</td>
<td></td>
</tr>
<tr>
<td>a) Deduction</td>
<td>Logic leap Order of logic confusion</td>
</tr>
<tr>
<td>b) Induction</td>
<td>Duplication and leakage Mix up deduction with induction</td>
</tr>
</tbody>
</table>
Average of a number of wrong answer or expression per question per each learner is 3.4. Therefore, we need to set a rate of difficulty for each wrong reason. But, circumstances of learner don’t have any relationship with difficulty of each question. So, we need to set a rate of difficulty for “Understanding”, “Miss” and “Logical thinking”.

3. Difficulty matching system to University students for job hunting

This system is designed for the purpose of employment support. This system’s candidate is a student of low academic ability. Students do not perform operation of this system but a leader operates it in principle. It is a print which a student gains and a system chooses a print automatically based on a student's learning information.

To the first time, the leader inputs student information into LMS. As student information, a student besides personal information, such as a name and age, inputs the results of the achievement test and the applicable lecture which took an examination before. Moreover, the result which the student answered is read in the print which the student submitted.

Although the print in which the exercise problem which suited each student's level from student information and the answer result of a student's exercise problem was indicated is printed, in a first-time print output, it selects only from the data inputted as student information.

3.1 Lesson process

A lesson is dealt with for 5 minutes and it deals with a total of 15 10-minute minutes for an exercise as one unit, and if the process of the lesson which uses this system is a lesson, it will perform this unit 4-5 times for 90 minutes.

A leader explains first the point which solves a problem over 5 minutes. The result which this system evaluated by the front unit and the student evaluated by self by the time explanation finished is read, and the exercise problem which analyzed the result and was suitable for each student is printed. After the end of printing of an exercise problem, this system performs printing of description and an additional problem. After a leader's explanation, a student solves applying an exercise problem for 5 minutes, and performs grading and evaluation in the remaining 5 minutes. A student carries out with the print of description of grading and evaluation, and each student writes down the result in an answer sheet, and hands a leader.

3.2 Contents recorded on LMS

A student replies to the question of two items in answer sheet form, after doing self-grading. One is a question about grading scale of 1 to 5. Another is a question about the difficulty in question scale of 1 to 5.

Although a student does self-diagnosis to an essential question and replies to it, an input is corrected, when a instructor checks input data and differs from the fact behind. If a leader has time, he will do correction work into a lecture, but by question correspondence etc., when there is no time, it checks out of lecture time.

3.3 Creation of a problem and a hint

Two kinds of questions are indicated on the print of an exercise questions which all the members solve, and an additional question. An additional question is a question for the student who finished solving an exercise question before the end of exercise time to solve. Moreover, on the print which indicated the answer, the additional question for the students who solved the answer and description of a basic question and an additional question and grading, an additional question early is indicated. The leader creates the question and the hint in advance.

3.4 Selection of a question and a hint

In selection of a question and a hint, it characterizes by a subject, a field, a unit, and a keyword in question beforehand. At this time, all the questions are associated as a network of a graph base like Fig 4 to each name.

The difficulty between questions is evaluated and given to the edge of the graph for each wrong reason. And edge has 3 weight, Understanding, Miss and Logical thinking. For example, the edge between learning 1 and learning 5 has the numbers that No.1 is 5, No.2 is 4 and No.3 is 3.

Fig 3: the model of THIS SYSTEM.
The selection method of a question or a hint in a basic question or an additional question uses a Bayesian network recursively. The example is shown in Fig 5.

Fig 5 is the length of the edge of the learning 1 and the learning 2, in 7, the length of the edge of 2, the learning 2, and the learning 3 has set to 3.

4. SYSTEM TRYING

About this system developed by this research, in order to verify the effect, it tried in the following way by dividing into A group and each B group a total of 20 junior year 10 men and 10 women who are subjects. In addition, a lecture was taken with this written examination measure lecture (1 top 90 minutes) of the former them before 6 tops.

4.1 Procedure of system use

It is 5 about today's contents of a lesson. Orientation about a part was performed. The 1st questionnaire about the degree of acquisition of today's contents of a lesson was carried out. The questionnaire entries of a questionnaire were 5 which the question number of Table 4 shows from 1 to 5. Although it was a question, the reply was taken as five steps of the following values. (5: I think very much so. 4: I think so. 3: It is not which, either. 2: I seldom consider. 1: I do not consider at all.) The lecture which uses this system was performed in A group. On the other hand, the lecture as conventional was performed to B group. The 2nd questionnaire survey was conducted on all subjects. The questionnaire used the formula like the same question as the 1st time.

4.2 Result

Table 2 is the result of totaling a questionnaire.

The total result for every question of five questions was made into the sequence of a lengthwise direction, and various numerical values were made into the line.

In the table, "Use this system" is the average value of a reply of A group, and "Not use this system" has pointed out the average value of the reply of B group. Moreover, "the difference of existence" is the value which subbed average of "Not use this system" from average of "Use this system". Moreover, it is the 1st questionnaire the questionnaire survey was conducted before the lecture "before execution", and "after execution" is a total result of the 2nd questionnaire the questionnaire survey was conducted after the lecture. "The difference execution before and after execution" pulls the value before execution from each value after execution.

The following was found when it analyzed what kind of thing has happened to the student from the total result.

1) The self-efficacy that the question was able to be solved increased.

For a not much diffident student, self-efficacy is very important feeling in order to make study continue. It is expected that continuation of study become easy more because self-efficacy increases from the usual lesson.
2) Increased the students who continue wanting to receive the same lecture.

Since this lecture assumes the lesson which a unit does not generate, whether student would continue liking to receive a lecture is an important point which can also serve as importance of existence of a lecture.

By this questionnaire, the rate that the students of A group which uses this system continue wanting to receive a lecture from the student of B group intentionally increased.

3) Change of the confidence attendance before and after attendance.

It turns out that the students whom B group which did not use this regards as the ability of the question concerned to be solved from attendance before of a lecture from the result of the question number 2 of Table 2 are decreasing in number.

If the student of B group is interviewed later, there is no clear backing "I thought that it could solve somehow before attendance" to the confidence which solves the question of the field concerned, confidence would not be lost but it will be surmised that it was false achievement thinking.

This is conjectured to be the result of producing in order not to solve a question by lecture attendance. False achievement is recognizing it as the ability to solve although the question concerned originally cannot be solved [12].

There was no change in the rate thought that the student of A group which uses this system can solve a question on the other hand. Although it is surmised that there were some students who had a false achievement intention like B group, it is surmised that the false achievement changed to confidence.

5. CONCLUSIONS

In this research, this system which is a system for the college student who goes to the university where selection nature is low to gain the academic ability for clearing an employment written examination was proposed. This system is a new study managerial system which can develop the lesson suitable for each student's needs as a method which held down cost according to the characteristic of the student concerned.

The system was tried in order to verify the effect of this system. In this trial, although there were few subjects and they were moreover execution of only one top, the student of the group which used this system from this questionnaire result who continues wanting to take a lecture on the increase of self-efficacy and a lesson increased, and not having lost confidence was checked.

After this research, we research more accurate information about difficulty rate of each question. In this research, difficulty rate of each question decide us. We will research mathematical model for decision of difficulty rate of each question.

References

Biography

Taku Jiromaru is a Ph.D. candidate of Graduate School of Science and Engineering, Yamagata University. He also founded and served CEO of OME Inc. and ConferenceService Inc, since 2009 and 2011, respectively. He is a member of JSET, CIEC, JPCATS, and several other academic societies. His current research interest areas include Educational Technology, Mathematical Education and Career Service System.

Dr. Tokuro Matsuo is, currently, Professor at Advanced Institute of Industrial Technology, Japan; Research Project Professor at Nagoya Institute of Technology, Japan; and Research Fellow of SEITI, Central Michigan University, USA. He is also commissioned as an adviser of information promotion from Japan Government since 2008; and chairman of tourism informatics research committee in Institute of Electrical Engineering of Japan. He received his Ph.D. in information engineering from Nagoya Institute of Technology in 2006.

He has served a lot of conference/program/workshop chairs including ACIS SNPD 2009, 2012 and 2013; CAINE 2012; ACIS SSNE 2011; IEEE/ACIS ICIS 2010 and 2013; IEEE IWEA 2007-2011; IEEE CEC 2011 and 2012; IEEE PRIWEC 2006; IEEE/ACM/WIC ECBS 2008 and 2009; ACAN 2005-2012; and other 20 academic events. He has received over totally 40 awards and research grants from international conference, research foundations, companies and government. He has ever published over 300 research papers and articles in this decade. He is a member of IEEE, ACIS, IIAI, IEEJ, and several other academic societies.

He is current research interest areas include Tourism Informatics, Material Informatics, Safety Service for Disaster Recovery, Designs on Secure Electronic Commerce Systems and e-Auction Protocols, Career Service System, Qualitative Reasoning and Simulations. Some of his researches are presented in several top ranked international conferences including AAAI, IEEE CEC, AAMAS, and WWW.
Actionable MOOCs through Course Certifying Agency Framework
Yeong-Tae Song
Towson University
Chris Connolly
Towson University
Yongik Yoon
Sookmyung University

Abstract

The advent of the Massive Open Online Course (MOOC) struck e-learning community by surprise. More universities are willing to offer their top-notch professor’s courses as MOOC. However, when utilizing the knowledge from MOOCs, learners need to go through a number of hurdles – getting course completion certification from various providers, validating the knowledge level of a MOOC in the related domain, and make it searchable for various purposes.

We propose a framework, a course certifying agency framework, which merges the learner’s profiles from various providers. This way MOOC completion information is stored in one place. We utilized web ontology language (OWL) to relate MOOC course profiles, learner profiles, learning goals, and required skill sets. It gives us the ability to match qualified learner profiles for a job position and/or to identify a set of related MOOC course profiles for some learning goal. The employer looks for a matching skill set from converged learner profiles. Each skill for the position goes through a mapping procedure with a corresponding MOOC course profile. After mapping skills to corresponding MOOCs, the framework searches for the converged profiles. The result is the list of learners who match or almost match a given job description.

Keywords: e-Learning, Massive Open Online Course, Learner profile, Course Profile, Ontology, Certifying MOOC, Semantic search

1. Introduction

The introduction of MOOC providers has introduced collections of openly-accessible knowledge from many traditional universities. Since its introduction, more than 800 openly-accessible and college-level courses have emerged from course providers such as Coursera, Udacity, and edX [21].

MOOCs are beginning to become highly visible and endorsed by many major top-tier universities. The American Council on Education, an organization that advises college presidents on policy, has gone so far as to endorse five MOOCs from Coursera for credit and is currently reviewing more from Udacity [13]. No longer are MOOCs plagued by the stigma that they no longer offer the quality of education provided by typical university classroom settings.

However, despite the increase in course quality now being seen in MOOCs, these courses offer knowledge in a vast variety of formats and learning pedagogies [3]. Learning material pertinent to a learner’s professional development often resides in a variety of locations and is highly disorganized [3]. With learners no longer engaging in traditional learning pedagogies and the weight of accomplishments in the e-learning community no longer easily discerned, several problems have emerged that our approach aims to solve.

There are three problems currently plaguing MOOC programs:

a) Learners desire to retrieve available courses relevant to their own learning goals potentially in a proper sequence to best achieve their learning goals from a distributed set of MOOC providers and their courses

b) There is no way currently to track learners’ accumulation of knowledge from different MOOC providers under a single authoritative agency who will facilitate management of this information

c) There is no agreement among MOOC providers on the approaches or standards to facilitate managing this information

The knowledge to be gained from completion of these courses represents viable skill sets that may go completely unrecognized once these learners enter the job market. Udacity, Coursera, and edX represent 400 thousand, 3 million, and 350 thousand students respectively, having access a to knowledge base of their skill sets could benefit both employers and the students themselves [21].

Ideally, a system should exist that will allow learners to retrieve courses from a variety of MOOC providers allowing them certifiable proof that the knowledge gained from courses administered by MOOC providers maps to specific skill sets that an employer desires in their potential employees.

Our approach aims to build a framework that will allow for learners to easily retrieve courses relevant to their specific learning goals in a proper sequence that best suits their learning preferences. The information about successfully complete courses from all MOOC providers will be tracked by an over-arching authority and be searchable by employers looking for employees to fill available job descriptions that require the skill sets taught by these completed courses.

The rest of the paper will discuss the components of this framework listed as follows:
Section 2 discusses related work and applicable standards for learner profile and course profile.

Section 3 illustrates proposed Course Certifying Agency (CCA) and constituting components.

Section 4 shows how each component is utilized in some typical scenarios.

Finally section 5 provides conclusion.

2. Relevant Works

MOOC stands for Massive Open Online Course and is an educational delivery method that is gaining acceptance in academic circles as an alternative to the traditional instructor led, classroom delivery method. These online courses typically involve videos of lectures combined with interactive assessments while encouraging student collaboration and use of social networking applications. MOOCs combine the connectivity of social networking with the facilitation of an expert in an online, resource rich, environment [18]. The focus on the connectivity of these courses necessitates their scalability and in this case massive truly means massive with enrollments reaching the thousands in a single offering [18, 22]. One important benefit of these courses is that, while they are many times taught by world renowned professors and at highly esteemed institutions, they are typically free of charge. Like any recent innovation, MOOCs are evolving rapidly to suit the needs of both providers and students. As such, MOOC’s final form and value has yet to be determined, but the opportunity presented by the MOOC format is attracting a great deal of attention [8].

There are many reasons that the popularity of MOOCs is increasing. With the cost of education rising faster than that of healthcare, MOOC has advantages for students in that they are available anywhere and at no charge [9, 17]. The only prerequisite for the courses is an Internet connection and interest in learning the material. Many students look upon these classes as a way to see if they are interested in a subject without having to pay [28]. If the course is too difficult, or they find they are not enthusiastic about the material, they simply stop attending. The student does not have to worry about a failing grade or an incomplete and is not bound by a financial investment in the course that will be lost by not completing it. It is also important to note that many MOOCs do not list a set of outcomes for qualification of success; students who drop out of the course may have achieved the educational goal they were pursuing by acquiring the desired knowledge. In this case, success or failure of the course is determined by the student’s goal and not a syllabus. Thus, a reason for the increasing popularity of the MOOC format is that they present low risk to the student and therefore learning can take place at even a modest interest level [4]. MOOCs are also taught in small chunked lessons that include information and assessments. This chunking of information creates a fast turnaround time between learning a concept and performing the assessment activity [23]. Classroom based courses often feature an hour long lecture and the students are sent home to practice the skills they learned. With the MOOC format the professor typically lectures for a short time then the students perform an assessment, followed by another short lecture and assessment. Assessment activities are designed to encourage learners to be socially active and to pose questions, work through problems, and discuss class topics using social media. This social learning lets the students support and learn from one another.

For the providing institution, they have the ability to attract large numbers of students and crossing national borders. This widespread exposure helps to increase the institution’s global awareness and notoriety. Some professors and institutions have recognized that MOOC is an opportunity to take their class worldwide and attract students that would be otherwise unavailable due to geographic or financial factors [1]. Institutions are also using MOOCs as a means of attracting students in the fields of engineering and computer science to their programs [29].

While this growth in popularity is demonstrative of the opportunity that MOOCs may represent, many academics still doubt the effectiveness of the MOOC format [17, 26]. They point out several weaknesses, one of which is the logistics and preparation involved in teaching a class of thousands. Some professors have stated that preparing for a MOOC becomes a fulltime job by itself. Personalized attention is not feasible when classes get that large and even though they spent many hours preparing many professors feel that the students are being short changed by the class [16, 26]. Moreover, students learn in very different ways and not all learning styles can be accommodated in a MOOC [23]. Complex concepts are another problem in that they are difficult to convey even when working in small groups. Some professors who have taught MOOCs have stated that they felt the need to decrease the rigor of the material [28]. The largest recognized evidence that the MOOCs are not as effective as traditional classes is the high percentage of students who do not complete the course [12]. Many professors see this large drop in participation as evidence that MOOCs are not as effective as a traditional classroom based courses [18]. The level of attrition is higher than that of traditional classes and had been shown to be as much as 80%. In one class, 150 thousand students registered and only 20 thousand finished. While 20 thousand students completing an offering of a single class is impressive it is still a decline of roughly 87 percent.

Most professors recognize that MOOCs are another tool that can be used to convey information; but also that the MOOC delivery format still needs refinement [4, 8, 23]. Some identified areas where improvement would be beneficial are assessment and certification. Providers and
instructors have moved quickly to try to address these weaknesses. Assessments have been improved and even automated; classes taught as MOOCs have had their curriculum aligned with the classroom version in the hope of making the classes comparable [2]. Recent studies have shown students who complete these MOOC courses have no significant difference in later performance than students who complete traditional courses [29]. In one study, students who completed their introductory computer science course through a MOOC were shown to pass more and fail fewer classes than those who completed the class in a classroom based setting [28]. Certification and accreditation of MOOCs could be considered the next step in their evolution. Although, early MOOCs were non-credit or certification courses, many providers including Udacity and Coursera have recognized the need for a verification of completion of the course and are even offering levels of competency such as “Highest Distinction”. Thus, providers are able to certify student achievement. Recently, the American Council on Education has decided that they will review 5-10 MOOCs and decide if they should be included in the council’s College Credit Recommendation Service. This ACE credit certifies non-credit training courses as equivalent of certain comparable college classes [31]. Many professors who either have just finished or are in the process of teaching their first MOOCs have echoed this sentiment and stated in a recent survey that some MOOCS should be counted as regular classes for credits [31].

3. Learner Profile Model

Currently there is no standardized learner profile format for MOOC providers so it is difficult to combine the content of the learner information and their completed course information from various MOOC providers. There are two competing standards in industry – IMS LIP and IEEE PAPI. In our approach, we mainly followed IMS LIP and its XML schema format to represent the converged learner profile and course completion information. In IMS Learner Information Profile, there are 11 core structures to describe a learner such as identifications, security keys, transcripts, goals, qcl (qualifications, certifications and license), activities, interest, competency, relationship, affiliation and accessibility [11]. We have added course related attributes to represent course completion information that should come from various MOOC providers’ sites:

1. Identification (Full Name, Email Address), 2. Learning Goal, 3. Qualification, Certification, and Licenses (QCL), 4. Activity (MOOC Completion Status)
   a. Course Titles / Course ID or Code (if any)  
   //this part may have multiple entries depending on the number of completed MOOCs //
   i. Category, ii. Completion Status, iii. Date of completion, iv. Certification Status (Type of Certificate such as highest distinction) v. Course content provider, vi. Organization’s Name (MOOC Providers)

The main purpose of conforming to industry standard such as IMS LIP for the learner profile is to make the system interoperable with other systems e.g. converging learner information from various MOOC providers’ sites. IMS LIP specification also defines a set of packages in XML that can be used to import data into and extract data from IMS compliant learning management systems [10]. A partial screenshot of the XML learner profile is shown in the Figure 1.

![Figure 1: A Part of Converged Learner Profile in XML Format](image1)

![Figure 2: The Dublin Core](image2)

2. Course Profile Model

In our proposed approach, we have followed an industry metadata standard - Dublin Core (DC) to represent the course content elements for the course profile. The main
purpose of using Dublin Core is to create simple descriptive records for all the MOOC courses. DC elements describe the resources in the networked environment in an effective and interoperable way. It has fifteen "core" elements as shown in Figure 2 [6]. We have used the following elements to express the course profile contents: title, description, creator, creator organization, type, audience, publisher, identifier, format, typical learning time, workload, category, difficulty level, language and cost. The only added element is ‘cost’ that we used to describe the price value of the courses. Figure 3 shows our Course Profile in XML format.

Learners can utilize the course attributes in the RDF graph in finding relevant MOOC courses to their learning goals. It can be done by using the Learning Goal Mapper that maps learner’s learning goal into a list of relevant MOOC courses. The course completion information and learner’s course completion information may be useful for potential employers who are looking for employees with some desired skill set. The desired skill set may be interpreted into a set of relevant MOOC courses by the Skill set – Course mapper. The resulting MOOC courses will be utilized in searching a learner’s profile that has corresponding MOOC course completion information, which can be done by the Learner Profile mapper.

3. Proposed Framework Components

The CCA framework consists of various software module components and persistent data storage. Our persistent data storage options include OpenLink Virtuoso Universal Server for our semantic applications. The course profiling module continuously monitors newly available MOOCs and populates course profile attributes in the Virtuoso server. Merging Learner profile module will update a learner profile when new course completion information for the learner is available from a MOOC provider. The CCA will have various mapping modules that will convert gathered data from distributed MOOC providers into both triple stores as part of RDF graphs via the Merging Learner Profile module and the Course Profiling module. The converged learner profile may be queried by the learners to check their converged profile for their converged MOOC completion information.

Figure 3: Course Profile in XML Format

![XML representation of the course profile](image-url)

Figure 4: CCA Architecture

### 3.1. Virtuoso RDF Triple Store

The central component used for the distributed learner profile will be an RDF store running an instance of OpenLink Virtuoso Universal Server [30]. The Virtuoso RDF Triple Store will house RDF graphs for two ontologies that are direct OWL representations of our Course Profile and Learning Profile models described in section 2 [14]. Using the SPARQL query engine that is built into the OpenLink Virtuoso RDF Triple Store server, we will be able to query the graph for relationships from various MOOC providers to find the courses most relevant to the learner’s learning goals or to query learners’ profiles for finding best matches with desired skill sets [7, 15, 24]. Additionally, using the built-in OWL inference properties that are possible with Virtuoso, we can expand our queries to result in more related resources in certain ways other than key words. In this case, RDFS and OWL object properties may be included for consideration in the expanded query [19, 5]. For the following examples, URIs will be prefixed per W3C guidelines for the purpose of readability. The two graphs representing our ontologies that model the Learner Profile and Course Profile will be prefixed as “CP:” and “LP:”, respectively. “CP:” is a shortened reference name for a URI with a Universal Resource Identifier (URI) such as <http://cp.towson.edu/CourseProfile/CourseProfile.owl>. For the given ontology, properties can be created to describe the relationships between the courses such as CP:isPrerequisiteOf and CP:isCorequisiteOf or their inverse properties CP:hasPrerequisite and CP:hasCorequisite, respectively, for a graph prefixed per W3C guidelines as CP [25].

### 3.2. dotnetRDF
Among many other APIs for RDF data sets, dotnetRDF (http://www.dotnetrdf.org/) was chosen for the manipulation of the proposed data sets. It is an open source .Net API that allows for manipulation of our RDF data sets in a programmatic way. Additionally, the API allows for the parameterized construction of SPARQL queries against the Virtuoso RDF Triple Store [27].

3.3. Course Profiler

A Course Profiler component will perform all duties pertaining to the maintenance of the course information made available from MOOC providers online and converged within the Course Certifying Agency.

3.3.1. MOOC Course List Monitoring

MOOC providers will be continually monitored via several methods to ensure that the data available in RDF stores accessible via semantic search are up-to-date.

3.3.1.1. Access of MOOC Provider Course Listings via RESTful API

As MOOC providers continue to engage in trending web technologies, it is possible to access course listing for easy integration of our CCA platform via a RESTful API. As an example, the Twitter API documentation provides a RESTful web method “GET statuses/retweets/:id”. This method allows users with authentication tokens to substitute an appropriate ID for a given tweet and have up to the first 100 retweets of said tweet returned via JSON format [25]. If such an API were available from MOOC providers, the returned data in JSON format could be packaged via a data manipulation factory class and converted with dotnetRDF into triples nodes representing the subject, predicate, and object of a triples statement and submit it to Virtuoso for insertion into the RDF triple store.

3.3.1.2. Parsing HTML of MOOC Course Search

With RESTful APIs seemingly non-existent for MOOC providers at the time of this writing, one approach for mining the course data of MOOC provider sites uses an age old method of parsing TTPWebRequests to perform actions against a web site as if it were being performed from a browser. Once this request has been returned to the agent that has spawned the request, the results are handled and parsed.

3.4. Course Profile Mapper

3.4.1. Mapping to Triples Format

The Course Mapper component will interpret the raw data gathered either via JSON or HTML parsing and package the data into RDF triples representing the abbreviated list of Course Profile properties defined in Table 1 below

<table>
<thead>
<tr>
<th>IEEE LOM/DC Standards Elements</th>
<th>MOOC Courses Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Course name with IDs / codes</td>
</tr>
<tr>
<td>Description</td>
<td>Class summary / course overview</td>
</tr>
<tr>
<td>Creator</td>
<td>Instructor brief information</td>
</tr>
<tr>
<td>Creator Organization</td>
<td>Academic institution</td>
</tr>
<tr>
<td>Publisher</td>
<td>MOOC provider name</td>
</tr>
</tbody>
</table>

Table 1. Course Profile Structure

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="">CP:1</a></td>
<td><a href="">dc:title</a></td>
<td>&lt;Agile Methodology&gt;</td>
</tr>
<tr>
<td><a href="">CP:1</a></td>
<td><a href="">dc:creator</a></td>
<td>&lt;Ron Smith&gt;</td>
</tr>
<tr>
<td><a href="">CP:1</a></td>
<td><a href="">dc:description</a></td>
<td>&lt;An introduction to Agile Methodology. &gt;</td>
</tr>
<tr>
<td><a href="">CP:1</a></td>
<td><a href="">dc:creatorOrganization</a></td>
<td>&lt;Harvard&gt;</td>
</tr>
<tr>
<td><a href="">CP:1</a></td>
<td><a href="">dc:publisher</a></td>
<td>&lt;Coursera&gt;</td>
</tr>
</tbody>
</table>

Table 2: Course Profile Structure (partial) in RDF Triples Format

The Course Profile structure in the table 2 is an example of the triple subject-predicate-object statement we would use to define a course listing. In this example a course listing is characterized with a unique identifying integer value that is independent of the MOOC that it originated from in our converged profile system.

As can be seen, the predicates of these course profile triple statements refer directly to the Dublin Core RDF specification indicated by the prefix “dc:” which is represented in whole by the URI <http://purl.org/dc/elements/1.1/>. This is done to remain in accordance with our standards declared in section 2.

3.4.2. Insert Mapped RDF Triples in Triple Store

dotnetRDF has the ability to designate a set of triples as part of a specific graph. In this instance, as mentioned earlier, the triples mapped using the course mapper will be placed in the CP graph. Figure 5 displays some example code displaying the direct manipulation of triples in a specific graph using the dotnetRDF API.
Figure 5: Code Example Utilizing dotnetRDF API

```csharp
PREFIX CP: <http://cp.towson.edu/cp/courseprofile.owl>
PREFIX LP: <http://lp.towson.edu/lp/learnerprofile.owl>

PREFIX CP:
PREFIX LP:

SELECT ?learner_email ?course_id
WHERE {
  ?course_id CP:hasPreRequisite CP:Software_Engineering;
  LP:hasLearner ?learner_email;
  LP:hasTakenCourse CP:272 .
}
```

Figure 6: Example SPARQL Query for Proposed SPARQL Endpoint

The ontology for Course Profile is defined by owl using Protégé as shown in the Figure 7 [20].

3.5. Learner Profiler

Learner Profiler component will perform maintenance regarding the learner’s demographics as well as their completed course information for persistent storage from various MOOC providers. Whenever a learner receives a MOOC completion certificate from a MOOC provider, the profiler updates the Learner Profile so all MOOC completion information may be searched in one place regardless of their providers.

3.6. Learner Profile Mapper

Learner Profile Mapper is a software module that is used when an employer’s request comes in to find a learner profile that matches with a certain desired skill set for the position. Each skill in the set will go through a mapping process to find a matching MOOC completion record or other related QCL history. The mapping process will have two steps – one for finding match MOOC and the other for searching learner profiles for the identified MOOC or QCL. Any matching over 80% considered relevant for the position. The selected learner profiles need to go through filtering process where location, desired matching level, and any other factors that are imposed by the employer will be utilized. The final result can be exported into serialized XML in the format shown in the Figure 1 for direct delivery to potential employers for the final selection.

3.7. CCA Component Summary

Utilizing the components of the framework described above, the CCA can perform the following operations:

1. Provide semantic search for all available MOOC provider courses based on learner’s learning goal. All MOOCs must be registered in the CCA before use. All object properties need to be set during the registration
process for intelligent semantic searches – take advantage of all defined relations among MOOCs.

2. Provide semantic search for employers seeking employees with desired skillsets based on their record of completed MOOC and other QCL. Selected profiles can be delivered to the employers in a serialized XML format.

3. Provide learners the ability to store learner profile information and completed courses from a variety of distributed MOOC providers under one validating, certifying umbrella

4. Continually update the course profile listings by monitoring a variety of MOOC provider’s search sites either via RESTful API techniques or via other crawling techniques

4. Using CCA

4.1. Setting up Learning Goal

Each learner is encouraged to set up their own learning goals. Each learning goal is interpreted by the CCA and produces a set of MOOCs that help achieve the learning goal. For that, when a course is registered, it is required to set up relationship with other MOOCs such as “superseded by”, “isSubconceptOf”, “isGeneralizationOf”, or “prerequisite of”. Newly identified relationship between MOOCs may be easily added because of the properties – URI - in RDF triple. So when a learning goal is submitted, the course mapper as shown in the Figure 4 is executed to extract the related keywords for the goal. With resulting keywords, it will execute a SPARQL query with the intention to find all related MOOCs. Once the learner has received a list of MOOCs, s/he will go through filtering for removing unnecessary MOOCs and ordering for the remaining MOOCs for proper sequence.

4.2. Searching for matching learner profile

When needed, employers may request the CCA for the matching learner profile for their advertised vacant position. They may submit a desired skill set(s) for the position(s). The CCA returns the list of learner profiles that match at least 80% of the desired skill set. The requestor may go through the filtering process for additional criteria and finalize their selections.

5. Conclusion

MOOC is becoming a more prevalent trend in postsecondary education and could be future generations’ solution for gaining a quality professional education at a low or no cost. MOOCs could function as the bridge between academia and industry allowing learners to customize their skill sets and aptitudes to what prospective employers are searching for. While MOOCs have proved themselves in some scenarios as a valid, credit-worthy endeavor in the eyes of potential employers or even educational councils, there is a need to establish verifiable quality assurance process and also the certification process for MOOC completion [13] so it may become valuable asset for all MOOC learners.

The Certifying Course Agency is an approach that imposes add-on values to MOOC courses so they may be recognized by potential employers as well as by entire learning community. It also facilitates and encourages the standardization of profiles of both courses and learners across distributed MOOC providers. The Certifying Course Agency has proposed current and future methodologies for the aggregation of both Course Listing data and Learner Profile data. With this aggregated data, learners have the potential to search using powerful semantic tools and orient themselves on a path toward their learning goals across various MOOC providers. Going further, the CCA also provides learners and employers a middle-ground to best serve one another both in building better workforces and attaining more fulfilling careers.

Acknowledgments

This research work (Grants No. 00049410111) was supported by Business for Cooperative R&D between Industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2011.

References

Dr. Yeong-Tae Song received his master’s and doctoral in computer science from the University of Texas at Dallas in 1993, 1999 respectively. He has been PIs for numerous research projects including NASA, state of Maryland, and various industry partners. His research areas have been software engineering, enterprise architecture, e-learning, and medical informatics. He has published about 50 publications including conference papers and journal papers. He is currently an associate professor at Towson University.

Dr. Yongik Yoon received his doctorate in computer science from KAIST in 1994. He has been chairing for various academic organizations and PIs for various research projects including IPTV, smart cloud, and smart TV. His research areas have been middleware, mobile systems, and multimedia systems. He is currently a professor and chair person of the department of multimedia science at Sookmyung University in Korea.

Chris Connolly is a manager of a training program for the US Government. He is also a doctoral candidate at Towson University. He has investigated issues in accessibility, web and software engineering, and in human/computer interaction. He has presented papers at the SIGITE, HCI and SITE conferences. In 2012 his presented paper was rated in the top 10% of those presented at the SITE conference in Austin, Texas.
Measurement and Development Cost Implications of Component Commonality in Software Product Line Engineering

Hamad Alsawalqah
Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea

Sungwon Kang
Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea

Bashar Al-Shboul
The University of Jordan, Amman, Jordan

Jihyun Lee
Daejeon University, Daejeon, South Korea

Abstract

Nowadays, software firms are applying the Software Product Line Engineering (SPLE) approach for its concepts of commonality and variability to provide product variety in a cost-effective manner. Yet, the effect of different amounts of component commonality on the perceived benefits from adopting SPLE is not well understood. Measuring commonality marks the starting point of precise analysis and better understanding of this relationship. In this paper, analytical tools developed in manufacturing are analyzed and adopted (i.e. Commonality Indices (CI)) in order to measure the commonality in SPLE. We used a case study of a Door Lock Control System (DLCS) for evaluating different levels of component commonality. To map the component commonality to the resulting development cost, linear regression is used to assess the relationship between the various commonality indices and the resulting development cost. This study suggests that reusing commonality indices from the manufacturing domain to assess the commonality of products in software product lines is sensible. The linear regression analysis shows that the relationship between those indices and development cost is verifiable, with high $R^2$ value ranges between 0.81 and 0.95, and significant, at the 0.025 ~ 0.005 level.

Keywords: Commonality, component, cost estimation, metrics/measurement.

1. Introduction

The Software Product Line Engineering (SPLE) is a current software development paradigm that results from the systematic combination of mass customization and the use of a common platform for the development of software products and software-intensive systems [1].

SPLE approach has been well recognized as an effective means to reduce the development cost, time to market, and helps to improve the quality [1, 2, 3]. Product family and product platform design is a strategy to facilitate simultaneous design of multiple distinct products based on a set of common features, subsystems, components, interfaces, and processes. A product platform is the common basis of all individual products within a product family from which these products can be derived.

The benefits of commonality, such as reduction of development costs, reduction of time to market, and enhancement of quality, are widely known by software companies. Yet, it is still not fully understood in the sense of its quantification and correlations to those benefits. Consequently, determining the extent to which to use component commonality that is the amount of component sharing among the products becomes more difficult. As a result; commonality is not always correctly implemented when developing new products making many companies losing some of its benefits rather than taking full advantage of it. One reason is the lack of appropriate methods and useful analytical measures (i.e. metrics) to assess the software product family based on commonality. Consequently, no real attempts are made to correlate component commonality to some of related quantifiable benefits (i.e. the reduction of development cost). The development of such analytical measures is a prerequisite to understand the relationships between commonality and its benefits.

To handle this apparent discrepancy, we present a set of commonality indices for measuring component commonality among a family of software products with the goal of assessing the ability of a product family to reduce the development cost. In order to do that, firstly, we analyzed and adopted existing manufacturing commonality indices for measuring commonality among family of software products. Secondly, we performed a linear-regression analysis study to validate those indices and to assess the correlation of those indices to the development cost.

---

1Department of Information and Communication Engineering Daejeon, 305-701, South Korea ellissar@kaist.ac.kr
2Department of Information and Communication Engineering Daejeon, 305-701, South Korea sungwon.kang@kaist.ac.kr
3Department of Business Information Technology Amman, 11942, Jordan bashar.shboul@gmail.com
4College of Liberal Arts, Daejeon University Daejeon, 305-701, South Korea jihyun30@kaist.ac.kr
The reminder of the paper is organized as follows: Section 2 discusses the related work. Section 3 overviews the commonality indices used in manufacturing. In Section 4, we analyze the commonality indices from software perspective; In Section 5, we explain the design of the experiment, demonstrating the validity and effectiveness of the indices in Section 6. In Section 7, we outline some applications of commonality Indices. Finally, we suggest some extensions and conclude the paper in Section 8.

2. Related Work

There are two main research areas that are related to our work. The first area is commonality measurements which can be classified into two categories: work from the software product line engineering domain and from the manufacturing family design domain. The second area is software product line cost models.

2.1 Commonality Measurements

1) Approaches in SPLE

In the SPLE literature, to the extent of our knowledge, only very few preliminary studies for defining the suitable commonality metrics have been conducted [4, 5, 6, 7]. In [4], a metric to measure the functional commonality of the core assets with the purpose to evaluate the reusability of a given product line core asset have been introduced. This limits the assessment of commonality to the core asset instead of the products family level. Another work presented in [5] attempts to measure the degree of commonality at the component level. Where only ‘fully common components’ (common among all the products) contribute to the amount of commonality while ‘partially common components’ (common among some products) do not contribute at all.

In [6], a research plan addressing the implications of commonality and reuse on the cost of software maintenance is presented. The authors provide a set of mathematical expressions to assess the impact of different levels of commonality and reuse on implementation cost. However, their work considered the commonality among different versions of the product in which each maintenance operation on one product’s version is seen as the creation of a new version of the product with a certain degree of commonality with the previous version. Hence, the commonality is not analyzed within the context of the SPL; instead, they limit it to a relation between different versions of the product.

In another attempt [7], three commonality parameters have been introduced with the required theoretical rigor for their ideas. The first one, \( w_k \), is interpreted as the fraction of “demand” placed on a specific product that is shared by other products in the family. Where demand is defined as: “the average rate of new requirements that are placed on the product family during the planning period”. The second parameter, \( w \), is the fraction of the demand that is shared by two or more products, while the third parameter, \( \omega \), represents the fraction of demand the independent case (stand-alone that is shared by multiple products.

However, none of these attempts consider the differences among components while measuring the commonality, for example, some components are quite far expensive than other components. The impact of such difference on the commonality is not considered. More importantly, the metrics in the related work have not been analyzed with respect to their relationship with some quantified benefits of the SPL empirically. In other words, even within these existing works, there is no real attempt to map the amount of commonality to some desired quantity (e.g. development cost). The other works which have been proposed in software reuse [i.e. 8, 9, 10, 11], have limitation in evaluating the concept of commonality in SPLE. These metrics are rather oriented towards traditional reuse. Consequently, understanding and evaluating the implications of component commonality decision become very difficult.

2) Approaches in Manufacturing Domain

In manufacturing domain, a variety of analytical tools for measuring the effectiveness of a given product family design have been developed. The vast majority of such analytical tools are called Commonality Indices (CI) and focus on measuring the amount of component commonality among a family’s products and their correlation to some desired quantity, e.g. cost. Those indices are analytically feasible against the limited information indicative of early design (even if this information represents rough estimates) [12]. Such indices mark the starting point of the design of new families of products and the redesign of existing families [13]. An overview of those indices is presented in Section 3.

2.2 Software Product Line Cost Models

Software product line cost models exist at various levels of detail. For instance, The Constructive Product Line Investment model (COPLIMO) [13], is based on the Constructive Cost Model II (COCOMO) [15]. COPLIMO is used to make detailed cost estimations considering the cost of initial development of the SPL in addition to the costs and revenues from the future extensions of the product line, but is time consuming. As opposed to COPLIMO, Other cost models [i.e. 16, 17] work on an abstract level with less accuracy, but higher speed than more detailed models. Structured Intuitive Model for Product Line Economics (SIMPLE) [16] is one of the most popular cost models that determine the cost at an abstract level. SIMPLE provides four cost functions (\( C_{org} \), \( C_{cab} \), \( C_{reu} \), and \( C_{unique} \)) that can be combined in a number of scenarios covering several possibilities of evolving and initiating a SPL [17]. However, those cost functions return the costs considering the whole product line. For example, \( C_{cab} \) returns how much it costs...
to develop a core asset base suited to satisfy a particular scope not on individual components level.

Since our main concern is to project the development cost and capture the effects of component commonality on this cost, the products and the core asset base are described in term of components. Furthermore, we assume that C orgy() does not depend on the amount of component commonality. Therefore, out of the four cost functions of SIMPLE, we do not include the function C orgy(), which used to calculate organizational and process related costs. The interested reader is referred to [18] for detailed survey of economic models for Software Product Lines.

3. Commonality Measurements in Manufacturing Domain

This section gives a description of six of the commonality indices found in the manufacturing literature. The selected indices are based on a component perspective in which they mainly measure the similarities or differences between the components within a product family. Note that the indices presented here use the typically available data at the early stages of the development process.

3.1 Degree of Commonality Index (DCI)

The DCI is the most basic and traditional measure of component commonality developed by Collier [19] to compute the average usage of components within a product family. It is the ratio of the total numbers of components (all occurrences) to the number of distinct components. The maximum value for DCI is ∑∅ while the minimum value is (equal to) 1 which means there is no commonality among the products. The DCI is defined as follows:

\[ DCI = \frac{\sum_{j=1}^{d} \emptyset_j}{d} ; \quad 1 \leq DCI \leq \sum_{j=1}^{d} \emptyset_j, \]

where \( \emptyset_j \) is the number of repetitions of component \( j \) in the family, and \( d \) is the total number of distinct components in the family. DCI is a cardinal measure (does not have absolute boundaries) [20]. Thus, it is not able to define the commonality increase.

3.2 Total Constant Commonality Index (TCCI)

The TCCI developed by Wacker [20] as a modified version of DCI to convert it to a relative index with absolute boundaries. The TCCI is defined as follows:

\[ TCCI = \frac{d - 1}{\sum_{j=1}^{d} \emptyset_j - 1} ; \quad 0 \leq TCCI \leq 1, \]

where the parameters are the same as those used in DCI. TCCI can be interpreted as the ratio between numbers of common components in a product family with the total number of components in the family. Although TCCI has its fixed boundaries, it considers the number of components only, ignoring other useful information such as the price and size of the components.

3.3 Commonality Index (CI)

Martin and Ishii [21] modified DCI to the CI that has fixed boundaries between 0 and 1, but measures unique components. The CI is given by the following formula:

\[ CI = 1 - \frac{\sum_{j=1}^{u} \max P_j}{\sum_{j=1}^{V_n} P_j} ; \quad 0 \leq CI \leq 1, \]

where \( u \) is the number of unique components (distinct components), \( P_j \) is the number of components in product \( j \), and \( V_n \) is the final number of varieties offered (number of products). CI reflects the ratio between numbers of unique components to the total number of components in the family. Therefore, an increase in the value of CI is better since it indicates that small numbers of distinct components are used to achieve product variants in the family. CI can reach the maximum value when the number of distinct components is equal to the number of components in the larger product in the family (\( \max P_j \)). Similar to DCI and TCCI, the CI does not account for the cost of each component or other weighting criteria; it focuses only on the number of distinctive components.

3.4 Piece-based Commonality Metric (C_piece)

Johnson [12] proposed \( C_{\text{piece}} \) as the ratio between the number of components shared and the number of components that could be shared in a given product family. This ratio is calculated for each distinct component in all variant products in the family being analyzed. These quantities are summed and then divided by the total number of distinct components. \( C_{\text{piece}} \) is defined as follows:

\[ C_{\text{piece}} = \frac{\sum_{j=1}^{d} (\sum_{i=1}^{m} \gamma_{ji} - 1)/(m - 1))}{d} ; \quad 0 \leq C_{\text{piece}} \leq 1, \]

where \( \gamma = 1 \), if variant \( i \) contains component \( j \), and \( \gamma = 0 \), if it does not. The total number of product variants is \( m \), and \( d \) is the number of distinct components in the family. Also, the authors incorporate a flexible weighting parameter, \( \emptyset_j \), as:

\[ C_{\emptyset} = \frac{\sum_{i=1}^{d} \emptyset_i (\sum_{j=1}^{m} \gamma_{ji} - 1)/(m - 1))}{\sum_{j=1}^{d} \emptyset_j} ; \quad 0 \leq C_{\emptyset} \leq 1, \]
where $\emptyset_j$ is to capture the relative importance of component $j$, such as its mass, fabrication cost, etc.

### 3.5 Component Part Commonality Index ($\text{CI}(c)$)

Jiao [22] proposed the $\text{CI}(c)$ as an extension of DCI, taking into account the cost of each component, quantity per operation, and product’s volume. The $\text{CI}(c)$ is given by the following formula:

$$\text{CI}(c) = \frac{\sum_{j=1}^{d} [P_j \sum_{i=1}^{m} \emptyset_{ij} \sum_{i=1}^{m} (V_i Q_{ij})]}{\sum_{j=1}^{d} [P_j \sum_{i=1}^{m} (V_i Q_{ij})]} \leq \sum_{j=1}^{d} \sum_{i=1}^{m} \emptyset_{ij},$$

where $j$ is the index of each distinct component, $P_j$ is the price of each type of purchased component or the estimated cost of each internally-made component, $m$ is the total number of end products in a product family, $i$ is the index of each member product of a product family, $\emptyset_{ij}$ is the number of repetitions of component $j$ in product $i$, $\sum_{i=1}^{m} \emptyset_{ij}$ is the total number of repetitions of a distinct component $d_j$ across all the member products in the family, $V_i$ is the volume of end product $i$ in the family, and $Q_{ij}$ is the quantity of distinct component $d_j$ required by the product $i$.

### 4. Commonality Measurements for SPLE

This section presents an analysis of the aforementioned indices mainly from software development perspective and discusses how they can be adopted to measure commonality in SPLE.

#### 4.1 Analysis of Manufacturing Commonality Indices

1) **Weighing criteria**

As we can see that DCI, TCCI, CI, and $\text{C}_{\text{piece}}$ consider all components with the same weight (same effects), and hence, they do not completely capture the effect of each component on the level of commonality in a product family. In reality, some of the components may have higher impact on the overall commonality. For example, if a software product is composed of three components (A, B, C) and the component A contributes 70% of that product cost, the above indices will compute the same commonality, regardless of having component A, B, or C as a common component for same number of products. Rationally, commonality must be increased if component A is the common component. Therefore, when information is available, proper weighting criteria must be included in the commonality index such as component development cost. For instance, sharing the costly components can bring greater reductions in development cost in comparison with the obtained reductions by sharing the cheap components. For some product families, other weighting criteria can be used depending on the aspect under consideration. For example, for safety critical products families, the ‘required trustworthiness’ can be used. Sharing components with high ‘required trustworthiness values’ can increase the quality of the platform, and thus increasing the quality of the derived products. Another weighting criterion can be the complexity of the component. In this paper we focus only on the component cost as weighting criteria.

2) **Reuse cost**

In manufacturing, each component, even the shared component, must be assembled again or purchased. Accordingly, there is no difference in cost if that component is common or distinctive one, except for some cost reduction due to factors like quantity discount and process cost (commonality in supply and process). For SPLs, such components are defined once and then can be reused either with or without modification. Thus, we will have two types of costs: one for developing that component in the domain engineering phase and the other is related to the cost of reusing that component, which might actually be very low (in the case of a common component). Therefore, for SPLE, two types of costs must be included while considering the cost as a weighting parameter for computing the commonality index.

3) **Component quantity**

In software, we do not need to consider the quantity since we develop a shared component once and we reuse it. Hence, $\text{CI}(c)$ cannot be utilized as it’s for software domain, instead, we set the demand and quantity parameters as 1. The modified version of $\text{CI}(c)$ is presented in Section 4.2.

4) **Compatibility**

In contrast with the manufacturing components, where the compatibility of a shared component with the rest of a design is usually obvious and the effort for integrating such a component with the overall design is usually small. Component connections with other components can result in very significant and painful differences due to the complexity and cost of reusing that component in other products. Such a factor should be considered when we assess the commonality of the component. In this paper, the standardization of component connections is implicitly considered in the estimation of component cost. The implications of component connections require further investigation and analysis in future work. For instance, component connections can be explicitly used as a parameter in the commonality index.

5) **Desired variability**

The main limitation of these indices is that they do not fully consider the desired variability in the product family. In other words, these indices promote commonality among all components, including the ones that should remain product specific or variant to differentiate products in the family, because those components add desired variety to the
product family. For a successful product line portfolio, each product within a product line should be different from the other products in ways that are meaningful to the customers in each relevant market segment. Without considering the benefits of the variability, one easily overestimates the need for commonality and, thus, ignores the essence of the product line approach. Therefore, reaching the perfect value for commonality of these indices is not the ideal target for the family design. Instead, the ideal target to achieve is the maximum possible value of these indices while considering the desired variability. For example, the minimum number of components to achieve the required variability, \( d_{\text{min}} \), is 20; thus the perfect commonality using TCCI, for instance, can be achieved when \( d = d_{\text{min}} \).

6) Computation

In order to use these indices for measuring software product family commonality, it is necessary to establish a definition of a component’s sharing status. This is required to avoid ambiguity in measuring the values of the parameters used in the above indices. We explain this concept by the following example:

Example: Assume that component X is developed as a common component and shared among four products (A, B, C, and D). Thus, we have one distinct component X with four repetitions in the family. However, if for product D we have to develop another candidate of component X, let’s call it X’, then we have two distinct components: X with three repetitions and X’ with one repetition. The computation of the aforementioned indices is shown in the appendix.

4.2 Adopted Commonality Indices

For the issues discussed in the previous Sub-section, we can measure commonality in SPLE by adopting manufacturing indices as follows:

1) DCI, TCCI, CI and \( \text{C}_{\text{piece}} \)

These indices can be applied for measuring the commonality for SPL as the parameters used to calculate their values can also be used to describe software components and the structure of software families (i.e. number of products using the component, number of unique components, and number of products).

2) \( \text{C}_\emptyset \) and \( \text{SCI}(e) \)

Due to the different characteristics between software and manufactured components, both these “specific” indices require modifications. For this reason we:

- Propose the Software Commonality Index \( (\text{SCI}(e)) \), which considers the cost of each component, as a modified version of \( \text{CI}(e) \) as follows:

\[
\text{SCI}(e) = \frac{\sum_{j=1}^{d} \mathcal{C}_j \left( \sum_{i=1}^{m} \varnothing_{ij} \right)}{\sum_{j=1}^{d} \mathcal{C}_j} ; 1 \leq \text{SCI}(e) \leq m,
\]

where \( \mathcal{C}_j \) is the cost of the distinct component j in the family, \( \sum_{i=1}^{m} \varnothing_{ij} \) represents the number of products using component j and the remaining parameters are the same as those used in \( \text{CI}(e) \) above. This index can be regarded as the extent to which the costs of common components are distributed across the products of a product family. When the value reaches m, in this case all the components are common among all the products in the family while the value of one means that all components are variants in the family (no commonality).

- The flexible weighting parameter, \( \varnothing_{j} \), which is used in \( \text{C}_\emptyset \) is considered and computed as the cost of software component. This is due to the difference in computing the price of manufactured component and the cost of software component (i.e. reuse cost). Thus \( \text{C}_\emptyset \) is modified as follows:

\[
\text{C}_{\text{cost}} = \frac{\sum_{j=1}^{d} \mathcal{C}_j \left( \sum_{i=1}^{m} \varnothing_{ij} \right) / (m-1)}{\sum_{j=1}^{d} \mathcal{C}_j} ; 0 \leq \text{C}_{\text{cost}} \leq 1,
\]

Here, the parameters are the same as those explained in Section 3.

The usage of these indices varies based on the strategy of the company and the level of available information on the components. Depending on that, the most relevant index or indices can be chosen based on Table 1.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>DCI</th>
<th>TCCI</th>
<th>CI</th>
<th>( \text{C}_{\text{piece}} )</th>
<th>( \text{SCI}(e) )</th>
<th>( \text{C}_{\text{Cost}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strategy</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Increasing the number of common components</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Reducing cost</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Available information</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of products</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>List of components for each product</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Components costs</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
<td>√</td>
</tr>
</tbody>
</table>

Table 1. Target commonality indices based on company strategy
5. Experiment Design

This section presents the indices assessment methodology. More specifically, it describes the hypotheses behind the research study performed in this paper and outlines the independent and dependent variables that are studied during both the data collection and analysis phases. The development cost modeling methodology used to project the consequences of components development design alternatives (product line platform and standalone based development) is also detailed in this section.

5.1 Hypotheses

The usefulness of the aforementioned indices arises when they correlated to some desired quantity of their effectiveness (i.e. development cost reduction). In order to assess the aforementioned commonality indices, the principal question we address in this Section is: what effect, if any, does the degree of commonality as measured by those indices have on the software development cost?

The cornerstone of any valid empirical study is the definition of a set of important hypotheses that need to be validated. Hence, we came up with the following testable hypotheses:

H1.1 The degree of commonality, as measured, has significant effects on the development cost.

a. The degree of commonality, as measured by commonality index X, has significant effects on the development cost.

H1.2 The degree of commonality as measured has a significant trend effect on the development cost.

a. The degree of commonality, as measured by commonality index X, has significant trend effects on the development cost.

where X represents each of the six indices mentioned in Sub-section 4.2 (i.e., X = DCI, TCCI,...). Therefore, we have six underlying hypotheses for H1.1 and H1.2, respectively.

5.2 Variables

In order to proceed with the experiment, the defined hypotheses need to be mapped onto a set of measurable independent and dependent variables. These variables are measured in the experiment and are used in the analysis phase. Within the cause-effect relationship that we are interested in, the adopted commonality indices represent the cause. Therefore, they are the independent variables for the study (each of these indices is an independent variable).

In the other hand, the development cost represents the effect in the cause-effect relationship as described above. Therefore, it is considered to be the dependent variable. The measurement of the dependent variable was performed by introducing a cost modeling methodology and inputs from a team of human experts.

5.3 Experimental Setup

To explore the relationships between the commonality indices and the resultant development cost, an experiment is designed using a simulation carried out on a family of software products. The purpose of the simulation is to generate valid designs of the family. Each design is represented by the set of component candidates that have to be selected to assemble the required list of products. The following Sub-sections detail the setup of the experiment:

1) Objects of Study

To assess alternative commonality indices based on their relationship to development cost, a family of five Door Lock Control Management System (DLCS) products was analyzed. The family is part of a home automation system that controls door lock and user control admission. Table 2 shows the family being analyzed which have different features in admittance control, door type, and door lock management. The reason for selecting DLCS is twofold. First, DLCS is one of the most used real-world problems in the software product line literature, hence, the necessary data for the analysis could be found. Second, the home automation domain is well understood by the participants of our study.

The initial design of the DLCS family was specified with the input of a team consists of six expert designers and the specifications of home automation SPL reported in [1]. The process that the team undertook to analyze and specify the initial design of the DLCS and necessary inputs for computing costs was as follows:

• Initial phase: all the experts were provided with a description of the DLSC and necessary references, and were given a period of ten days to improve their understanding of the study and the design of the DLCS family. During this period, several sessions were conducted to discuss any ambiguity faced the team. By the end of this period we ensured that experts are qualified to create rational components’ designs for the DLCS family.

• Second phase: two sessions were hold with the purpose of identifying the required components’ candidates, to realize the five products (in a stand-alone development fashion and SPL fashion).

Graduate students of software engineering and electrical engineering at KAIST, who are familiar with SPL approach, hardware design and possess more than 4 years of experience in software development.
The outputs of this phase are reported in Table 3. The first output represents the necessary components’ candidates for building each product individually (each component in the family will be developed specifically for one product). This assignment of components’ candidates to the products in the family represents the “Initial Design” of the DLCS (i.e. no commonality Design). The Initial Design represents one object in our study. The second output represents a set of recommendations about which components can be shared among some or all the products. These recommendations are used to generate different objects (Designs) of the DLCS, each with a certain amount of component commonality. Experts agreed that these recommendations are rational based on their understanding of the domain, their experiences, and details presented in [1]. In the final phase, experts provided the necessary inputs for measuring the dependent variable.

2) Data Collection

As mentioned previously, in this study, the economic value of a product line arises from the assumed reduction in development cost due to component commonality. Therefore, the cost model has to be capable of projecting the development cost and capturing the effects of component commonality on this cost. These projections are used to assess the commonality indices analyzed previously. In addition, the cost model is used to estimate the components costs in order to compute Cost and SCI(\(C\)). For these reasons and those discussed in Sub-section 2.2, we introduce a cost function for estimating a SPL development cost as follows:

\[
\text{Cost}_{PL} = \sum_{j=1}^{J} \text{Cost}^j,
\]

where \(\text{Cost}^j\) is the total cost of component \(j\) calculated with the following equation:

\[
\text{Cost}^j = \sum_{k=1}^{k^j} \text{Cost}^j_k ; (j=1,2,\ldots,J),
\]

where \(k^j\) is the number of different component candidates to be developed for component \(j\) while \(\text{Cost}^j_k\) is the cost of candidate \(k\) of component \(j\) in the family. As we discussed previously, each component’s candidate that used in the family is considered as a distinctive component which has its own cost. Considering the context of SPLE, for estimating the cost of candidate \(k\) of component \(j\), we have to underline that SPLE consists of two main phases, namely, domain engineering, and application engineering. The former is the phase in which core assets are defined and realized while the latter is the phase in which family products are built by reusing the core assets and building products specific software. Based on this distinction, we applied SIMPLE’s costs functions for estimating the cost of candidate \(k\) of component \(j\), \(\text{Cost}^j_k\), as follow:

\[
\text{Cost}^j_k = \begin{cases} 
\text{Cost}^j_k + \text{Cost}^j_{\text{reused}_k} & \text{if } C^j_k \in \text{CAB} \\
\text{Cost}^j_{\text{unique}_k} & \text{if } C^j_k \notin \text{CAB} \\
0 & \text{otherwise} \end{cases},
\]

where \(\text{CAB}\) represents the set of components in the core asset base. In this work, any component candidate shared by two or more products was considered as a core asset component. \(\text{Cost}^j_k\) is the cost to develop component candidate \(C^j_k\) as a shared component and reuse it once. The shared component introduces extra costs for making it more generic. We calculate this cost by applying a Relative Cost of Writing for Reuse (RCWR) factor on the component traditional development cost (the cost of developing component for one-time use). \(\text{Cost}^j_{\text{reused}_k}\) is the cost to reuse the shared component candidate \(C^j_k\) in the intended product variant. We calculate this cost by applying a Relative Cost of Reuse (RCR) factor on the component traditional development cost. \(\text{Cost}^j_{\text{unique}_k}\) is the cost for developing component candidate \(C^j_k\) without reuse as unique component and to use it once (not based on product line platform).

The application of these costs requires estimations of the effort for the components. Most methods for estimating the required effort for a component depend on the size of the component.

As we discussed above, the final phase was for estimating the necessary inputs for measuring components’
### Table 3. Components’ candidates for generating Designs of the DLCS family

<table>
<thead>
<tr>
<th>Output 1</th>
<th>Candidates(^b)</th>
<th>Output 2</th>
</tr>
</thead>
</table>
| **C1**: User Control Plug-in: represents the variable plug-in components for different user admission/access
| C\(^1\)_1: User control for manual access for P1. | Develop a generic user control for manual, \(C\_{g(1,2)}\), as a common component among P1 and P2. \(C\_{g(1,2)}\) provides manual access functionality for P1 and P2, and can replace both of \(C\_1\) and \(C\_2\). |
| mechanisms | C\(^1\)_2: User control for manual access for P2. | |
| | C\(^1\)_3: User control for keypad with short password, P3. | |
| | C\(^1\)_4: User control for keypad with long password, P4. | |
| | C\(^1\)_5: User control for fingerprint, P5. | |
| **C2**: Authentication Plug-in: represents variable plug-in components for different authentication | C\(^2\)_1: Authentication for the verification of a short password, P3. | Develop \(C\_1\) as a common component among P3 and P4. As a consequence, extra cost is introduced for making it more generic. |
| mechanisms. | C\(^2\)_2: Authentication for the verification of a long password, P4. | |
| | C\(^2\)_3: Authentication for the verification of a fingerprint, P5. | | 
| **C3**: User Control Manager: controls the operations of user admission and have the basic users’ settings | C\(^3\)_1: User control manager for P1. | Develop a generic user control manager, \(C\_{g(1–5)}\), as a common component. |
| | C\(^3\)_2: User control manager for P2. | \(C\_{g(1–5)}\) provides user control functionality for all products, P1–P5. |
| | C\(^3\)_3: User control manager for P3. | |
| | C\(^3\)_4: User control manager for P4. | |
| | C\(^3\)_5: User control manager for P5. | |
| **C4**: Authentication Manager: control authentication basic operations and threads, always grants authentication in the case of the absence of the authentication plug-in [1] | C\(^4\)_1: Authentication manager for P1. | Develop a generic authentication manager, \(C\_{g(1–6)}\), as a common component among all the products. |
| | C\(^4\)_2: Authentication manager for P2. | |
| | C\(^4\)_3: Authentication manager for P3. | |
| | C\(^4\)_4: Authentication manager for P4. | |
| | C\(^4\)_5: Authentication manager for P5. | |
| **C5**: Sensor Plug-in, represent variable plug-in components with different types and speeds (different manufacturers). | C\(^5\)_1: Type A provides low speed for P2. | Cannot be developed as common component. |
| | C\(^5\)_2: Type B provides medium speed for P3. | |
| | C\(^5\)_3: Type C provides high speed for P4. | |
| | C\(^5\)_4: Type D provides very high speed for P5. | |
| **C6**: Lock Actuator Plug-in, represents variable plug-in components to control different actuators | C\(^6\)_1: Lock actuator for swing door, P1. | Develop a common lock actuator for sliding door, \(C\_{g(3–5)}\), capable of operating with different type of sensors. |
| | C\(^6\)_2: Lock actuator, compatible with \(C\_{g(5)}\). | |
| | C\(^6\)_3: Lock actuator, compatible with \(C\_{g(5)}\). | |
| | C\(^6\)_4: Lock actuator, compatible with \(C\_{g(5)}\). | |
| | C\(^6\)_5: Lock actuator, compatible with \(C\_{g(5)}\). | |
| **C7**: Lock Control, provide the basic operations and generic classes to be inherited by a variety of connected plug-in components. | C\(^7\)_1: Lock control for basic lock operations for P1. | Develop a generic lock control, \(C\_{g(1–5)}\), as a common component among all the products. |
| | C\(^7\)_2: Lock control for basic lock operations for P2. | |
| | C\(^7\)_3: Lock control for basic lock operations for P3. | |
| | C\(^7\)_4: Lock control for basic lock operations for P4. | |
| | C\(^7\)_5: Lock control for basic lock operations for P5 | |
| **C8**: Door Lock Plug-in, represents the variable plug-in components for door lock control | C\(^8\)_1: Manual door lock, P1. | Developing a generic electronic door lock, \(C\_{g(3–5)}\), which can be used among products P3–P5 as common component. |
| | C\(^8\)_2: Electronic without authentication and auto close for P2. | \(C\_{g(3–5)}\) provides authentication and auto close functionality for products P3–P5. |
| | C\(^8\)_3: Electronic with authentication and without auto close for P3. | With this replacement, P3 will have extra functionality (auto close) as a consequence of increasing component commonality. |
| | C\(^8\)_4: Electronic with authentication and auto close for P4. | |
| | C\(^8\)_5: Electronic with authentication and auto close for P5. | |

\(^b\) \(C\_k\) to denote candidate k of component j
Table 4. Cost factors of components

<table>
<thead>
<tr>
<th>Component</th>
<th>Candidates</th>
<th>Cost($)</th>
<th>RCWR</th>
<th>RCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>C^1</td>
<td>c^1_1</td>
<td>810</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^1_2</td>
<td>1000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^1_{g(1-2)}</td>
<td>1445</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c^1_3</td>
<td>1550</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^1_4</td>
<td>2100</td>
<td>1.42</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>c^1_5</td>
<td>3583</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C^2</td>
<td>c^2_1</td>
<td>1650</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^2_2</td>
<td>2846</td>
<td>1.50</td>
<td>0.1</td>
</tr>
<tr>
<td></td>
<td>c^2_3</td>
<td>3917</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C^3</td>
<td>c^3_1</td>
<td>3583</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^3_2</td>
<td>3750</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^3_3</td>
<td>4600</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^3_4</td>
<td>4950</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^3_5</td>
<td>6167</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^3_{g(1-5)}</td>
<td>7680</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>C^4</td>
<td>c^4_1</td>
<td>2350</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^4_2</td>
<td>2710</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^4_3</td>
<td>3667</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^4_4</td>
<td>4000</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^4_5</td>
<td>4829</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^4_{g(1-5)}</td>
<td>6100</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>C^5</td>
<td>c^5_1</td>
<td>833</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^5_2</td>
<td>1067</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^5_3</td>
<td>1333</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^5_4</td>
<td>1861</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>C^6</td>
<td>c^6_1</td>
<td>1700</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^6_2</td>
<td>2067</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^6_3</td>
<td>2433</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^6_4</td>
<td>2860</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^6_5</td>
<td>3500</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^6_{g(2-5)}</td>
<td>5225</td>
<td>0.15</td>
<td></td>
</tr>
<tr>
<td>C^7</td>
<td>c^7_1</td>
<td>3533</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^7_2</td>
<td>5617</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^7_3</td>
<td>5867</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^7_4</td>
<td>6210</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^7_5</td>
<td>6500</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^7_{g(1-5)}</td>
<td>8833</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>C^8</td>
<td>c^8_1</td>
<td>1500</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^8_2</td>
<td>1833</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^8_3</td>
<td>3667</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^8_4</td>
<td>4860</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^8_5</td>
<td>4910</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>c^8_{g(3-5)}</td>
<td>5689</td>
<td>0.15</td>
<td></td>
</tr>
</tbody>
</table>

In order to do that, the experts decided to apply Wideband-Delphi estimating [23] to estimate components’ size in Function Point [24]. After obtaining the size of each component candidate, experts and based on their experiences specified that the average programmer productivity for such system is 30FP/month and on average each of them would be paid 2500$/month (as a developer). Consequently, we had the first input for our cost model (traditional development cost). We denote this cost as I_k.

Next step was to estimate RCWR and RCR values for components’ candidates that have potential to be part of the core assets. The Wideband-Delphi was applied to estimate these values. The estimated cost factors for each component are shown in Table 4. For example in the table, c^3_{g(1-5)} is a generic component, therefore it has only RCR value, while c^1 have a potential to be developed for reuse, thus added to the core asset; therefore it has RCWR and RCR values.

As a summary, the simulation process using DLCS is described as: “A set of components are identified, C ≡ \{c^1, c^2, ..., c^j\}. Each component, c^j, may take on one out of a finite set of candidates, C_k ≡ \{c^1_j, c^2_j, ..., c^k_j\}. That is, c^j = c_k \exists c\in C_k , where k = 1, 2, ..., k, denotes the k\textsuperscript{th} candidate of c^j. For each c_k, the related estimated costs are denoted as I_k, RCWR\_k, RCR\_k. A set of feasible designs a product family, P\_family = {p_1, p_2, ..., p_I}, is denoted as S \equiv \{s_1, s_2, ..., s_n\}, are generated by choosing one of the component candidates c_k \ ( k = 1, 2, ..., K), for a component C^j(j = 1, 2, ..., J) of a product P_i (i = 1, 2, ..., I), subject to satisfying certain constraints. The cost I, and degree of commonality measured by DCI, TCC, CI, C\_piece, C\_cost and SCI(C^j).

In order to perform this simulation, we developed an automated tool that would generate all the valid Designs and would calculate the values of all independent variables and the dependent variable for the generated Designs. For this purpose, we extended the cost model as follows:
\[ a_{kj} = \sum_{i=1}^{I} x_{kj}^{(i)} \quad (j = 1, 2, \ldots, J; k = 1, 2, \ldots, K), \]

where, \( x_{kj}^{(i)} \) is a 0-1 integer variable takes its value as follows:

\[
x_{kj}^{(i)} = \begin{cases} 
1 & \text{if } c_{kj}^{(i)} \text{ is selected for component } c_j \text{ of } p_i \\
0 & \text{if } c_{kj}^{(i)} \text{ is not selected for component } c_j \text{ of } p_i 
\end{cases}
\]

(\( i = 1, 2, \ldots, I; j = 1, 2, \ldots, J; k = 1, 2, \ldots, K \)).

Thus, \((a_{kj} - 1)\) represents the number of products reusing candidate \( k \) of component \( j \).

5.4 Analysis Technique

For each valid family configuration generated by the simulation process, the value of the development cost and the commonality using these indices were computed. The collected data is quantitatively reasonable from the perspective of the amount of data. We obtained 6408 index values (1086 valid DLCS designs, six commonality indices) and 1068 cost values. The 6408 index values are the values for the independent variables, whereas the 1086 values are the values for the dependent variable. Over all we obtained 7602 data. To assess the relationship between the various commonality indices and the resultant development cost, a simple linear regression analysis was applied on these data points to find out how the variation in commonality affects/relates to the development costs of the respective family under consideration. \( R^2 \) served as the basis of index quality. The higher \( R^2 \) is, the better the index. Having obtained the results from six regression analyses, these were analyzed individually and then compared.

6. Analysis and Results

This section describes the results of the analysis performed over the observations made during the experiment and shows how the quantitative results of the analysis are relevant for supporting our hypotheses. As well it states the possible threats to validity of the research.

6.1 Regression Study

The summary results for each regression are reported in Table 5. At first, we observed that the validity of all regression models was quite robust in explaining the relationship between cost and commonality, as indicated by the \( R^2 \)-squared and adjusted \( R^2 \)-square values. The results determine that, whereas overall null hypothesis 1.1 is acceptable, each of the underlying hypotheses 1.1a, 1.1b, 1.1c, 1.1d, 1.1e, and 1.1f are also accepted. Therefore, regardless of the index used, the relationship between commonality and cost is verifiable, though there are variations in the degree of commonality and resultant costs. The significance for these results is very high.

As the next step, we compared the changes in the predicted costs with observed costs for different degrees of commonality measured by each of the referred commonality indices. The trends are depicted below. The trend analysis revealed a significant linear relationship between the measured commonality and the total development cost. The results determine that, whereas overall null hypothesis 1.2 is acceptable, each of the underlying hypothesis 1.2a, 1.2b, 1.2c, 1.2d, 1.2e, and 1.2f are also accepted. The results empirically analyze and confirm the relationship between commonality and development cost. Moreover, this study provides a way to directly relate the degree of commonality to the development cost. For example, we can define a regression model to estimate the development cost similar to the one defined by Collier [18] to evaluate or compare different designs as shown in Fig.1.

Table 5. Regression analysis results

<table>
<thead>
<tr>
<th>Independent Variable</th>
<th>No of Obs.</th>
<th>R-squared</th>
<th>Adjusted R-squared</th>
<th>T stat (Constant Coefficient)</th>
<th>T stat (Ind. Variable Coefficient)</th>
<th>Significance (p-value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI</td>
<td>1086</td>
<td>0.8110</td>
<td>0.8109</td>
<td>139.64</td>
<td>-66.66</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(160341.33)</td>
<td>(-46016.64)</td>
<td></td>
</tr>
<tr>
<td>TCCI</td>
<td>1086</td>
<td>0.8247</td>
<td>0.8245</td>
<td>195.70</td>
<td>-69.78</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(129648.38)</td>
<td>(-114440.67)</td>
<td></td>
</tr>
<tr>
<td>CI</td>
<td>1086</td>
<td>0.8247</td>
<td>0.8245</td>
<td>195.70</td>
<td>-69.78</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(129648.38)</td>
<td>(-92188.31)</td>
<td></td>
</tr>
<tr>
<td>CPiece</td>
<td>1086</td>
<td>0.8111</td>
<td>0.810</td>
<td>243.15</td>
<td>-66.66</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(114324.69)</td>
<td>(-184066.5)</td>
<td></td>
</tr>
<tr>
<td>CCost</td>
<td>1086</td>
<td>0.9547</td>
<td>0.9546</td>
<td>520.2</td>
<td>-147.7</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(115649)</td>
<td>(-91798.16)</td>
<td></td>
</tr>
<tr>
<td>SQI_c</td>
<td>1086</td>
<td>0.9547</td>
<td>0.9546</td>
<td>371.19</td>
<td>-147.7</td>
<td>0.005</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(138598)</td>
<td>(-22949.5)</td>
<td></td>
</tr>
</tbody>
</table>
For this experiment, the important conclusion based on Fig.2 is that the development cost decreases as the degree of commonality measured using the proposed indices increases. Moreover, it appears that $C_{\text{Cost}}$ and SCI tend to reveal the relation between commonality and cost more clearly than the other indices. That is not surprising because both indices already weigh the components based on their costs while measuring the commonality. The values of the SCI and $C_{\text{Cost}}$ were measured in the above sections under deterministic component costs assumptions. However, in practice, the cost of a component is difficult to estimate. One way to handle this apparent discrepancy is by determining the sensitivity of the value of both indices to components costs; thus, the effect of estimating the component costs can be analyzed. Here, the values of SCI and $C_{\text{Cost}}$ in Design 5 were analyzed. For each value, seven simulations were conducted (similar to the procedure used in [25]). In each simulation, the components’ cost estimates were replaced with normal distribution centered on the target ‘values’ with a given standard deviation (5%, 10%, 15%, 20%, 25%, and 30%). A 100,000-trial simulation was conducted after computing the distribution of each cost. The consistency of the results of the simulation with the original values as indicated by the resultant $\text{Mean}_{\text{index}}$ and the target values can be observed in Table 6. However, it is not strange since cost estimates were replaced with normal distribution centered on the target ‘values’. Although the high standard deviations of the inputs component cost up to 35% of the corresponding values, the standard deviation of the resultant SCI values is very low, less than 5% of the MeanSCI. Furthermore, the standard deviation of the resultant $C_{\text{Cost}}$ value is also less than 5% of the MeanC_cost. Based on these results, we conclude that both indices are very robust regarding the component cost.

**Table 6.** Consistency of the results of the simulation with the original values

<table>
<thead>
<tr>
<th>SCI</th>
<th>C_{Cost}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>Std Dev</td>
</tr>
</tbody>
</table>

**Figure 1.** Example of regression model for our experiment

Estimated development cost = average of costs in the experiment + regression coefficient × (index value – average of commonality index values in the experiment).

For example, using the SCI value, the model is:

Estimated development cost = $83,848 - 22,949 \times (\text{index value} - 2.388)$.

Using this model we can estimate the cost for the family of DLCS with SCI= 2.8 as follows:

Estimated development cost = $83,848 - 22,949 \times (2.8 - 2.388) = $74392.

**Figure 2.** Trend analysis (Development cost, Commonality indices)
6.2 Threats to Validity

We identified the following threats concerning the construct validity, internal validity, external validity, and conclusion validity of the work presented in this paper.

1) Conclusion validity

In our experiment, only one product family have been analyzed and used for generating objects of the study. The reason is that we could not find any publicly available software products family with the necessary information (components costs, design of components, number of products). This is a threat to the validity of the conclusions. We are currently working on two other SPL case studies from different domains. Additionally, we are trying to access a wider range of appropriate participants who had enough industrial knowledge within the area of software product lines.

2) Construct validity

Lack of a formal hypothesis and statistical tests: for the relationship between commonality indices and the development cost, the hypotheses of interest are clearly, though textual, presented and statistically tested using a regression analysis study.

Accuracy of dependent and independent variables measures: the development cost (dependent variable) was measured using the cost model we proposed. The cost model utilizes the cost functions used in a well know cost model, SIMPLE. The cost functions are based on the main component’s cost parameters in software development economics [i.e. 9, 13, 14]. However; those costs are still based on participants’ estimates and assumptions. As regards to the set of commonality indices (independent variables), the validity of the used indices is considered through the following points:
- Selecting widely used indices in the manufacturing literature that have shown their effectiveness and validity in measuring commonality through real industrial applications and studies.
- An analysis of the applicability of these indices for measuring commonality in SPL considering the nature of the software.

Furthermore, we manually computed development cost and commonality indices for 10 different generated designs to ensure that there were no bugs in our tool.

3) Internal validity

Differences among subjects: to minimize this threat, the variations in subject skills were reduced through the team selection criteria, where subjects (team members) have to possess same number of years of experience in software development. Furthermore, the subjects are graduate students attending a software product line engineering course and working on the same project. Thus, subjects have approximately the same degree of experience with regard to product family design concepts, commonality and variability analyses, and cost estimation techniques. Therefore, error variance from difference among subjects is kept to a possible minimum.

Accuracy of subject assumptions: in practice, the cost of component is difficult to estimate. To enhance the estimation accuracy, we select a widely used case study that the subjects were familiar with, namely DLCS. Furthermore, prototyping was used when necessary to improve the accuracy of subject assumptions. Based on that, we considered their responses valid.

Fatigue effects: in our experiences, graduate students usually attend a 2-hour’s meetings for their research

### Table 6. Sensitivity analysis results for SCI and C\text{Cost}

<table>
<thead>
<tr>
<th>STD\text{Cost}</th>
<th>Original index value (target)</th>
<th>Simulation results (output statistics of SCI and C\text{Cost})</th>
<th>Mean\text{Index}</th>
<th>STD\text{Index}</th>
<th>Max</th>
<th>Range</th>
<th>STD\text{Index}</th>
<th>Mean\text{Index}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Min</td>
<td>SCI\text{Cost}</td>
<td>SCI\text{Cost}</td>
<td>SCI\text{Cost}</td>
<td></td>
<td></td>
<td>SCI\text{Cost}</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td>5%</td>
<td>3.249</td>
<td>3.237</td>
<td>3.249</td>
<td>2.20%</td>
<td>3.262</td>
<td>0.025</td>
<td>0.68%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td>10%</td>
<td>3.249</td>
<td>3.23</td>
<td>3.249</td>
<td>4.30%</td>
<td>3.268</td>
<td>0.038</td>
<td>1.32%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td>15%</td>
<td>3.249</td>
<td>3.225</td>
<td>3.249</td>
<td>6.80%</td>
<td>3.274</td>
<td>0.049</td>
<td>2.09%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td>20%</td>
<td>3.249</td>
<td>3.217</td>
<td>3.249</td>
<td>7.12%</td>
<td>3.282</td>
<td>0.065</td>
<td>2.19%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td>25%</td>
<td>3.249</td>
<td>3.21</td>
<td>3.249</td>
<td>10.26%</td>
<td>3.287</td>
<td>0.077</td>
<td>3.16%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td>30%</td>
<td>3.249</td>
<td>3.204</td>
<td>3.249</td>
<td>11.83%</td>
<td>3.293</td>
<td>0.089</td>
<td>3.64%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td>35%</td>
<td>3.249</td>
<td>3.196</td>
<td>3.249</td>
<td>14.15%</td>
<td>3.301</td>
<td>0.105</td>
<td>4.36%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td>0.519</td>
<td>0.507</td>
<td>0.35%</td>
<td>0.532</td>
<td>0.025</td>
<td>0.67%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td></td>
<td>10%</td>
<td>0.519</td>
<td>0.519</td>
<td>0.67%</td>
<td>0.538</td>
<td>0.038</td>
<td>1.29%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td></td>
<td>15%</td>
<td>0.519</td>
<td>0.495</td>
<td>1.05%</td>
<td>0.544</td>
<td>0.049</td>
<td>2.02%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td></td>
<td>20%</td>
<td>0.519</td>
<td>0.487</td>
<td>1.12%</td>
<td>0.552</td>
<td>0.065</td>
<td>2.16%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td></td>
<td>25%</td>
<td>0.519</td>
<td>0.48</td>
<td>1.61%</td>
<td>0.557</td>
<td>0.077</td>
<td>3.11%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td></td>
<td>30%</td>
<td>0.519</td>
<td>0.474</td>
<td>1.83%</td>
<td>0.563</td>
<td>0.089</td>
<td>3.53%</td>
<td>SCI\text{Cost}</td>
</tr>
<tr>
<td></td>
<td>35%</td>
<td>0.519</td>
<td>0.466</td>
<td>2.22%</td>
<td>0.571</td>
<td>0.105</td>
<td>4.29%</td>
<td>SCI\text{Cost}</td>
</tr>
</tbody>
</table>
projects. Therefore, we ensured that all the sessions held with the participants last for less than 75 minutes to avoid the fatigue effect in our study.

4) External validity
Representativeness of the case study: the selected product line might not be representative of all product line practices. However, we chose one of the most used real-world problems in the software product line literature. Nevertheless, this does not automatically imply external validity of the results; instead, additional replications are necessary to determine if our findings can be generalized to other domains.

Uncertainty in modeling the problem: there is a range of uncertainty factors in modeling the problem such as individual development cost, cost of developing reusable components, and cost of reuse. As regards to the impact of those uncertainty factors on $C_{\text{Cost}}$ and SCI($C^{(c)}$) values, a sensitivity analysis has been carried out. Sensitivity analysis helps in determining how sensitive the value of the indices is to the component costs, and thus, the effect of estimating the component costs can be analyzed.

7. Applications of the Commonality Indices

This section outlines some examples of how commonality indices can be used during product family design. The benefits of commonality indices come from the information that they can provide during product family design. Such information helps the designers to: firstly, decide which design to use among alternatives by ranking different designs based on the amount of commonality, and secondly, to focus on the design of components that give the most commonality. To demonstrate how commonality indices are helpful while designing the product families, we show four candidate Designs of DLCS family in Table 7. In the table, each design represents the selected component candidate for each component ($C^{(1)} - C^{(5)}$) to build each product in the family (P1 ~ P5). For example, $C^{(2)}_{\text{p(1-2)}}$ is used for the component $C^{(1)}$ in P1 and P2 while $C^{(4)}$ is used for $C^{(1)}$ as a common component among P3 and P4. The computed values of the commonality indices and development cost for each Design compared to the Initial Design (Design 1) are reported in Tables 8 and 9 respectively. Next we present some applications of commonality measurements in supporting product family design.

7.1 Ranking for Different Designs Based on the Amount of Commonality

The results reported in Table 8 show that Design 5 has the highest values for all the commonality indices. This trend is consistent between the commonality indices. Furthermore, based on the performed simulation, Design 5 is the optimized design on commonality measurements (the design with highest amount of commonality among all the 1086 possible DLCS’s Designs). On the other hand, looking at Table 9 reveals that Design 4 has the lowest development cost. Again, this Design is the optimized design on development cost in our experiment. The reason is that in Design 5, $c^{(2)}$ is designed as a common component among products 3 and 4. This increases the values of commonality indices compared to Design 4. However, developing $c^{(2)}$ as common and reusing it costs more than developing $c^{(1)}$ for product 1 and $c^{(2)}$ for Product 2. In other words, reusing $c^{(2)}$ did not pay off compared to traditional development. This interesting finding shows that the highest commonality Design does not guarantee the lowest development cost. Instead a set of non-dominated solutions can exist when we simultaneously consider commonality and development cost. To solve this issue, human experts can decide the “best” solution among this set of Designs by considering other factors such as the evolution cost. For Design 2 and Design 3, the trend is not consistent between the commonality indices. For instance, both Designs have same values for DCI, TCCI, CI and $C_{\text{piece}}$. However, Design 3 has higher values for $C_{\text{Cost}}$ and SCI($C^{(c)}$) making Design 3 more effective in term of development cost reduction. This reduction can be seen in the development cost of Design 3 compared with the development cost of Design 2 as shown in Table 9. Therefore, when different designs result in the same index value (i.e. DCI), the computation of “specific” index (i.e. SCI($C^{(c)}$)) can differentiate between those designs. A closer look at Table 8 reveals that Design 2 has significant impact on the values of the commonality indices, therefore it is not surprising to see that Designs 3~5 are actually further improvement to Design 2 (all these designs share ‘user control manager’, ‘authentication manager’ and ‘lock control’ as common components).

7.2 Recommendations on Which Component Can be Designed /Redesigned as a Common Component.

Commonality indices can be used to evaluate the impact of each component within a product family on the degree of commonality. Hence, designers can prioritize and focus on components based on that impact, where those with higher impact come first (those components are the most valuable to share). Such prioritization can help the designer on planning and allocating the resources for designing/redesigning components. By considering the Initial Design, as the base line for assessing the influence of each component on the overall commonality, we compute the percentage of improvement results from changing component design as follows:

$$\frac{\text{new value} - \text{initial value}}{\text{initial value}} \times 100\%$$

where new value represents the index value after making the change in the component under consideration (to achieve its
target design), and initial value represents the index value of the Initial Design (Design 1). For instance, to assess the impact of each component on the overall commonality of Design 5 (Design 5 represents the ideal design of components in term of commonality), we apply the following procedure:

For each component, we use its design as presented in Design 5 while considering the original design of the remaining components (original design is the baseline for assessing the improvement percentage). The same procedure can be applied on each one of the components’ candidates that are used in the family (i.e. \( C_{g(1-2)}^1, C_{g}^1 \)). In this way, we can quantify the percentage of improvement in the values of each commonality index which the component under consideration can achieve. Table 10 shows the improvement of each component on the commonality measurements. When some components have the same improvements percentage, other indices values can offer more insights to the designer. For example, \( C_3^3, C_4^4 \) and \( C_7^7 \) have the same impact on TCCI (11.11%) and CI (13.79%), but by looking to the improvement in SCI(\( c_3^3 \)), we can rank these components as: \( C_7^7 \) (41.13%), \( C_3^3 \) (34.12%), and the last is \( C_4^4 \) (26.62%).

### Table 7. Four Designs for DLCS family

<table>
<thead>
<tr>
<th>Component</th>
<th>Design 2</th>
<th>Design 3</th>
<th>Design 4</th>
<th>Design 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P1 P2 P3 P4 P5</td>
<td>P1 P2 P3 P4 P5</td>
<td>P1 P2 P3 P4 P5</td>
<td>P1 P2 P3 P4 P5</td>
</tr>
<tr>
<td>( C_1^1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_1^1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_2^2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{g(1-2)}^1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_1^1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_2^2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_3^3 )</td>
<td>x x x x x</td>
<td>x x x x x</td>
<td>x x x x x</td>
<td>x x x x x</td>
</tr>
<tr>
<td>( C_{g(1-5)}^3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{g(1-5)}^4 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_1^5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_2^5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_3^5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_4^5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{g(2-5)}^6 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_1^6 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_2^6 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_3^6 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_4^6 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{g(2-5)}^6 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_1^7 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{g(1-5)}^7 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{g(1-5)}^8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_1^8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_2^8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_3^8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_4^8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_5^8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( C_{g(3-5)}^8 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 8. Commonality indices for the Family Designs

<table>
<thead>
<tr>
<th>Index</th>
<th>Design 1</th>
<th>Design 2</th>
<th>Design 3</th>
<th>Design 4</th>
<th>Design 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI</td>
<td>1</td>
<td>1.850</td>
<td>1.850</td>
<td>2.055</td>
<td>2.177</td>
</tr>
<tr>
<td>TCCI</td>
<td>0</td>
<td>0.472</td>
<td>0.472</td>
<td>0.527</td>
<td>0.556</td>
</tr>
<tr>
<td>CI</td>
<td>0</td>
<td>0.586</td>
<td>0.586</td>
<td>0.655</td>
<td>0.689</td>
</tr>
<tr>
<td>C_piece</td>
<td>0</td>
<td>0.212</td>
<td>0.212</td>
<td>0.263</td>
<td>0.294</td>
</tr>
<tr>
<td>SCI (c)</td>
<td>1</td>
<td>2.833</td>
<td>3.095</td>
<td>3.185</td>
<td>3.249</td>
</tr>
<tr>
<td>C_Cost</td>
<td>0</td>
<td>0.458</td>
<td>0.524</td>
<td>0.546</td>
<td>0.562</td>
</tr>
</tbody>
</table>

Table 9. Comparison of component’s cost between the Initial Design and the Four Designs

<table>
<thead>
<tr>
<th>Cost ($) / Component</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design 1</td>
<td>9,043</td>
<td>8,413</td>
<td>23,050</td>
<td>17,556</td>
<td>5,094</td>
<td>12,560</td>
<td>27,727</td>
<td>16,770</td>
<td>120,213</td>
</tr>
<tr>
<td>Design 2</td>
<td>8,365</td>
<td>8,471</td>
<td>9,216</td>
<td>7,320</td>
<td>5,094</td>
<td>9,276</td>
<td>10,600</td>
<td>10,729</td>
<td>72,353</td>
</tr>
<tr>
<td>Design 3</td>
<td>9,043</td>
<td>8,413</td>
<td>9,216</td>
<td>7,320</td>
<td>5,094</td>
<td>9,276</td>
<td>10,600</td>
<td>10,729</td>
<td>69,691</td>
</tr>
<tr>
<td>Design 4</td>
<td>8,365</td>
<td>8,413</td>
<td>9,216</td>
<td>7,320</td>
<td>5,094</td>
<td>9,276</td>
<td>10,600</td>
<td>10,729</td>
<td>69,012</td>
</tr>
<tr>
<td>Design 5</td>
<td>8,365</td>
<td>8,413</td>
<td>9,216</td>
<td>7,320</td>
<td>5,094</td>
<td>9,276</td>
<td>10,600</td>
<td>10,729</td>
<td>69,070</td>
</tr>
</tbody>
</table>

Table 10. Impact of each component (in Design 5) on the values of commonality indices (compared to the Initial Design)

<table>
<thead>
<tr>
<th>Index</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
<th>C8</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCI</td>
<td>5.71%</td>
<td>2.78%</td>
<td>12.12%</td>
<td>12.12%</td>
<td>0%</td>
<td>8.8%</td>
<td>12.12%</td>
<td>5.71%</td>
</tr>
<tr>
<td>TCCI</td>
<td>5.56%</td>
<td>2.78%</td>
<td>11.11%</td>
<td>11.11%</td>
<td>0%</td>
<td>8.3%</td>
<td>11.11%</td>
<td>5.56%</td>
</tr>
<tr>
<td>CI</td>
<td>6.9%</td>
<td>3.45%</td>
<td>13.79%</td>
<td>13.79%</td>
<td>0%</td>
<td>10.34%</td>
<td>13.79%</td>
<td>6.90%</td>
</tr>
<tr>
<td>C_piece</td>
<td>1.43%</td>
<td>0.69%</td>
<td>3.03%</td>
<td>3.03%</td>
<td>0%</td>
<td>2.2%</td>
<td>3.03%</td>
<td>1.43%</td>
</tr>
<tr>
<td>SCI (c)</td>
<td>4%</td>
<td>3.79%</td>
<td>34.12%</td>
<td>26.62%</td>
<td>0%</td>
<td>19.44%</td>
<td>41.13%</td>
<td>12.96%</td>
</tr>
<tr>
<td>C_Cost</td>
<td>1%</td>
<td>0.95%</td>
<td>8.53%</td>
<td>6.66%</td>
<td>0%</td>
<td>4.8%</td>
<td>10.28%</td>
<td>3.24%</td>
</tr>
</tbody>
</table>

8. Conclusion

SPLE is a paradigm for achieving large scale reuse through capturing the commonalities and variabilities. The decision about commonality occurs early in the development process and inherently encompasses the subsequent software developments. Therefore, it is critical to have an analytical measure of the effectiveness of a given product family design on the commonality in addition to the other quantities (i.e., development cost).

To serve this purpose, our current work proposes how commonality can be measured in SPLE. We have shown that bringing the manufacturing knowledge to software product lines can be a great help to researchers and practitioners in measuring the degree of commonality. The results show that the degree of commonality of software families as measured by the analyzed indices has a significant effect on the development cost. Those indices provide a method to directly relate the degree of component commonality to various dependent variables which represent some quantified benefits of the SPL. Future research should look at other contributors to the total cost of the system (i.e., evolution and maintenance costs) and focus more on mapping commonality to software quality aspects. Furthermore, this focused analysis was able to highlight some promising trends. Still, the findings of this paper are within the context of the case study. Therefore, like any other empirical studies, additional experimentations on real applications should proceed for further validation and to enable the generalization of trend relationships between the value of the commonality indices and the development cost.

References


Hamad I. Alsawalqah is a student of PhD Program in Department of Information and Communications Engineering at Korea Advanced Institute of Science and Technology (KAIST). He received his B.S. in Computer Information Systems from University of Jordan in spring 2004. He received his M.A in Management Information Systems from Amman Arab University for Graduate Studies in spring 2006. In summer 2008, he received his second M.A in Software Engineering from KAIST. His research interests include SPL and Product Management.

Sungwon Kang received his BA from Seoul National University, Korea, in 1982 and received his MS and PhD in computer science from the University of Iowa, USA, in 1989 and 1992, respectively. From 1993, he was a principal researcher of Korea Telecom R & D Group until October 2001 when he joined KAIST and is currently an associate professor of the university. Since 2003, he has been an adjunct faculty member of Carnegie-Mellon University, USA, for the Master of Software Engineering Program. He is the editor of Korean Journal of Software Engineering Society. His current research areas include software architecture, software product line, software modeling and analysis, and software testing.

Bashar A. Al-Shboul is an assistant professor at University of Jordan, Jordan. He received his B.S. in Information Technology from Al-Balqa'a Applied University, Jordan. He received his M.S. in Computer Science from the University of Jordan. In 2012, he received his Ph.D. in Information & Telecommunications Technology from Korea Advanced Institute of Science and Technology. His research interests include Query Expansion, Legal Document Analysis, Topic Mining, Social Media Analysis, Trend Detection, and Syntactic & Semantic Plagiarism Detection.

Jihyun Lee is an assistant professor at Daejeon University, Korea. She received the BS degree in Information and Communications Engineering from the Chonbuk National University and the MS and the PhD in Computer Sciences from the Chonbuk National University. She worked as a research assistant professor at KAIST from 2005 to 2011. Her research interests include SPL, software testing, software architecture, and business process maturity.
## APPENDIX: Computation of Commonality Indices for Design 5

<table>
<thead>
<tr>
<th>Component</th>
<th>(C_{\text{piece}})</th>
<th>(C_{\text{Cost}})</th>
<th>(SCl^{(C)})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (C_1^{g(2-3)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2 (C_4^{g(2-3)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3 (C_3^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4 (C_2^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>5 (C_3^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>6 (C_5^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>7 (C_6^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>8 (C_1^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>9 (C_4^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>10 (C_7^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>11 (C_3^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>12 (C_2^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>13 (C_1^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>14 (C_0^{g(1-5)})</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

### Index Value

\[
\begin{align*}
DCI & = \frac{37}{17} = 2.177 \\
TCCI & = 1 - \frac{17 - 1}{37 - 1} = 0.556 \\
CI & = 1 - \frac{17 - 8}{37 - 8} = 0.689 \\
C_{\text{piece}} & = \frac{5.0}{17} = 0.294 \\
C_{\text{Cost}} & = \frac{38849}{69070} = 0.5625 \\
SCI^{(C)} & = \frac{224464}{69070} = 3.249
\end{align*}
\]
A Polynomial-Time Algorithm for Checking the Equivalence for Real-Time Deterministic Restricted One-Counter Transducers Which Accept by Final State

Mitsuo Wakatsuki †
The University of Electro-Communications, Japan

Etsuji Tomita ‡
The University of Electro-Communications, Japan

Tetsuro Nishino §
The University of Electro-Communications, Japan

Abstract
We are concerned with a subclass of deterministic pushdown transducers, called deterministic restricted one-counter transducers (droct’s), and study the equivalence problem for real-time droct’s which accept by final state. After providing some properties of these droct’s, we present a polynomial-time algorithm for checking the equivalence for these droct’s.

Keywords: equivalence problem, deterministic pushdown transducer, deterministic restricted one-counter transducer, polynomial-time algorithm, formal language theory

1. Introduction
One of the most interesting questions in formal language theory is the equivalence problem for deterministic pushdown automata (dpda’s) and the corresponding deterministic context-free grammars. Although Sénizergues [4] has proved that the equivalence problem for any pair of dpda’s is decidable, his algorithm is very complicated and is hard to be implemented. A checking the equivalence for some dpda’s can play an important role in the learning process for these dpda’s [5]. From a practical point of view, it is desirable to have a polynomial-time algorithm for checking the equivalence.

A deterministic one-counter automaton (doca) is a dpda having only one stack symbol, with the exception of a bottom-of-stack marker. A deterministic restricted one-counter automaton (droca) is a dpda which has only one stack symbol. The class of languages accepted by droca’s is a proper subclass of that of languages accepted by doca’s. Moreover, the class of languages accepted by droca’s is proper subclass of that of languages accepted by doca’s. However, the class of languages accepted by droca’s which accept by final state properly contains the class of regular languages. Valiant has shown that the equivalence problem for doca’s is decidable in single exponential time [9][10] and the inclusion problem for doca’s is undecidable in exponential space [9]. On the other hand, Higuchi et al. have presented polynomial-time algorithms for checking the inclusion (also the equivalence) for droca’s which accept by empty stack [2] and for real-time droca’s which accept by final state [3].

A deterministic pushdown transducer (dpdt) is a dpda provided with outputs. The equivalence problem for dpdt’s is essentially more difficult than for dpda’s. In this paper, we are concerned with a subclass of dpdt’s called deterministic restricted one-counter transducers (droct’s), which are droca’s provided with outputs, and present a new direct branching algorithm for checking the equivalence for a pair of real-time droct’s which accept by final state. It is based upon the algorithms for checking the equivalence of dpda’s or dpdt’s in some classes [6]–[8] and of strict droca’s [2]. The worst-case time complexity of our algorithm is polynomial in the description length of these droct’s.

2. Definitions and Notation
We assume that the reader is familiar with the basics of automata and formal language theory. Our definitions and notation are based on Refs. [6]–[8].

Definition 1 A deterministic pushdown transducer (dpdt for short) which accepts by final state is denoted by $T = (Q, \Gamma, \Sigma, \Delta, \mu, q_0, Z_0, F)$, where $Q$, $\Gamma$, $\Sigma$, $\Delta$, $\mu$ are the finite sets of states, stack symbols, input symbols, output symbols, and transition-output rules respectively, $q_0 (\in Q)$ is the initial state, $Z_0 (\in \Gamma)$ is the initial stack symbol, and $F (\subseteq Q)$ is the set of final states. We denote an empty string in $\Gamma^*$. $\Sigma^*$ by $\varepsilon$.

The set of transition-output rules is a set of rules of the form $(p, A)^{a/z}_\rightarrow(q, \theta)$ with $p, q \in Q$, $A \in \Gamma$, $a \in \Sigma \cup \{\varepsilon\}$, $z \in \Delta^*$, $\theta \in \Gamma^*$, which satisfies the following conditions:

(i) If $(p, A)^{a/z}_\rightarrow(q, \theta)$ with $a \in \Sigma \cup \{\varepsilon\}$ is in $\mu$, then $\mu$ contains no rule of the form $(p, A)^{a/z}_\rightarrow(r, \gamma)$ for any $(r, \gamma) \neq (q, \theta), z' \neq z$.

(ii) If $(p, A)^{\varepsilon/z}_\rightarrow(q, \theta)$ is in $\mu$, then $\mu$ contains no rule of the form $(p, A)^{a/z'}_\rightarrow(r, \gamma)$ for any $a \in \Sigma$, $z' \in \Delta^*$, $r \in Q$, $\gamma \in \Gamma^*$. Such a rule as $(p, A)^{\varepsilon/z}_\rightarrow(q, \theta)$ is called an $\varepsilon$-rule.
The deterministic pushdown automaton (dpda for short) $M = (Q, \Gamma, \Sigma, \delta, q_0, Z_0, F)$ with

$$
\delta = \{(p, A) \xrightarrow{\alpha}(q, \theta) \mid (p, A)^{\alpha/z}(q, \theta) \in \mu, z \in \Delta^* \}
$$

is called the associated dpda for the above dpdt $T$, and is just as in Ref. [6], Definition 2.1, pp.190-191. A dpdt or a dpda is said to be real-time if it has no $\varepsilon$-rules.

**Definition 2** A dpda $M$ (respectively, a dpdt $T$) is said to be a deterministic restricted one-counter automaton (transducer, resp.) (droca (droot, resp.) for short) if $\Gamma = \{Z_0\}$ (Cf. [1], p.201). When the droca $M$ is the associated dpda for the droot $T$, $M$ is called the associated droca for $T$.

**Definition 3** A configuration $(p, \alpha)$ of the dpdt $T$, with its associated dpda $M$, is an element of $Q \times \Gamma^*$, where the leftmost symbol of $\alpha$ is the top symbol on the stack. In particular, $(q_0, Z_0)$ is called the initial configuration. The height of a configuration $(p, \alpha)$ is $|\alpha|$. Here, for a string $\alpha$, $|\alpha|$ denotes the length of $\alpha$. Moreover, for a set $S$, $|S|$ denotes the cardinality of $S$.

**Definition 4** The dpdt $T$, with its associated dpda $M$, makes a move $(p, A\omega) \xrightarrow{a/z}(q, \theta \omega)$ for any $\omega \in \Gamma^*$ iff $\mu$ contains a rule $(p, A)^{a/z}(q, \theta)$ with $a \in \Sigma \cup \{\varepsilon\}$. A sequence of such moves through successive configurations as

$$(pi_i, \alpha'_i \alpha''_i) \xrightarrow{a/z_i}(pi_{i+1}, \alpha'_{i+1} \alpha''_{i+1}), 1 \leq i \leq m,$$

is called a derivation, and is written as

$$(p_1, \alpha_1 | \alpha''_1) \xrightarrow{x/y}(m) (p_{m+1}, \alpha'_{m+1} | \alpha''_{m+1}),$$

where $x = a_1 a_2 \cdots a_m$ and $y = z_1 z_2 \cdots z_m$, or simply

$$(p_1, \alpha_1 | \alpha''_1) \xrightarrow{x/y}(m) (p_{m+1}, \alpha'_{m+1} | \alpha''_{m+1}),$$

if

$$(p_1, \alpha_1 | \alpha''_1) \xrightarrow{x/y}(M) (p_{m+1}, \alpha'_{m+1} | \alpha''_{m+1})$$

as in Ref. [6], Definition 2.3, pp.191-192.

**Definition 5** For a configuration $(p, \alpha) \in Q \times \Gamma^*$ of the dpdt $T$ which accepts by final state, define

$$\text{TRANS}(p, \alpha) = \{x/y \in \Sigma^* \times \Delta^* \mid (p, \alpha)^{x/y}\gamma}(q, \beta)\text{ for some } q \in F, \beta \in \Gamma^*\},$$

and

\begin{align*}
\text{FIRST}(p, \alpha) = \{a \in \Sigma \mid (p, A)^{a/z}(q, \theta) \in \mu, \\
\alpha = A\alpha'', A \in \Gamma, \alpha'' \in \Gamma^*, \\
\text{for some } q \in Q, \theta \in \Gamma^*, z \in \Delta^*\},
\end{align*}

and

\begin{align*}
\text{FIRST}_{\text{free}}(p, \alpha) = \begin{cases}
\{a \in \text{FIRST}(p, \alpha) \mid (p, \alpha)^{a/z}(q, \beta), \\
L(q, \beta) \neq \emptyset, \text{ for some } q \in Q, \beta \in \Gamma^*, \\
z \in \Delta^*\} & \text{if } p \notin F, \\
\{\varepsilon\} \cup \{a \in \text{FIRST}(p, \alpha) \mid (p, \alpha)^{a/z}(q, \beta), \\
L(q, \beta) \neq \emptyset, \text{ for some } q \in Q, \beta \in \Gamma^*, \\
z \in \Delta^*\} & \text{if } p \in F.
\end{cases}
\end{align*}

**Definition 7** For $w, h, t \in \Delta^*$, let $h^{-1}w = t$ if $w = h t$, and $wt^{-1} = h$ if $w = h t$. Then for $\Delta$, let $\Delta^* = \{h^{-1} | h \in \Delta^*\}, \Delta^{\varepsilon*} = \Delta^* \cup \Delta^{-*}$, where $\Delta^* \cap \Delta^{-*} = \{\varepsilon\}$, and for $h \in \Delta^{\varepsilon*}$, let $(h^{-1})^{-1} = h$. Moreover, let $\Delta^{-*} = \Delta^{\varepsilon*} - \{\varepsilon\}$. For $k^{-1} \in \Delta^{-*}$ with $k \in \Delta^*$, define $|k^{-1}| = -|k|$. Then for $h \in \Delta^{\varepsilon*}$, define

$$||h|| = \begin{cases}
|h| & \text{if } h \in \Delta^*, \\
|k| & \text{if } h = k^{-1} \in \Delta^{-*}.
\end{cases}$$

**Definition 8** Let $(p, \alpha)$ be a configuration of a dpdt $T_1$ which accepts by final state, $(\bar{p}, \beta)$ be that of a dpdt $T_2$ which accepts by final state, and $h \in \Delta^{\varepsilon*}$. If $\text{TRANS}(p, \alpha) = h \text{TRANS}(\bar{p}, \beta) = \{x/y \mid x/y \in \text{TRANS}(p, \alpha)\}$, then it is written as $(p, \alpha) \equiv h(\bar{p}, \beta)$. Here, if $h \in \Delta^{-*}$, then $(p, \alpha) \equiv h(\bar{p}, \beta)$ is the same as $k(p, \alpha) \equiv (\bar{p}, \beta)$ with $k = h^{-1} \in \Delta^*$ which stands for $\{x/y \mid x/y \in \text{TRANS}(p, \alpha)\} = \text{TRANS}(\bar{p}, \beta)$. Such a formula as $(p, \alpha) \equiv h(\bar{p}, \beta)$, $h \in \Delta^{\varepsilon*}$ or $k(p, \alpha) \equiv (\bar{p}, \beta), k \in \Delta^*$, is called an equivalence equation.

If $\text{TRANS}(T_1) = \text{TRANS}(T_2)$, then the two dpdt’s are equivalent, and it is written as $T_1 \equiv T_2$. Otherwise, $T_1 \neq T_2$. International Journal of Computer & Information Science, Vol. 14, No. 2, December 2013 46
3. Basic Properties and Propositions

For a real-time droct \( T = (Q, \Gamma, \Sigma, \Delta, \mu, q_0, Z_0, F) \), define \( \tau = \text{Max}\{|z| \mid (p, A) \xrightarrow{\alpha \beta}(q, \theta) \in \mu \} \) and \( \rho = \text{Max}\{|\theta| \mid (p, A) \xrightarrow{\alpha \beta}(q, \theta) \in \mu \} \). Without loss of generality, we may assume that \( \rho \leq 2 \), i.e. stack height increases by at most one per one move.

3.1 Basic Properties of Real-Time DROCT’s

Which Accept by Final State

Lemma 1. Let \( T = (Q, \Gamma, \Sigma, \Delta, \mu, q_0, Z_0, F) \) be a real-time droct which accepts by final state, and \( (p, \alpha) \in Q \times \Gamma^* \) be a configuration of \( T \). Then, \( \tau \leq |Q| + 2 \). If there exists a derivation \( (p, \alpha) \xrightarrow{T}(q, \beta) \) for some \( q \in Q, \beta \in \Gamma^* \), then \( \tau \leq |Q| + 2 \). Then, the worst-case time complexity of this procedure is polynomial with respect to the size of \( T \).

Proof: (i): \( \tau \leq |Q| + 2 \). If there exists a derivation \( (p, \alpha) \xrightarrow{T}(q, \beta) \) for some \( q \in Q, \beta \in \Gamma^* \), then \( \tau \leq |Q| + 2 \). Then, the worst-case time complexity of this procedure is polynomial with respect to the size of \( T \).

Lemma 2. Let \( T = (Q, \Gamma, \Sigma, \Delta, \mu, q_0, Z_0, F) \) be a real-time droct which accepts by final state, and \( (p, \alpha) \in Q \times \Gamma^* \) be a configuration of \( T \). If there exists a derivation \( (p, \alpha) \xrightarrow{T}(q, \beta) \) for some \( x \in \Sigma^*, y \in \Delta^*, q \in Q, \beta \in \Gamma^* \), then there exists a derivation \( (p, \alpha) \xrightarrow{x/y T}(q, \beta) \) with \( n \leq |Q| (|Q| - 1) \) for some \( x \in \Sigma^*, y \in \Delta^* \).

Proof: It is clear because \( \Gamma = \{Z_0\} \). (See [3], Lemma 3.1, p.94 for the details).

From Lemma 2, we can construct a procedure to check whether any configuration \( (p, \alpha) \in Q \times \Gamma^* \) of \( T \) is live or not since for all configurations \( (r, \gamma) \in Q \times \Gamma^* \) such that \( (p, \alpha) \xrightarrow{x/y T}(r, \gamma) \) with \( |x| \leq |Q| (|Q| - 1) \) for some \( x \in \Sigma^* \) and \( y \in \Delta^* \), it suffices to check whether \( r \in F \) or not. This procedure can be also carried out in a polynomial-time with respect to the size of \( T \).

3.2 The Basic Proposition Concerning the Equivalence

We shall check the equivalence of two real-time droct’s \( T_1 = (Q_1, \Gamma_1, \Sigma, \Delta, \mu_1, q_{01}, Z_{01}, F_1) \) and \( T_2 = (Q_2, \Gamma_2, \Sigma, \Delta, \mu_2, q_{02}, Z_{02}, F_2) \) which accept by final state, whose associated droca’s are \( M_1 \) and \( M_2 \) respectively. We are only concerned with the case where \( L(T_i) \neq \emptyset \) for \( i = 1, 2 \). Let \( \tau = \text{Max}\{\tau_1, \tau_2\} \), where \( \tau_i \) are constants depending on \( T_i \) defined as above.

To begin with, we give the most elementary proposition concerning outputs.

Proposition 1 (i) Suppose \( T_1 \equiv T_2 \) holds and

\[
(q_{01}, Z_{01}) \xrightarrow{w/u_{T_1}} (p, \alpha) \text{ with } L(p, \alpha) \neq \emptyset,
\]

\[
(q_{02}, Z_{02}) \xrightarrow{w/u_{T_2}} (\bar{p}, \beta) \text{ with } L(\bar{p}, \beta) \neq \emptyset,
\]

for some \( w \in \Sigma^*, w_1, w_2 \in \Delta^*, p \in Q_1, \bar{p} \in Q_2, \alpha \in \Gamma_1^* \) and \( \beta \in \Gamma_2^* \), then it holds that

\[
w_1 h = w_2 \text{ for some } h \in \Delta^{\pm*}
\]

and

\[
(p, \alpha) \equiv (\bar{p}, \beta).
\]

(ii) If we have, in addition to (i), another pair of derivations \( (q_{01}, Z_{01}) \xrightarrow{w'/u_{T_1}} (p', \alpha') \) with \( L(p', \alpha') \neq \emptyset \) and \( (q_{02}, Z_{02}) \xrightarrow{w'/u_{T_2}} (\bar{p}', \beta') \) with \( L(\bar{p}', \beta') \neq \emptyset \) for some \( w' \in \Sigma^*, w'_1, w'_2 \in \Delta^* \) such that \( w'_1 h' = w'_2 \) for some \( h' \in \Delta^{\pm*}, \alpha' \in \Gamma_1^{*}, \beta' \in \Gamma_2^{*} \), and hence \( (p, \alpha') \equiv (\bar{p}', \beta') \), then \( h = h' \).

Proof: These properties can be easily derived from Definitions 5, 6, and 8, and Lemma 1 (ii).

The above \( h \in \Delta^{\pm*} \) in eq.(3) and eq.(4) is to compensate the difference between the two outputs, and is called an output compensating part. From (ii) in Proposition 1, the output compensating part is unique for each equivalence equation as eq.(4) when \( T_1 \equiv T_2 \).

Note that although the stack symbol \( A \in \Gamma_i \) (\( i = 1, 2 \)) are frequently used below, the stack symbol of \( T_i \) is only \( Z_{0i} \) and \( Z_{0i} = A \).

Definition 9 Define for \( (p, A) \in Q_i \times \Gamma_i \) \( i = 1, 2 \) such that \( N(p, A) \neq \emptyset \),

\[
k_i(p, A) = \text{Max}\{\text{Min}\{|x| \mid (p, A) \xrightarrow{u T}(q, \epsilon), x \in \Sigma^*, y \in \Delta^* \mid q \in Q_i\}.
\]

Define then \( k_i (i = 1, 2) \) for the real-time droct \( T_i \) as

\[
k_i = \text{Max}\{k_i(p, A) \mid N(p, A) \neq \emptyset, p \in Q_i, A \in \Gamma_i\}.
\]

Lemma 3 Suppose \( T_1 \equiv T_2 \) holds and

\[
(q_{01}, Z_{01}) \xrightarrow{x_1/u_{T_1}} (p, \alpha) \xrightarrow{x_2/u_{T_1}} (p, \alpha'),
\]

\[
(q_{02}, Z_{02}) \xrightarrow{x_1/v_{T_2}} (\bar{p}, \beta) \xrightarrow{x_2/v_{T_2}} (\bar{p}, \beta')
\]
for some \( x_0 \in \Sigma^* \), \( x \in \Sigma^+ \), \( u_0, u, v_0, v \in \Delta^* \), \( p \in Q_1 \), \( \bar{p} \in Q_2 \), \( \alpha \in \Gamma^+_1 \), \( \alpha' \in \Gamma^+_1 \), \( \beta \in \Gamma^+_2 \), \( \beta' \in \Gamma^+_2 \).

(i) If \( N(p, \alpha) \neq \emptyset \), then the following (a) and (b) hold.

(a) If \( |Q_1| \leq |\alpha| \leq |\alpha'| \), then \( |\beta| \leq |\beta'| \).

(b) If \( |\beta| \leq |\beta'| \), then \( |\alpha| \leq |\alpha'| \).

(ii) If \( N(\bar{p}, \beta) \neq \emptyset \), then the following (c) and (d) hold.

(c) If \( |Q_2| \leq |\beta| \leq |\beta'| \), then \( |\alpha| \leq |\alpha'| \).

(d) If \( |\alpha| \leq |\alpha'| \), then \( |\beta| \leq |\beta'| \).

Proof: By symmetry, we prove only the case (i). From the assumption that \( T_1 \equiv T_2 \) and \( N(p, \alpha) \neq \emptyset \), the properties (a) and (b) can be derived from Lemma 1 (i) and (ii). (Cf. [2], Lemma 3.2, pp.306-307.) \( \square \)

**Lemma 4** Suppose \( T_1 \equiv T_2 \) holds and

\[
(q_{01}, Z_{01}) \xrightarrow{\frac{x}{w}} (p, \alpha), \quad (q_{02}, Z_{02}) \xrightarrow{\frac{x}{v}} (\bar{p}, \beta)
\]

for some \( x \in \Sigma^* \), \( u, v \in \Delta^* \), \( p \in Q_1 \), \( \bar{p} \in Q_2 \), \( \alpha \in \Gamma^+_1 \), \( \beta \in \Gamma^+_2 \). Then, it holds that \( \text{FIRST}_{live}(p, \alpha) = \text{FIRST}_{live}(\bar{p}, \beta) \).

Proof: From the assumption that \( T_1 \equiv T_2 \), it follows from Definition 6. (Cf. [2], Lemma 3.3, p.307.) \( \square \)

4. The Equivalence Checking Algorithm

The equivalence checking is carried out by developing step by step a so-called comparison tree as in Refs. [6]–[8].

At the initial stage, the comparison tree contains only the root labeled \( (q_{01}, Z_{01}) \equiv (q_{02}, Z_{02}) \) which is said to be in \textit{unchecked} status. In each step, the algorithm considers a node labeled \( (p, \alpha) \equiv h(\bar{p}, \beta) \) such that eq.(1) and eq.(2) with eq.(3) for some \( w \in \Sigma^* \), \( w_1, w_2 \in \Delta^* \), and tries to prove or disprove this equivalence. In case \( \text{FIRST}_{live}(p, \alpha) = \text{FIRST}_{live}(\bar{p}, \beta) \) such that \( \epsilon \), if \( h = \epsilon \) then we turn the node to be in \textit{checked} status. Otherwise, i.e., \( h \neq \epsilon \), conclude that \( "T_1 \neq T_2" \). Also, if another internal node with the same label has appeared elsewhere in the tree, then we turn the above node to be \textit{checked}. From Proposition 1 (ii), if another internal node labeled \( (p, \alpha') \equiv h'(\bar{p}, \beta') \) such that \( h' \neq h \) has appeared elsewhere in the tree, then conclude that \( "T_1 \neq T_2" \). Otherwise, we expand it by \textit{branching, skipping, or stack reduction}, which will be described below.

4.1 Branching

Branching is a basic step of developing the comparison tree. It is based on moves of configurations by one step.

**Lemma 5** The equivalence equation eq.(4) holds iff the following conditions (i), (ii) and (iii) hold.

(i) \( \text{FIRST}_{live}(p, \alpha) = \text{FIRST}_{live}(\bar{p}, \beta) \).

(ii) For each \( a_i \in \text{FIRST}_{live}(p, \alpha) - \{ \epsilon \} = \{ a_1, a_2, \ldots, a_l \} \subseteq \Sigma \) (\( i = 1, 2, \ldots, l \)), let

\[
(p, \alpha) \xrightarrow{T_1} (p, \alpha_i) \quad \text{and} \quad (\bar{p}, \beta) \xrightarrow{T_2} (\bar{p}, \beta_i)
\]

for some \( (p_i, \alpha_i) \in Q_1 \times \Gamma^+_1 \), \( (\bar{p}_i, \beta_i) \in Q_2 \times \Gamma^+_2 \).

(iii) Concerning the above condition (ii), it holds that \( \alpha_i \equiv \beta_i \).

Proof: It follows from Proposition 1 (i). \( \square \)

The checking whether conditions (i) and (ii) in Lemma 5 hold or not is named \textit{output branch checking} to the node labeled eq.(4) in question. When it is verified to hold, the checking is said to be \textit{successful}. Then we expand the above node to have \( l \) sons in \textit{unchecked} status labeled by (iii), with edges labeled \( z_i \| a_i/\bar{z}_i \), \( i = 1, 2, \ldots, l \), and we turn the node in question to be \textit{checked}. The step of developing the comparison tree in this way is named \textit{branching} to the node in question. If condition (i) or (ii) does not hold, conclude that \( "T_1 \neq T_2" \).

When the branching has been applied to the node in question, the number of its sons is at most \( |\Sigma| \).

4.2 Stack Reduction

From Lemma 1, for any node labeled \( (p, \alpha) \equiv h(\bar{p}, \beta) \), if \( N(p, \alpha) = \emptyset \), then there exists \( \alpha' \in \Gamma^+_1 \) such that \( N(p, \alpha') = \emptyset \) and \( \text{TRANS}(p, \alpha') = \text{TRANS}(\bar{p}, \beta) \) (i.e. \( (p, \alpha') \equiv (p, \alpha) \)) with \( 1 \leq |\alpha'| \leq |Q_1| \). Similarly, if \( N(\bar{p}, \beta) = \emptyset \), there exists \( \beta' \in \Gamma^+_2 \) such that \( N(\bar{p}, \beta') = \emptyset \) and \( \text{TRANS}(\bar{p}, \beta') = \text{TRANS}(\bar{p}, \beta) \) (i.e. \( (\bar{p}, \beta') = (\bar{p}, \beta) \)) with \( 1 \leq |\beta'| \leq |Q_2| \).

**Definition 10** Let \( (p, \alpha) \equiv h(\bar{p}, \beta) \) be a label of the node in question. If \( N(p, \alpha) = \emptyset \), let \( \alpha_0 \) be the \textit{shortest} stack such that \( N(p, \alpha_0) = \emptyset \). Otherwise, let \( \alpha_0 = \alpha \). Moreover, if \( N(\bar{p}, \beta) = \emptyset \), let \( \beta_0 \) be the \textit{shortest} stack such that \( N(\bar{p}, \beta_0) = \emptyset \). Otherwise, let \( \beta_0 = \beta \). If it holds that \( |\alpha_0| < |\alpha| \) or \( |\beta_0| < |\beta| \), we say that the stack reduction is \textit{applicable} to the node in question.

When stack reduction is applicable to the node in question, we expand it to have only one son labeled by \( (p, \alpha) \equiv h(\bar{p}, \beta) \) in \textit{unchecked} status, with edge labeled \( \epsilon \| \alpha/\beta \), where \( \lambda \) is a new symbol not in \( \Sigma^* \), and then we turn the node in question to be \textit{checked}. The step of developing the comparison tree in this way is named \textit{stack reduction} to the node in question.

When the stack reduction has applied to the node in question, the following (i) or (ii) holds:

(i) \( 1 \leq |\alpha_0| \leq |Q_1| \), \( |\alpha_0| < |\alpha| \) and \( |\beta_0| \leq |\beta| \),

(ii) \( |\alpha_0| \leq |\alpha| \), \( 1 \leq |\beta_0| \leq |Q_2| \) and \( |\beta_0| < |\beta| \).

4.3 Essential Checking

Let the comparison tree just having been constructed up to a certain stage be denoted by \( T(T_1 : T_2) \).

**Definition 11** If \( (p_1, \alpha_1 \gamma_1) \equiv h_1(\bar{p}_1, \alpha_1 \gamma_1) \) and \( (p_2, \alpha_2 \gamma_2) \equiv h_2(\bar{p}_2, \alpha_2 \gamma_2) \) are labels of two nodes in
$T(T_1 : T_2)$ which are connected by an edge labeled $u_1, x_1/v_1 \in \Delta^* \times (\Sigma \cup \{\lambda\})^* \times \Delta^*$ such that

$$(p_1, \alpha_1) = \frac{\sigma(x_1/v_1)}{T_1} (p_2, \beta_2)$$

with $u_1 h_2 = h_1 v_1, |\beta_2| \geq |\alpha_2|$, and $|\beta_2| \geq |\alpha_2|$, where $\sigma(x_1) \in \Sigma^*$ is the string obtained by eliminating all $\lambda$’s from $x_1$, then we write

$$(p_1, \alpha_1 \mid \gamma_1) \equiv h_1 (p_1, \alpha_1 \mid \gamma_1) \uparrow x_1/v_1$$

$$(p_2, \alpha_2 \mid \gamma_1) \equiv h_2 (p_2, \alpha_2 \mid \gamma_1)$$

A sequence of such father-son relations as

$$(p_i, \alpha_i \mid \gamma_i) \equiv h_i (p_i, \alpha_i \mid \gamma_i) \uparrow x_i/v_i$$

$$(p_{i+1}, \alpha_{i+1} \mid \gamma_i) \equiv h_{i+1} (p_{i+1}, \alpha_{i+1} \mid \gamma_i)$$

with $u_i h_{i+1} = h_i v_i$, for $i = 1, 2, \ldots, m$, is named a derivation path, and is written as

$$(p_1, \alpha_1 \mid \gamma_1) \equiv h_1 (p_1, \alpha_1 \mid \gamma_1) \uparrow x_1/v_1$$

$$(p_m+1, \alpha_m+1 \mid \gamma_1) \equiv h_m (p_m+1, \alpha_m+1 \mid \gamma_1)$$

where $x = x_1 x_2 \ldots x_m, u = u_1 u_2 \ldots u_m$, and $v = v_1 v_2 \ldots v_m$. Here, “$\uparrow$” may be omitted.

**Definition 12** Concerning a node labeled $(p, \alpha') \equiv h(\bar{p}, \beta')$ in $T(T_1 : T_2)$, the checking as to whether the following conditions (i) and (ii) hold or not is named the essential checking.

(i) $\text{FIRST}_{live}(p, \alpha') = \text{FIRST}_{live}(\bar{p}, \beta')$.

(ii) When there exists a derivation path to this node such that

$$(q_1, Z_1) \equiv (q_0, Z_0) \frac{u'(x'/v')}{T_1(T_1 : T_2)}$$

$$(p, \alpha) \equiv h(\bar{p}, \beta) \frac{u'(x'/v')}{T_1(T_1 : T_2)} ((p, \alpha') \equiv h(\bar{p}, \beta'))$$

for some $x' \in (\Sigma \cup \{\lambda\})^+, x'' \in (\Sigma \cup \{\lambda\})^+, \alpha' \in \Gamma_1^+, \alpha \in \Gamma_1^+$ and $\beta' \in \Gamma_2^+$, the following (a) and (b) hold.

(a) In the case where $N(p, \alpha) \neq \emptyset$:

(a1) If $|Q_1| \leq |\alpha| \leq |\alpha'|$, then $|\beta| \leq |\beta'|$.

(a2) If $|\beta| \leq |\beta'|$, then $|\alpha| \leq |\alpha'|$.

(b) In the case where $N(\bar{p}, \beta) \neq \emptyset$:

(b1) If $|Q_2| \leq |\beta| \leq |\beta'|$, then $|\alpha| \leq |\alpha'|$.

(b2) If $|\alpha| \leq |\alpha'|$, then $|\beta| \leq |\beta'|$.

The essential checking is applied to the root and all the newly added sons by branching or skipping in the comparison tree. If the essential checking fails for some node, then we conclude that “$T_1 \neq T_2$” since $T_1 \neq T_2$ holds from Lemmas 3 and 4.

### 4.4 Skipping

In order to prevent the comparison tree from growing larger and larger infinitely by successive application of branching or stack reduction steps, certain nodes are expanded by other steps of skipping. Let the comparison tree which has just been constructed up to a certain stage be denoted by $T(T_1 : T_2)$.

**Definition 13** Suppose that a node in question is labeled

$$(p, \omega_1, \alpha'') \equiv h(\bar{p}, \omega_2, \beta'')$$

where $\alpha = \omega_1 \alpha'' \land \beta = \omega_2 \beta''$ with $\alpha'' \neq \varepsilon$ or $\beta'' \neq \varepsilon$. We say that the prerequisite for skipping to it is satisfied if the tree $T(T_1 : T_2)$ contains a branching node labeled

$$(p, \omega_1) \equiv h(\bar{p}, \omega_2),$$

where $\omega_1 \in \Gamma_1^+$ and $\omega_2 \in \Gamma_2^+$ such that

$$(q, \gamma) \equiv (\bar{q}, \bar{\gamma})$$

and $\text{FIRST}_{live}(q, \gamma) = \text{FIRST}_{live}(\bar{q}, \bar{\gamma}) = \{\varepsilon\}$, where $\gamma = \varepsilon$ or $N(q, \gamma) = \emptyset$ with $1 \leq |\gamma| \leq |Q_1|$, and $\bar{\gamma} = \varepsilon$ or $N(\bar{q}, \bar{\gamma}) = \emptyset$ with $1 \leq |\bar{\gamma}| \leq |Q_2|$, for some $x \in (\Sigma \cup \{\lambda\})^+$, $u, v \in \Delta^*$ with $u = \varepsilon v q, q \in F_1, \bar{q} \in F_2$.

Now find a shortest string $\sigma(x_0) \in \Sigma^+$ such that

$$(p, \omega_1) \equiv h(\bar{p}, \omega_2) \frac{u_0, v_0}{T_1(T_1 : T_2)} (q, \gamma) \equiv (\bar{q}, \bar{\gamma})$$

and $\text{FIRST}_{live}(q, \gamma) = \text{FIRST}_{live}(\bar{q}, \bar{\gamma}) = \{\varepsilon\}$, where $\gamma = \varepsilon$ or $N(q, \gamma) = \emptyset$ with $1 \leq |\gamma| \leq |Q_1|$, and $\bar{\gamma} = \varepsilon$ or $N(\bar{q}, \bar{\gamma}) = \emptyset$ with $1 \leq |\bar{\gamma}| \leq |Q_2|$, for some $u_0, v_0 \in \Delta^*$, and check whether it is successful or not to have $u_0 = h v_0$. Then the skipping to the node in question is said to be applicable if the above checking is successful for every possible $(q, \gamma) \equiv (\bar{q}, \bar{\gamma})$ as above. A node labeled $(q, \gamma, \alpha'') \equiv (\bar{q}, \bar{\gamma}, \beta'')$ is defined to be a skipping-end from the node in question, and an edge label between them is defined to be $u_0 \frac{\sigma(x_0)}{T_1} v_0$. When skipping is applicable to the node in question, we expand it to have skipping-ends in unchecked status, then we turn the node in question to be skipping. The step of developing the comparison tree in this way is named skipping to the node labeled eq.(5) with respect to eq.(6). Whenever a new node is added to the comparison tree, we must apply skipping again to the node which has been already applied skipping, because it is possible that some new skipping-ends will be added to it afterward.

When the skipping has been applied to the node in question, the number of skipping-ends of it is at most $|F_1|(|Q_1| + 1) \times |F_2|(|Q_2| + 1)$. 

---

4.5 The Whole Algorithm

The whole algorithm is shown below. Here, the next node to be visited is chosen as the “smallest” of the unchecked or skipping nodes, where the size of a node labeled \((p, \alpha) = h(p, \beta)\) is the pair \((\text{Max}\{|\alpha|, |\beta|\}, \text{Min}\{|\alpha|, |\beta|\})\), under lexicographic ordering.

The Equivalence Checking Algorithm

Check whether \(N(p, \alpha) \neq \emptyset\) or not for each configuration \((p, \alpha) \in Q_1 \times \Gamma_1^+\) with \(1 \leq |\alpha| \leq |Q_1|\). Similarly, check whether \(N(\tilde{p}, \beta) \neq \emptyset\) or not for each configuration \((\tilde{p}, \beta) \in Q_2 \times \Gamma_2^+\) with \(1 \leq |\beta| \leq |Q_2|\).

Let the comparison tree consist of only a root labeled \((q_0, Z_0)\) in unchecked status. if the essential checking fails for the root then conclude that “\(T_1 \neq T_2\)” ; halt fiwhile the comparison tree contains an unchecked or a skipping node do if the comparison tree contains an unchecked node then let \(P\) be the smallest unchecked node else let \(P\) be the smallest skipping node fi suppose \(P\) is labeled \((p, \alpha) = h(\tilde{p}, \beta)\); if stack reduction is applicable to \(P\) then apply the stack reduction to \(P\); turn the status of \(P\) to be checked, while its newly added son is in unchecked else if \(\text{FIRST}_{\text{live}}(p, \alpha) = \text{FIRST}_{\text{live}}(\tilde{p}, \beta) = \{\varepsilon\}\) then if \(h = \varepsilon\) then turn \(P\) to checked else conclude that “\(T_1 \neq T_2\)” ; halt fi else if \((p, \alpha') = h'(\tilde{p}, \beta') (h' \neq h)\) appears as the label of another internal node then conclude that “\(T_1 \neq T_2\)” ; halt else if \((p, \alpha) \equiv h(\tilde{p}, \beta)\) appears as the label of another internal node then turn \(P\) to checked else if the prerequisite for skipping to \(P\) is satisfied then if skipping to \(P\) is applicable then apply the skipping to \(P\); if either any skipping-end has been newly added to \(P\) as its son, or any label of an edge from \(P\) has been changed by the above skipping then turn the status of \(P\) to be skipping, while its newly added sons are in unchecked; turn the status of all s-checked nodes to be skipping else turn the status of \(P\) to be s-checked fi apply the essential checking to all newly added sons of \(P\); if the essential checking fails for some son of \(P\) then conclude that “\(T_1 \neq T_2\)” ; halt fi else conclude that “\(T_1 \neq T_2\)” ; halt fi

Example 1

Let us apply our algorithm to the following pair of real-time dropt’s: \(T_1 = \{q_0, p_1, p_2, p_3\}, \{A\}, \{a, b, c, d\}, \{a, b\}, \mu_1, p_0, \{p_3\}\) and \(T_2 = \{q_0, q_1, q_2, q_3\}, \{B\}, \{a, b, c, d\}, \{a, b\}, \mu_2, q_0, \{q_3\}\), where

\[
\mu_1 = \{(p_0, A) \xrightarrow{a/b} (p_0, A^2), (p_0, A) \xrightarrow{b/\varepsilon} (p_1, \varepsilon), (p_1, A) \xrightarrow{b/a} (p_2, A), (p_2, A) \xrightarrow{d/a} (p_3, \varepsilon)\}
\]

and

\[
\mu_2 = \{(q_0, B) \xrightarrow{a/b} (q_0, B^2), (q_0, B) \xrightarrow{b/a} (q_1, B), (q_1, B) \xrightarrow{b/a} (q_2, \varepsilon), (q_2, B) \xrightarrow{d/a} (q_3, B^2)\}.
\]

Successive application of branching steps yields an intermediate tree containing early nodes numbered \(\mathfrak{G}_1 \equiv \mathfrak{G}_3 \equiv \mathfrak{G}_2 \equiv \mathfrak{G}_1 \equiv \mathfrak{G}_3 \equiv \mathfrak{G}_2 \equiv \mathfrak{G}_1 \equiv \mathfrak{G}_3 \equiv \mathfrak{G}_2\). When \(\mathfrak{G}(p_3, \varepsilon) \equiv (q_3, B^2)\) is visited, stack reduction is applied to yield its son \(\mathfrak{G}(p_3, \varepsilon) \equiv (q_3, B)\). And then, after successive application of branching steps, when \(\mathfrak{G}(p_2, A^2) \equiv a^2(q_2, B^2)\) is visited, stack reduction is similarly applied to yield its son \(\mathfrak{G}(p_2, A^2) \equiv a^2(q_2, B)\). After successive application of branching and stack reduction steps, when \(\mathfrak{G}(p_1, A^4 \cdot A) \equiv (q_0, B^4 \cdot B)\) is visited first, skipping is applied with respect to \(\mathfrak{G}(p_0, A^4) \equiv (q_0, B^4)\) to yield skipping-ends \(\mathfrak{G}(p_3, \varepsilon \cdot A) \equiv (q_3, B^2 \cdot B)\) and \(\mathfrak{G}(p_3, \cdot A) \equiv (q_3, B \cdot B)\). When the same node \(\mathfrak{G}(p_0, A^4 \cdot A) \equiv (q_0, B^4 \cdot B)\) is visited again afterwards, the tree is not changed any more by this step. Then the algorithm halts with the correct conclusion that “\(T_1 \equiv T_2\)”. (In fact, \(\text{TRANS}(T_1) = \text{TRANS}(T_2) = \{a^i b^j c^j (a/b) \xrightarrow{a^i \cdot j-1} a^2 \mid i \geq j \geq 1\})
5. Termination and Correctness

5.1 Termination of the Algorithm

In case the given real-time droct’s $T_1$ and $T_2$ are equivalent, we shall show that the algorithm halts in a finite number of steps, which is a polynomial in the description length of $T_1$ and $T_2$, concluding that “$T_1 \equiv T_2$”.

**Lemma 6** Suppose that $T_1 \equiv T_2$. Let $T(T_1 : T_2)$ be the comparison tree which has been constructed as far as possible, in which every node is either checked or s-checked status. Then the following properties hold:

(i) For any node labeled $(p, \alpha) \equiv h(\bar{p}, \beta)$ in $T(T_1 : T_2)$, it holds that $|\alpha| \leq S_1 = (|Q_1| |Q_2| + 1) + |Q_2|$, $|\beta| \leq S_2 = (|Q_1| |Q_2| + 1) + |Q_2|$, and $|\tau| \leq |Q_1| |Q_2| + 1$.

(ii) The number of nodes in $T(T_1 : T_2)$ is at most $\delta_T = |Q_1| |S_1| + 1 |Q_2| (S_2 + 1) \times (|\Sigma|, |F_1| |Q_1| + 1 |F_2| |Q_2| + 1) + 1$.

(iii) For any edge labeled $u \backslash v$ in $T(T_1 : T_2)$, it holds that $|x| \leq \delta_T$, $|u| \leq \tau_1 S_1$, and $|v| \leq \tau_2 S_1$, where $\delta_T = \max\{k_1 S_1, k_2 S_2\}$.

We can prove easily that, for any node in $T(T_1 : T_2)$, the time complexity of application of the branching, the skipping, and the stack reduction is polynomial with respect to $|Q_1| |Q_2|$, $|\Sigma|$, $|\mu_1|$, $|\mu_2|$, $k_1$, and $k_2$, respectively. Here, for $k_i (i = 1, 2)$, it has been shown that $k_i \leq 2 |Q_i|^3$ from Ref. [11], Lemma 10, pp. 1711-1712. Similarly, we can prove that the time complexity of the essential checking is polynomial with respect to the above constants. Also, for any node labeled $(p, \alpha) \equiv h(\bar{p}, \beta)$ in the tree, the time complexity of the checking whether $N(p, \alpha) \neq \emptyset$ or not, $N(\bar{p}, \beta) \neq \emptyset$ or not, $L(p, \alpha) \neq \emptyset$ or not, and $L(\bar{p}, \beta) \neq \emptyset$ or not, is polynomial with respect to the above constants, respectively. Therefore, the worst-case time complexity of the whole equivalence checking algorithm is polynomial with respect to $|Q_1| |Q_2|$, $|\Sigma|$, $|\mu_1|$, and $|\mu_2|$.

**5.2 Correctness of the Algorithm**

Now we show the correctness of our algorithm.

**Lemma 7** The algorithm halts in a finite number of steps with the correct conclusion.

**Proof:** From Lemma 6, it is assured that the algorithm halts in a finite number of steps. In case $T_1 \equiv T_2$, it is obvious that the algorithm conclude that “$T_1 \equiv T_2$”. So, we shall consider the other case (i.e. $T_1 \not\equiv T_2$). Suppose for the sake of contradiction that the algorithm has halted with a conclusion that “$T_1 \equiv T_2$”. Let $T(T_1 : T_2)$ be the comparison tree which has been developed as far as possible. Then it should follow that we have the following Claim $E_n$ for any nonnegative integer $n$.

**Claim $E_n$** For every node labeled $(p, \alpha) \equiv h(\bar{p}, \beta)$ which has been applied branching, skipping or stack reduction in $T(T_1 : T_2)$, if there exists a derivation $(p, \alpha) \Rightarrow (q, \gamma)$.
for some \( x \in L(p, \alpha) \subseteq \Sigma^* \), \( u \in \Delta^*, \ q \in F_1, \ \gamma \in \Gamma_1^* \), then the followings hold:

(i) There exists a derivation

\[
(p, \beta) \xrightarrow{u/v} (q, \gamma) \text{ with } u = hv
\]

for some \( v \in \Delta^* \), \( q \in F_2, \ \bar{\gamma} \in \Gamma_2^* \).

(ii) There exists a derivation path such that

\[
(p, \alpha) \equiv (\bar{p}, \beta) \xrightarrow{u_0|x_0/c_0} \xrightarrow{T_1|T_2} (q, \gamma_0) \equiv (q, \bar{\gamma}_0)
\]

for some \( x_0 \in (\Sigma \cup \{\lambda\})^* \), \( u_0, v_0 \in \Delta^* \), where \( |\sigma(x_0)| \leq |x| \), \( u_0 = hv_0 \), \( |\gamma_0| \leq |\gamma| \), and \( |\bar{\gamma}_0| \leq |\bar{\gamma}| \).

**Proof of Claim E_n:** This can be proved by induction on \( n \).

**Proof of Lemma 7 (continued):** Now if we apply this claim especially to the root labeled \((q_0, Z_{01}) \equiv (q_{02}, Z_{02})\), then we know that the condition of Proposition 1 holds, since we have encountered no checking failure. Thus, \( T_1 \equiv T_2 \) should hold, contradicting the assumption that \( T_1 \neq T_2 \). Therefore, some checking failure does occur at some stage, concluding that \( T_1 \neq T_2 \).

We have the following theorem.

**Theorem 1** For the given two real-time droct’s which accept by final state, our equivalence checking algorithm halts in a polynomial-time with respect to the cardinalities of states, input symbols, output symbols, and transition-output rules with the correct conclusion.

6. Conclusions

We have presented an algorithm for checking the equivalence for real-time droct’s which accept by final state. Then the worst-case time complexity of our algorithm is polynomial with respect to the description length of these droct’s. For future work, our present algorithm should be extended to non-real-time droct’s which accept by final state.

Acknowledgments

This work was supported in part by Grants-in-Aid for Scientific Research Nos.20500007 and 23500011 from the MEXT of Japan.

References


Mitsuo Wakatsuki received his Dr. Eng. degree from the University of Electro-Communications in 1993. He is now an Assistant Professor in the Graduate School of Informatics, Graduate School of Informatics and Engineering at the University of Electro-Communications. His research interests include theory of automata and formal languages, computational learning theory, and combinatorial optimization problems. He is a member of IEICE, IPSJ, and JSAI.
Etsuji Tomita received Dr. Eng. from Tokyo Tech. He was Professor at Univ. of Electro-Communications (UEC Tokyo). He is presently Professor Emeritus at UEC Tokyo and is with the Advanced Algorithms Research Laboratory, UEC Tokyo. He is also Research Advisor of JST ERATO Project, and is with Tokyo Tech. His research interests include theory of automata and formal languages, algorithmic learning theory, and combinatorial optimization. He served as PC Chair of ALT 2005, Conference Chair of ICGI 2006, and Director of IPSJ. He is an Editor of ISRN Discrete Mathematics and some other journals. He was given the Funai Information Technology Prize and was awarded Theoretical Computer Science Top Cited Article 2005-2010. He is a Fellow of IEICE and IPSJ.

Tetsuro Nishino graduated from department of Mathematics, Waseda University, and continued his research and obtained a D. Sc. in Mathematics in 1991. He was a researcher at Tokyo Research Laboratory, IBM Japan during 1984-1987. He was an Associate Professor at School of Information Science, Japan Advanced Institute of Science and Technology, Hokuriku during 1992-1994. In 1994, he joined at Department of Communications and Systems Engineering, The University of Electro-Communications as an Associate Professor and in 2006 he became a Professor. He received The Funai Information Technology Prize (2003) and IBM Faculty Award (2008).
Abstract

This paper describes Akara 2010, the first distributed shogi system in history to defeat a professional shogi player in a public game. The system employs a novel design to build a high-performance computer shogi player for standard tournament conditions. The design enhances the performance of the entire system by means of distributed computing. To utilize a large number of computers, a majority-voting method using four existing programs is combined with a distributed-search method.

Keywords: chess variant, shogi, distributed search, majority voting, Akara 2010

1. Introduction

Human-computer chess matches have provided an index of advances in modern artificial intelligence. One of the most prominent pieces of historical evidence was provided by Deep Blue, which won the six-game match against Garry Kasparov by one point in 1997 [1]. Since then, chess machines have started playing chess better than top-level human players. Nowadays, programs running on mobile phones seem to have reached the grandmaster level, and Pocket Fritz 4 won the Copa Mercosur tournament in 2009 [2].

In creating artificial intelligence, shogi (a Japanese chess variant) is probably a more difficult target than chess [3]. Because shogi has a dropping rule, i.e., a captured piece can be dropped back onto the board, the average number of legal moves is greater than in chess (the raw branching factor in shogi is around 80) [4]. This suggests that a shogi program needs to search for a much larger game tree than a chess program to reach the grandmaster level. In fact, human-computer shogi matches are still challenging for a computer player in 2013.

In chess and shogi, the performance of game-tree searches is considerably improved by pruning techniques, such as alpha-beta pruning [5], with the use of a large transposition table [6, 7]. Therefore, a distributed search should visit more nodes in the tree than a single-processor search does without losing the efficiency of such pruning techniques. Unlike SMP parallelization such as dynamic tree splitting (DTS) [8], the difficulty in distributed computing comes from the limitation in network-
connection speed between multiple processors: (i) 
synchronizing individual searches at each processor when 
one processor experiences a beta cutoff; and (ii) managing 
the large transposition table that has to be updated and 
probed frequently by the multiple processors.

In this paper, we discuss distributed computing in 
shogi to play grandmaster-level games. We propose a 
novel design to build a high-performance yet fault-
tolerant system for demonstrations in standard tournament 
conditions of shogi. Our system design utilizes multiple 
existing shogi programs. It also utilizes massive computer 
resources, i.e., many existing computers of various 
architectures at various places with internet connections.

Our system uses the majority-voting method proposed 
by Obata et al. [9]. The voters are four different existing 
programs, Gekisashi\footnote{Developed by Y. Tsuruoka, D. Yokoyama et al., commercial software from Mynavi corporation.}, GPS Shogi\footnote{T. Kaneko, T. Tetsuro et al., source codes are available online from http://gps.tanaka.ecc.u-tokyo.ac.jp/gpsshogi/}, Bonanza\footnote{Developed by K. Hoki, commercial software from Magnoria, Inc., source codes are available online at http://www.geocities.jp/bonanza_shogi/}, and YSS\footnote{Developed by H. Yamashita, commercial software (named Al Shogi) from e frontier, Inc.}. Their performance is enhanced by state-of-the-art techniques such as machine learning of evaluation 
functions, which has now become common practice in 
computer shogi due to the successful application in 
Bonanza [12]. Given a large number of available 
computers, the majority-voting method is combined with 
the distributed-game-tree-search method [13].

On the basis of this proposal, the shogi program Akara 
2010 was developed and defeated one of the top female 
players, Ichiyo Shimizu, on October 11, 2010. Here, the 
game used a standard time control, i.e., 180 minutes plus 
a 60-second countdown. This was the first time a shogi 
machine defeated a professional in a public game. This 
paper presents some new results of the majority-voting 
method.

2. Related Work

Distributed computing of game-tree searches has been 
extensively studied. One prominent strategy for using a 
massive computer resource is to build specialized 
hardware for the game-playing program to lift the 
limitation in network-connection speed. Deep Blue [1] 
and Hydra [14] are famous examples of massively 
parallelized chess machines.

The second strategy for distributed computing is to 
refine the divisions of the tasks on the bases of alpha-

beta pruning, where the entire game tree is decomposed 
into multiple sub-trees. One naive way would be to divide 
the tasks at the root position. Here, all children of the root 
node are partitioned among the multiple processors, and 
each processor takes charge of a subset of the children by 
using a local transposition table. Since this naive way 
does not change the subsets throughout an entire search, 
all separate tasks across multiple processors can be 
independently computed without frequent network 
communication. One important extension from such a 
a naive parallelization would be dividing the tree at not 
only the root node but also multiple interior nodes of 
different depths. Such an extension can be found in early 
work, e.g., the young brothers wait concept (YBWC) [15], 
asynchronous parallel game-tree search (APHID) [16], 
and transposition-table driven scheduling alpha-beta 
(TDSAB) [17].

The third strategy is the extended use of pondering 
methods [18, 19]. In an ordinary pondering method, the 
expected move of the opponent is considered as already 
played, and a computer player starts searching during the 
opponent’s thinking time to avoid the computer being idle 
while the opponent is thinking. The reported method is to 
start searching speculatively ahead on the basis of the 
expected move sequence during both the player’s and the 
opponent’s thinking time.

Althöfer and Snatzke have reported 3-Hirn in two-

player games [20], where two programs provide move 
opinions and a human selects a single move from these 
two. Therefore, the majority-voting method can be 
regarded as a reduction from 3-Hirn, where the human 
termination in two computer programs is removed. The 
majority voting has been applied to shogi [9] and 
chess [21]. Moreover, it has been reported that majority 
voting is even effective in the Monte-Carlo tree search of 
Go programs [22].

The majority-voting method utilizes the distributed 
environment for better performance. This was 
demonstrated in tournament conditions in 2009 [23]. The 
distributed shogi-tree search has also been demonstrated 
in tournament conditions in 2010 [13]. The notable 
advantage of these methods is simplicity, and most 
existing shogi programs can be made into worker 
programs with minimal modifications.

Recently, an artificial intelligence system has 
challenged human experts in games other than variants of 
two-player games. In 2011, Watson defeated two former 
winners of the quiz show Jeopardy! [24]. Watson is a 
question answering system that can carry out natural 
language processing and reasoning to answer questions on 
a quiz show [25].

3. Setup of Entire System

Figure 1 shows the outline of the entire system. The 
workers in the three sets (A, B, and C) individually 
compute their best move for a given position. Each 
worker is one of the four programs running on a single
computer. Because the computer has multiple cores, each worker carries out a shared-memory parallel search for the given game tree. Sets A, B, and C connect to servers 1, 2, and 4, respectively.

Server 1 continuously makes and runs shogi games between two workers. For the server program, we used the shogi-server program package that is available online. This program had been maintained and run on the internet shogi server, Floodgate, for the past two years [26]. For the purpose of programs and the hardware test, game server 1 connected worker set A and majority-voting server 2 for 24 hours. The game records and log messages were continuously uploaded to the Web, and the authors were able to check the activity of the system any time.

The system is able to play a game with human players using a shogi-board Graphical User Interface (GUI) client connected to game server 1. We used the SFICP GUI program with some modifications to suspension and resumption functions [27].

Majority-voting server 2 and the following play the central role in the entire system. The main role of server 2 is to gather move opinions on the current game position from workers and decide the move to play on the basis of majority voting, where the weights of votes are adjustable parameters. Majority-voting server 2 allows shutdowns and connections from workers anytime in the middle of a game. However, at least one worker has to be connected to guarantee the continuation of a game.

Unit 3 is a wrapper program that absorbs the difference in communication protocols. Moreover, unit 3 manages to restart each worker and establish a new connection again to the majority-voting server 2 when the old internet connection experiences an interruption.

The dark-shaded box indicates the units indispensable to continue a shogi game. These were necessary to inform the GUI client of the move decision of the system. Moreover, the pondering function while the opponent is thinking was provided by majority-voting server 2, and the suspend-and-resume function was provided by game server 1, majority-voting server 2, and the GUI client. These indispensable units were composed using reliable hardware and software that had passed more than 1000 game tests (see Table I).

On the other hand, the light-shaded box indicates optional units that are designed to boost the performance of the entire system. The purpose of the separation from the dark-shaded ones is to utilize whatever hardware and system software are available at that moment. The requirements for the workers inside the light-shaded box are intended to result in a freezing or termination state without sending low-quality moves when they experience abnormal behavior. Distributed-search server 4 divides

---

Table 1. List of computers used in Akara2010.

<table>
<thead>
<tr>
<th>Set</th>
<th>Processor</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hongo Campus, the University of Tokyo</strong></td>
<td>Intel Xeon W3680 3.33GHz, 6 cores</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>Intel Xeon W3680 3.33GHz, 6 cores</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel Xeon E5410 2.33GHz, 8 cores</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>Intel Xeon 2.40GHz, 2 cores</td>
<td>60</td>
</tr>
<tr>
<td><strong>isths Cluster, Graduate School of Information Science and Technology, The University of Tokyo</strong></td>
<td>Intel Xeon 2.80GHz, 2 cores</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Intel Xeon 2.40GHz, 2 cores</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel Xeon 2.80GHz, 8 cores</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>AMD Opteron 2380 2.4GHz, 8 cores</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Intel Xeon E5410 2.33GHz, 8 cores</td>
<td>9</td>
</tr>
<tr>
<td><strong>InTrigger Cluster</strong></td>
<td>Intel Xeon X5560 2.80GHz, 8 cores</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Intel Xeon X5560 2.80GHz, 8 cores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Intel Xeon X5470 3.33GHz, 8 cores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Intel Xeon X5365 3.00GHz, 8 cores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>AMD Opteron 2376 2.3GHz, 8 cores</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>AMD Opteron 280 2.4GHz, 4 cores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Intel Core2 Quad Q6700 2.66GHz, 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel X5680, 12 cores</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Intel W3680, 6 cores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Intel Core i7 980X, 6 cores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Intel W3440, 4 cores</td>
<td>1</td>
</tr>
<tr>
<td><strong>Komaba Campus, the University of Tokyo</strong></td>
<td>Intel X5365 3.00GHz, 8 cores</td>
<td></td>
</tr>
<tr>
<td><strong>The University of Electro-Communications</strong></td>
<td>Intel X5680, 12 cores</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Intel X5680, 12 cores</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Intel W3680, 6 cores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Intel Core i7 980X, 6 cores</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Intel W3440, 4 cores</td>
<td>1</td>
</tr>
<tr>
<td><strong>Kanagawa</strong></td>
<td>Intel Core i7 980X 3.3GHz, 6 cores</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>Intel Core i7 980X 3.3GHz, 6 cores</td>
<td>1</td>
</tr>
</tbody>
</table>

Used by ‘Gekisashi,’ ‘GPS Shogi,’ ‘Bonanza,’ and ‘YSS’
the game tree into multiple sub-trees and assigns one of these sub-trees to each worker in set C. These workers ran on the cluster istbs, InTrigger\(^2\), and the other computers (see Table 1).

4. Majority-Voting Method

The primary goal of the majority-voting method is to strengthen each base program by composing an ensemble of the four programs. Because these programs are different, we are able to have four different principal variations. Some variations may branch off at the root node, while the others branch off at some interior nodes. In the example shown in the left panel of Fig. 2, the number in the nodes indicates votes, and move \(m_2\) is the majority. The selection of \(m_2\) is unfounded in the strict sense. However, one can speculate that selecting \(m_2\) supported by three programs is less risky than selecting the other one supported by a single program.

The right panel of Fig. 2 shows an example of a game tree, where the majority voting is effective. There are four playable move sequences. Because of the huge size of the game tree and the limitation of the time control, we have no hope to verify these sequences by in-depth consideration. Under these conditions, selecting move \(m_2\) supported by three playable sequences is reasonable. However, a single base program has a probability of 0.25 to select move \(m_1\). Majority voting is able to reduce this probability.

There are two more favorable properties of the majority-voting method in addition to the enhancement of strength. The first is fault tolerance. The standard tournament conditions of shogi games are real-time games. Therefore, stable computations and input/output functions are necessary to win a game. However, to the best of our knowledge, existing distributed-search methods do not take into account hardware or network trouble. That is, the more computers, the more accidents. In contrast, the majority-voting method is fault tolerant. The majority-voting server allows the shutdown and connection of a worker at any point in time. Therefore, whatever the number of computers we use in the light-shaded box in Fig. 1, the probability of having accidents is mostly determined only by those in the dark-shaded box.

The second favorable property of the majority-voting method is that it is difficult for a human player to predict the result of votes in advance. This property is expected to help disturb the opponent modeling of a human player. That is, even if she/he knows a tactical deficiency of a specific program in advance, she/he cannot use this knowledge to win the game when the other programs do not have the same deficiency.

We have designed the majority-voting server to be as simple as possible to avoid unexpected accidents. The server program is written by using about 1000 lines of Perl script. The server program had the following three phases:

**Phase 1:** The system considers a move to play. After waiting for the thinking time \(t\), the majority-voting server sends the majority move to the game server and all workers. The system proceeds to phase 2.

**Phase 2:** The system predicts the opponent’s move. As soon as it receives the opponent’s move from the game server, the systems forward it to all workers and returns to phase 1. After waiting for a time shorter than \(t\), the opponent is assumed to play the majority move. The system sends the predicted move to all workers and proceeds to phase 3.

**Phase 3:** The system considers a response move to the predicted move. It waits for the opponent’s move from the game server. If the opponent’s move and the predicted move are the same, then it returns to phase 1. Otherwise, the system sends the opponent’s move as an alternating move to all workers and then returns to phase 1.

If the system is the player who moves first, the phase starts from 1; otherwise the phase starts from 2. To resume a suspended game, the majority-voting server sends a move sequence to the suspended position to all workers. Then the phase starts from 1 if the system is the player who moves first from the suspended position. Otherwise, it starts from 2.

\(^{2}\) New IT Infrastructure for the Information-explosion Era, MEXT Grant-in-Aid for Scientific Research on Priority Areas.
The thinking time $t$ is determined in terms of the extent of voting agreements. When all workers vote for the same move, the time $t$ is reduced by half from a standard thinking time. On the other hand, when all workers vote for different moves, the time $t$ is increased up to twice from the standard thinking time. The standard thinking time is set to consume 180 minutes in 60 moves.

The workers are required to carry out the following processing:

- They keep on considering the best move for the current position. As soon as the best move at the moment changes, they send the new best move to the majority-voting server.
- When a move arrives from the majority-voting server, the workers update the current position in accordance with the move. Also, when an alternation move arrives from the majority-voting server, they play back the last move of the position and then update the current position in accordance with the alternation move.

The workers also take care of the following auxiliary functions:

- They send the total nodes searched to the majority-voting server at the right interval to keep network connections alive.
- They send the opening signal when a worker recognizes the current position as an opening. This signal is used to reduce the thinking time in early opening.
- They send the thinking-stop signal. This signal is sent when a worker finds the current position in the opening database or the position is solved. The majority-voting server is able to play a move immediately when the majority is achieved from workers that send this signal.

Table 2 shows the performance of the majority-voting method. Each winning percentage is measured by 1000 games whose opening moves are randomly chosen in accordance with their opening databases. The opponent in all games is Gekisashi, and the winning percentage, 50%, of Gekisashi is a theoretical value. For each move, all players use four seconds of the same ordinary desktop computer with a sufficient memory allocation for the transposition tables. To resolve tied votes, the weight of the vote of Gekisashi is set to a value slightly higher than one. The majority voting provides the highest percentage, 73%. Although these results are produced using the short time control, they indicate that the majority voting using these four programs is effective.

### Table 2. Performance of majority voting.

<table>
<thead>
<tr>
<th>Player</th>
<th>Win</th>
</tr>
</thead>
<tbody>
<tr>
<td>Majority voting of these four players</td>
<td>73%</td>
</tr>
<tr>
<td>Gekisashi</td>
<td>50%</td>
</tr>
<tr>
<td>GPS Shogi</td>
<td>36%</td>
</tr>
<tr>
<td>Bonanza</td>
<td>62%</td>
</tr>
<tr>
<td>YSS</td>
<td>37%</td>
</tr>
</tbody>
</table>

![Fig. 3. Extract from log file of majority-voting server of game on October 11, 2010 and some explanations [28].](image)

5. Distributed Search for Game Trees

Distributed search for a game tree remains challenging, and a definitive method is still unknown. Because we needed to use a distributed computing environment to build the entire system in a practical fashion, we adopt a simple master/slave model of communication in this work. In this section, we use the terminologies master and slaves instead of server and worker. That is, the distributed-search server is the master program, and the workers in set C are the slave programs.

The master program expands a shallow and narrow tree $T$ rooted at the current node and assigns each slave to search for a large tree rooted at a node contained in $T$. The workers also take care of the following auxiliary functions:

- They keep on considering the best move for the current position. As soon as the best move at the moment changes, they send the new best move to the majority-voting server.
- When a move arrives from the majority-voting server, the workers update the current position in accordance with the move. Also, when an alternation move arrives from the majority-voting server, they play back the last move of the position and then update the current position in accordance with the alternation move.

- They send the total nodes searched to the majority-voting server at the right interval to keep network connections alive.
- They send the opening signal when a worker recognizes the current position as an opening. This signal is used to reduce the thinking time in early opening.
- They send the thinking-stop signal. This signal is sent when a worker finds the current position in the opening database or the position is solved. The majority-voting server is able to play a move immediately when the majority is achieved from workers that send this signal.

Table 2 shows the performance of the majority-voting method. Each winning percentage is measured by 1000 games whose opening moves are randomly chosen in accordance with their opening databases. The opponent in all games is Gekisashi, and the winning percentage, 50%, of Gekisashi is a theoretical value. For each move, all players use four seconds of the same ordinary desktop computer with a sufficient memory allocation for the transposition tables. To resolve tied votes, the weight of the vote of Gekisashi is set to a value slightly higher than one. The majority voting provides the highest percentage, 73%. Although these results are produced using the short time control, they indicate that the majority voting using these four programs is effective.

5. Distributed Search for Game Trees

Distributed search for a game tree remains challenging, and a definitive method is still unknown. Because we needed to use a distributed computing environment to build the entire system in a practical fashion, we adopt a simple master/slave model of communication in this work. In this section, we use the terminologies master and slaves instead of server and worker. That is, the distributed-search server is the master program, and the workers in set C are the slave programs.

The master program expands a shallow and narrow tree $T$ rooted at the current node and assigns each slave to search for a large tree rooted at a node contained in $T$. The workers also take care of the following auxiliary functions:

- They keep on considering the best move for the current position. As soon as the best move at the moment changes, they send the new best move to the majority-voting server.
- When a move arrives from the majority-voting server, the workers update the current position in accordance with the move. Also, when an alternation move arrives from the majority-voting server, they play back the last move of the position and then update the current position in accordance with the alternation move.

- They send the total nodes searched to the majority-voting server at the right interval to keep network connections alive.
- They send the opening signal when a worker recognizes the current position as an opening. This signal is used to reduce the thinking time in early opening.
- They send the thinking-stop signal. This signal is sent when a worker finds the current position in the opening database or the position is solved. The majority-voting server is able to play a move immediately when the majority is achieved from workers that send this signal.

Table 2 shows the performance of the majority-voting method. Each winning percentage is measured by 1000 games whose opening moves are randomly chosen in accordance with their opening databases. The opponent in all games is Gekisashi, and the winning percentage, 50%, of Gekisashi is a theoretical value. For each move, all players use four seconds of the same ordinary desktop computer with a sufficient memory allocation for the transposition tables. To resolve tied votes, the weight of the vote of Gekisashi is set to a value slightly higher than one. The majority voting provides the highest percentage, 73%. Although these results are produced using the short time control, they indicate that the majority voting using these four programs is effective.

5. Distributed Search for Game Trees

Distributed search for a game tree remains challenging, and a definitive method is still unknown. Because we needed to use a distributed computing environment to build the entire system in a practical fashion, we adopt a simple master/slave model of communication in this work. In this section, we use the terminologies master and slaves instead of server and worker. That is, the distributed-search server is the master program, and the workers in set C are the slave programs.

The master program expands a shallow and narrow tree $T$ rooted at the current node and assigns each slave to search for a large tree rooted at a node contained in $T$. The workers also take care of the following auxiliary functions:

- They keep on considering the best move for the current position. As soon as the best move at the moment changes, they send the new best move to the majority-voting server.
- When a move arrives from the majority-voting server, the workers update the current position in accordance with the move. Also, when an alternation move arrives from the majority-voting server, they play back the last move of the position and then update the current position in accordance with the alternation move.

- They send the total nodes searched to the majority-voting server at the right interval to keep network connections alive.
- They send the opening signal when a worker recognizes the current position as an opening. This signal is used to reduce the thinking time in early opening.
- They send the thinking-stop signal. This signal is sent when a worker finds the current position in the opening database or the position is solved. The majority-voting server is able to play a move immediately when the majority is achieved from workers that send this signal.

Table 2 shows the performance of the majority-voting method. Each winning percentage is measured by 1000 games whose opening moves are randomly chosen in accordance with their opening databases. The opponent in all games is Gekisashi, and the winning percentage, 50%, of Gekisashi is a theoretical value. For each move, all players use four seconds of the same ordinary desktop computer with a sufficient memory allocation for the transposition tables. To resolve tied votes, the weight of the vote of Gekisashi is set to a value slightly higher than one. The majority voting provides the highest percentage, 73%. Although these results are produced using the short time control, they indicate that the majority voting using these four programs is effective.
The small tree $T$ is composed on the basis of the move ordering of the slaves.

- The slave assigned a leaf node of $T$ takes charge of all children of the leaf node. On the other hand, the slave assigned an internal node of $T$ takes charge of all children nodes that are not contained in $T$.
- Each slave sends its best child and the corresponding evaluation value every time these are updated through their tree searches, and the master program updates and sends the best move to the majority-voting server.

This distributed-search method is effective because each slave is able to start a tree search from a deeper node than the current node if the master program expands a proper small tree $T$. The server program is written in about 2,500 lines of Perl script.

This distributed-search method has two advantages. The first is the less-frequent communication between master and slave programs. The messages that each slave has to send to the master are less than 1000 bytes, and the communication frequency is about once per second. The second is the applicability to an existing program. The requirements for a slave are to compute a good move ordering, the best child, and an evaluation value. Note that any practical shogi program has these functions.

The results of the performance test using GPS Shogi and ordinary desktop computers with the time control of 15 seconds per move confirm that our implementation is effective. The distributed search using eight slaves has a winning percentage of more than 70% against a single slave program.

6. Demonstration of Akara 2010

Akara 2010 defeated a professional player in a public game on October 11, 2010, the log file of which is available online (see Fig. 3). In this game, because most computers had not been available for testing, the weights of votes from the light-shaded box in Fig. 1 were set sufficiently smaller than one to stay on the safe side. Although the distributed-search components did not strongly influence the move decision, the components were effectively used to determine thinking time $t$ as described in Sec. 4.

Figure 4 shows the changes in evaluation values of the four different programs. The vertical axis indicates the evaluation value, a large value of which means that the current game position is advantageous for Akara 2010. We see that most evaluation values are positive. This means all programs regarded the game to be favorable to Akara 2010 for most positions. Although the four
different programs agreed on that point, they had different opinions about detailed superiority. For example, Gekisashi had greater confidence than the others around the 24th move, but it lost this confidence around the 40th move. Moreover, GPS Shogi and Bonanza evaluated positions around the 46th move highly than the other programs, but they lost their confidence around the 60th move. These results indicate that the four programs evaluated the game positions from different points of view.

Figure 5 shows the results of voting. There were 36 moves in total. Here, ten moves were accepted unanimously, and six moves were selected by fewer than the half votes. The results indicate that there was sufficient diversity of move opinions from the four programs.

In Fig. 5, we also see that the number of votes was smaller than the others at the 24th, 42nd, 44th, 46th, 52nd, 84th, and 86th moves. At these moves, the internet connection to the cluster units in set C experienced interruptions and we restarted some of the distributed-search servers four times. This indicates that our shogi system was fault tolerant against occasional failure of units in the dark-shaded box in Fig. 1.

Figure 6 shows one of the positions where these programs output multiple opinions. For the first four minutes of the standard thinking time, GPS Shogi, Bonanza, and YSS supported “66. B*5g” and Gekisashi supported “66. B*5g”. Because not all programs agreed, the thinking time was extended to eight minutes. While Akara 2010 was thinking more deeply, Bonanza changed its opinion to “66. B*5g”, and the move won the majority.

Figure 7 shows several plausible variations from the position in Fig. 6. Shogi moves are shown by using both Japanese and English notations. The evaluation values at each plausible-variation leaf were computed by using GPS Shogi (the upper row) and Bonanza (the bottom row). Each value at these leaves was computed by using two minutes on recent desktop computers. We see that the evaluation values varied from 50 to −400 (exchanging a pawn corresponds to about 200). Moreover, the two programs supported different variations, especially positions followed by “67. N-7g”. These observations tell us that the position at the 66th move was still not easy for Akara 2010. However, the game had been decided at the 67th move because the human player who had used up all 180 minutes of her thinking time made a blunder (“67. G-7g”) in the 60-second countdown.

7. Conclusion

We have presented a system design to build a high-performance computer shogi player. In the system design, we have considered enhancing the performance by using a distributed computing environment. At the same time, we have also considered stabilizing the entire system to play a game under standard tournament conditions. To utilize a large number of computers, a majority-voting method has been combined with the distributed-search method. On the basis of this system design, we have developed a shogi-machine, Akara 2010. The move decision is made by the majority voting of four programs: Gekisashi, GPS Shogi, Bonanza, and YSS. Because the entire system of Akara 2010 had been operational for only a few days, a statistically reliable test of the performance was not available. We have carried out partial tests of the entire system, where the majority-voting component increased the winning probability from 62% to 73%, and the distributed-search component increased it from 50% to 70% or more. Moreover, Akara 2010 defeated a
professional player in a public game. These results indicate the effectiveness of our system design.

Acknowledgments

We are grateful to Dr. Hitoshi Matsubara for the fruitful discussion on the human-computer shogi match project, Dr. Tsuyoshi Hashimoto for the fruitful discussion on the shogi opening database, and Dr. Hideyuki Nakashima for his support of this work. This work was supported by the Information Processing Society of Japan (IPSJ) and by JSPS KAKENHI Grant Number 25730212.

References


28. Utl: http://www.ipsj.or.jp/50anv/shogi/20101012-2.html (last access: 2013)
Information for Authors

Scope

The aim of the International Journal of Computer & Information Science is to present information on well defined theoretical results and empirical studies, to describe current and prospective practices and techniques, to provide critical analysis of the use of CASE tools and to present broad issues in the fields of Computer Science and Engineering, Software Engineering and Information Science and Engineering. A primary aim of the journal will be to make the research accessible and useful to practitioners through carefully written methodology section and discussions of research results in ways that relates to practitioners.

Submission Policy

1. The IJCIS does not accept manuscripts or works that have been published in a journal, but accepts substantially revised and expanded versions of papers that were previously presented at conferences. In this case, the author should inform the Editor-in-Chief of the previous publication when the paper is submitted.

2. Submissions, referencing and all the correspondence will be conducted by e-mail.

3. Manuscripts should be in English and are subject to peer review before selection.

Procedure for Submission for Review

1. Send your complete text of manuscript to the Editor-in-Chief (e-mail: lee@cps.cmich.edu) via e-mail as an attachment document either in Word, PDF or PostScript format.

2. It may be useful for non-English speakers to have a native English speaker read through the article before submission.

3. Your e-mail message (to the Editor-in-Chief) should include the title of your paper, author(s) information (department/section, university/company, phone/fax numbers, e-mail address), the corresponding author’s complete address (mailing address), and area(s) of the paper.

Style of Manuscript to be Submitted for Review

1. The text should be double-spaced, two columned, single-sided in Times 12-point type on 8.5-inch x 11-inch paper with margins of 1 inch on all four sides. It should not exceed 30 double-spaced pages. All pages must be numbered sequentially.

2. Provide a 200-250 word abstract, 5-7 keywords. Your abstract should briefly summarize the essential contents.

3. All figures/tables must be captioned, numbered and referenced in the text.

4. An acknowledgement section may be presented after the conclusion section.

5. References should appear at the end of the paper with items referred to by numbers in square brackets.

Journal Papers: author’s first initials, followed by last name, title in quotation marks, journal (in Italics), volume and issue number, inclusive pages, month and year. For e.g.:


Papers of Proceedings: author’s first initials, followed by last name, title in quotation marks, proceedings / symposium / workshop name (in Italics), location, month, and year, and inclusive page numbers. For e.g.:


Books: author’s first initials, followed by last name, title (in Italics), edition, location: publisher, year. For e.g.:


Accepted Manuscript Submission

All accepted papers are edited electronically. You are required to supply an electronic version of the accepted paper. The detailed guidelines for preparing your final manuscript will be sent upon article acceptance. It is also available on the ACIS website at http://acisinternational.org.

Page Charges

After a manuscript has been accepted for publication, the author will be invoiced a page charge of $400.00 USD for up to 10 pages if he/she is not a member of ACIS and thereafter an additional $30 per printed page to cover part of publication after the receipt of your final manuscript (camera ready).