Task-based Example Miner for Intelligent Tutoring Systems

Ritu Chaturvedi
University of Windsor

Task-based Example Miner for Intelligent Tutoring Systems

by

Ritu Chaturvedi

A Dissertation
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada

Task-based Example Miner for Intelligent Tutoring Systems
by
Ritu Chaturvedi

APPROVED BY

May 24th, 2016

Declaration of Previous Publication

This thesis includes five original papers that have been previously published or submitted to in blind reviewed journal and conference proceedings, as follows:

Thesis Chapter	Publication title/full citation	Publication status
Part of chapter 2	Chaturvedi, R. & Ezeife, C. I. (2012). Data mining techniques for design of its student models. In Fifth ACM International Conference on Educational Data Mining-EDM, June 21 - 23, Chania, Greece. (pp. 232-233).	published
Part of chapter 1	Chaturvedi, R. & Ezeife, C. I. (2013). Mining the impact of course assignments on student performance. In Sixth ACM Inter- national Conference on Educational Data Mining-EDM, July 6 to 9, Tennessee,USA. (pp. 308-309).	published
Part of chapters 4 and 5	Chaturvedi, R. & Ezeife, C. I. (2014). Mining relevant examples for learning in its student models. In 2014 IEEE International Conference on Computer and Information Technology (pp. 743-750).	published
Part of chapter 1	Chaturvedi, R. & Ezeife, C. (2015a). Mining Boolean data using martrix algebra. In IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), Oct. 2015 (pp. 901-906).	published
Major part of chapters 4 and 5	Chaturvedi, R. & Ezeife, C. I. (2015b). Task-based example mining for learning in an ITS. Submitted to JEDM (Journal Of Educational Data Mining).	submitted

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor. I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis. I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

Intelligent tutoring systems (ITS) aim to provide customized resources or feedback on a subject (commonly known as domain in ITS) to students in real-time, emulating the behavior of an actual teacher in a classroom. This thesis designs an ITS based on an instructional strategy called example-based learning (EBL), that focuses primarily on students devoting their time and cognitive capacity to studying worked-out examples so that they can enhance their learning and apply it to similar graded problems or tasks. A task is a graded problem or question that an ITS assigns to students (e.g. task T1 in C programming domain defined as "Write an assignment instruction in C that adds 2 integers"). A worked-out example refers to a complete solution of a problem or question in the domain. Existing ITS systems such as NavEx and PADS, that use EBL to teach their domain suffer from several limitations such as (1) methods used to extract knowledge from given tasks and worked-out examples require highly trained experts and are not easily applicable or extendable to other problem domains (e.g. Math), either due to use of manual knowledge extraction methods (such as Item Objective Consistency (IOC)) or highly complex automated methods (such as syntax tree generation) (2) recommended worked-out examples are not customized for assigned tasks and therefore are ineffective in improving student success rate.

This thesis proposes a new modular model for an EBL-based ITS called Example Recommendation System (ERS). ERS extracts knowledge in terms of basic learning units (LU) (e.g. scanf is a LU in the domain of C programming) from all task solutions and worked-out examples in its domain by using regular expression analysis and represents this knowledge in vector space. The prime contribution of knowledge extraction method of ERS is its extendibility to new domains without requiring highly trained experts. Experiments on two different domains show that LUs are extracted with 81% correctness for domain 1 (Programming in C) and 95% for domain 2 (Programming in Miranda). Knowledge extraction also serves as a crucial data pre-processing step for ERS, which then uses the extracted knowledge to mine its repository of worked-out examples using data mining methods such as k-nearest neighbors, in order to generate customized list of examples for each task in its domain. The accuracy of ERS's customization model is 93%, while its f_score is 88%. An evaluation of ERS demonstrates that the key elements (simpler and efficient automated knowledge extraction, extendibility to other domains, task-based customization, and clear integration of all components) have been accomplished and the overall goal of optimizing learning has been achieved. Experiments show that students score an average of 89% in tasks for which ERS recommends worked-out examples, compared to an average of 73% for tasks that students attempt without using any such examples.

Keywords: Data Mining, Intelligent tutoring system, student model, domain model, example-based learning, task, worked-out examples, learning units, asymmetric Boolean data, vector space model, similarity functions, feature extraction, classification, prediction, clustering

Dedication

Dedicated to my father and my father-in-law.

Acknowledgments

I would like to gratefully acknowledge the guidance, support and encouragement of my doctoral advisor, Dr. Christie I. Ezeife, and the members of my committee, Dr. D. Martinovic, Dr. A. Ngom and Dr. J. Lu, during my time at the University. My heartfelt and special thanks to Dr. Ezeife for her continued mentorship, insightful discussions and her belief in me. I also thank her for editing my work relentlessly, leading me towards a smarter path and also for the financial support she offered through grants such as Natural Science and Engineering Research Council (NSERC), Canada, FedDev/OBI and SSHRC. I feel very proud to be her student and thank her for providing an excellent example as a successful woman professor and researcher.

My gratitude extends to all staff members of the School of Computer Science and to my colleagues, for their help and support. I take this opportunity to express special gratitude to Gloria, who has been a true friend and a moral support in this journey of mine.

I could not imagine this success without the continual help, support and inspiration from my friends, especially Romi (Saraswat) and Vidya (Balachandar). Thank you for being there for me, especially at odd and cranky times.

I take this opportunity to express special gratitude and a heartfelt thanks to my big sister Madhu and her family (Aditya and Aayushi) and my little brother Devtosh and his family (Mona, Atharva and Yajur) for their extended support and encouragement from miles apart.

Finally, I am at loss of words to thank the 3 most important men in my life – my father (Papa), my son (Sridhar) and my husband (Pramod). Thank you Papa, for believing in me and for your countless blessings! Pramod and Sridhar - I thank you both for

supporting me and guiding me when the light at the end of this thesis seemed far. I dedicate this milestone to your love, support, patience and encouragement.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

This chapter is an introduction to Intelligent Tutoring Systems (ITS), its components (especially domain and student models) and data mining techniques used in such systems. It highlights the motivation behind designing an ITS driven by worked-out examples and describes the main thesis contributions.

1.1 Intelligent Tutoring Systems

There are several diverse learning environments to teach a course in today's technological world such as traditional in-class, distance learning, web-based online systems and blended environments that combine classroom teaching and web-based technology. According to Moore's definition (Moore et al., 2011), distance learning is a form of instruction in which the instructor and learner need not be at the same place at any time for the instructions to be delivered. Online learning is a newer version of distance learning which uses technology (such as the web) and shows some transformation of an individual's experience into the individual's knowledge using different levels of interactivity. For example, if student $s1$ has browsed a resource (such as a worked-out example on adding 2 fractions) n number of times ($s1$'s experience), then $s1$ is assumed to have mastered the resource ($s1$'s knowledge). Examples of commonly used online learning environments are Learning Management Systems (e.g. Blackboard (Windsor, 2014a)) used by University of Windsor) and ITS (e.g. Wayang Outpost (Arroyo et al., 2003) that helps high school students prepare for standardized math tests). For an online learning environment to be categorized as an ITS, it must satisfy the following requirements (Lukasenko, 2012):

1. learning environment must adapt to each student
2. it must use artificial intelligence and machine learning techniques to guide and coach the student and
3. it must consist of four models:
    (a) domain model that stores expert domain knowledge such as all correct answers or solutions
    (b) tutor model that emulates the role of a teacher in providing guidance and assistance to students as and when necessary
    (c) student model that stores a user model for each student's information such as his/her performance in the domain
    (d) user interface module that acts as the front-end for students using the ITS

According to Brusilovsky et al. (2003), an ITS typically has 2 main functions :

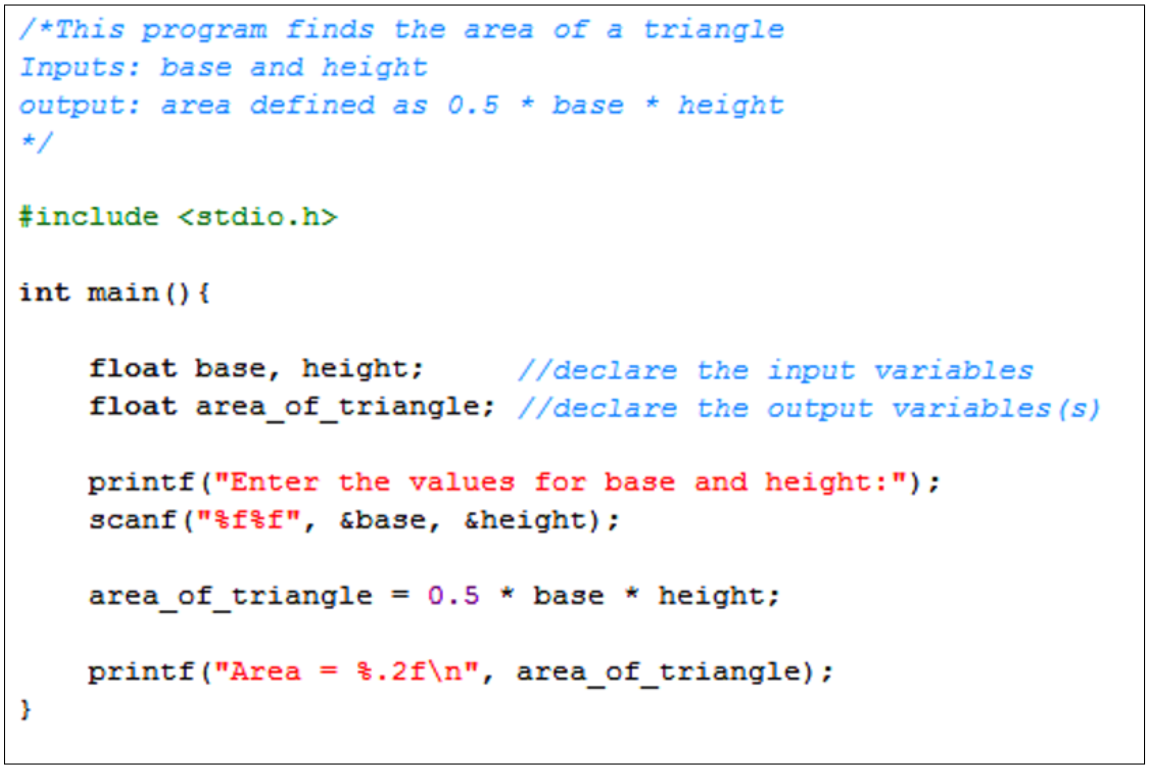
1. adaptive presentation of learning materials : ITS present learning materials based on student's cognitive styles (e.g. inductive/deductive), their form of perception (e.g. for a learner who prefers visual perception, the learning materials are presented as figures and graphs, as opposed to being presented as text), the level of learning difficulty (e.g. a learner with low difficulty level will be presented with easy questions) and their performance in domain knowledge (e.g. learners who achieve more than 80% in the domain concepts covered so far will be assigned task T1, whereas others will be assigned task T2).

2. intelligence based on objectives such as curriculum sequencing (e.g. each learner gets a personalized lesson plan based on his/her performance), problem-solving support (e.g. providing help to the learner while completing a task) and intelligent solution analysis (e.g. identifying difficulty level of a task for a learner, calculating the learner's knowledge level on a domain concept).

For an ITS to achieve this functionality, it requires to capture student data that defines both their learning behavior (e.g. interaction with the ITS resources) and their knowledge on the subject taught by the ITS (e.g. marks in a test or task). It also requires to store and manage its domain resources efficiently.

1.2 Domain and Student Model

A domain model defines the expertise required to teach the domain (a subject is typically called as a domain in ITS systems). In traditional classroom teaching, such expertise comes from a combination of the teacher and the text book put together. ITS domain model requires to store all basic concepts that experts (such as teachers) believe are important for students to learn, and the resources required to learn these concepts. These resources include lessons (in the form of lecture slides or videos) and their objectives, worked-out examples for each lesson and gradable tasks / tests that students are assigned so that their performance in the domain can be measured objectively. Domain model also stores solutions to all such tasks and tests. A worked-out example (WE) in this thesis (also referred to as example) is defined as a complete or partial worked-out solution for a question or instruction (similar to examples in textbooks). Figure 1.1 shows a sample worked-out example find_Area, which is essentially the solution to the following instruction: "Write a program that computes and prints the area of a triangle, given its base and height". A task is defined as a question or problem on any topic or concept in the domain. For example, task T1 in the domain of Programming in C is defined as "T1: Write a C program to find the area of a rectangle". A task solution is the solution of the question asked in a task. Students are typically assigned tasks by the ITS and are graded on them. This thesis uses the same structure to define a task solution and a worked-out example. ITS domain experts are also required to define the level of detail with which a resource in the ITS (such as a worked-out example) is represented. Experts typically define learning units (LU) (also referred to as a topics or concepts) as the finest level of detail for their domain's resources. For example, "scanf", which is a command for entering values into variables from the keyboard, is an LU in the domain of



```
/*This program finds the area of a triangle
Inputs: base and height
output: area defined as 0.5 * base * height
*/

#include <stdio.h>

int main(){

    float base, height;      //declare the input variables
    float area_of_triangle; //declare the output variables(s)

    printf("Enter the values for base and height:");
    scanf("%f%f", &base, &height);

    area_of_triangle = 0.5 * base * height;

    printf("Area = %.2f\n", area_of_triangle);
}
```

Figure 1.1: Worked-out example find_Area as solution to the instruction 'Write a program that computes and prints the area of a triangle, given its base and height'

C programming. Similarly, "fraction" is an LU in the domain of Math. The development of domain model is a very tedious job, and requires the time and effort of several domain experts. This thesis proposes automated methods to design and build domain models that reduce the effort required of experts.

A student model (SM) is an approximate, partial, mainly qualitative representation of the student's knowledge about a specific domain (Lukasenko, 2012). The objectives of an ITS dictate what comprises the student model and how it should be represented. For example, an ITS which supports teaching strategies that are adaptive to a student's learning style will require each student's SM to store his/her learning style. A SM can be represented using various structures such as files, relational databases (RDB), ontology or more function-specific network structures such as Bayesian networks (Lukasenko, 2012; Wang & Beck, 2013). Student modeling is a process of storing student data and making inferences about student's characteristics, their learning behavior and abilities (Chrysafiadi & Virvou, 2013). Student data can be categorized as static or dynamic. Examples of static characteristics are name, gender, preferences (e.g. student prefers to use examples before attempting a quiz) and learning styles (e.g. student prefers a learning style of collaboration or groups). Examples of dynamic student characteristics are cognitive abilities (e.g. student's understanding of a concept in the domain) and performance (e.g. marks scored in tests). Design of a student model for an ITS is based on 2 objectives : diagnostic (e.g. to predict student's needs and adapt the learning materials accordingly) and strategic (e.g. sequencing the course material based on students' needs and performance). The most common types of student models based on information stored in it and objectives for which the student model is created are stereotype, overlay and perturbation (SOMYUREK, 2009; Millan et al., 2010), as listed below.

1. Stereotype model creates groups of students based on their characteristics. For example, when a student $s1$ logs in to the ITS for the first time, he/she is assigned to one of the stereotype groups created by the designers of student model based on $s1$'s learning style of 'group work'.

2. Overlay model defines a student $s1$'s knowledge of the domain, typically represented by marks (e.g. 56 out of 100) or by grades (e.g. Pass / Fail). For example, in a scenario of an ITS that tutors adding fractions, a student $s1$ is asked to attempt task TM1 defined as "Add 2 fractions $\frac{1}{2} + \frac{1}{2}$ and show the result". Student s1's answer to TM1 is $\frac{2}{4} = \frac{1}{2}$. The ITS then grades $s1$ (using its domain model) and assigns $s1$ a grade of 'Fail' for task T1 and stores it in s1's student model.

3. Perturbation model not only stores a student $s1$'s knowledge but also the set of mistakes $s1$ makes. For example, perturbation model for task TM1 (shown in (2)) stores s1's mistake, in addition to the grade (e.g. Grade = 'Fail'; Mistake MM1 = student $s1$ adds the denominators of the 2 fractions even when they have a common denominator).

Student model used in this research for the domain of C Programming uses both a static component (e.g. student name and id) and a dynamic component (e.g. student's marks in a task). It is an overlay model and is designed to customize learning resources (such as worked-out examples) to student's needs (e.g. current task the student is assigned) and to predict student success in given tasks.

1.3 Data Mining techniques

Data Mining is the process of discovering interesting and useful patterns and relationships in large volumes of data. Data mining activities can be categorized as descriptive and predictive. Descriptive data mining focuses on finding new and non-trivial patterns from existing datasets, whereas predictive data mining uses some attributes of the existing dataset to predict unseen values of other attributes of interest. For example, National Basketball Association (NBA) uses data mining to analyze the movements of players to help coaches plan their strategies. An analysis of the game played between the New York Knicks and the Cleveland Cavaliers on January 6, 1995, revealed that when Mark Price played the guard position, John Williams attempted four jump shots and made each one (Encyclopedia.com, 2002). The process of data mining makes it possible for coaches to design strategies at their fingertips, instead of spending hours looking at the video footage to extract this piece of information. Similarly, an analysis of data collected from grocery stores uses data mining to reveal an interesting pattern in their stores - customers who buy diapers also buy beer (Pang-Ning et al., 2005). In both the above examples, useful information is extracted from the general properties of existing data using descriptive data mining. Association rule mining is one such descriptive mining technique that extracts rules from existing data to define relationships among them (e.g. Diapers -> Beer). Yet another descriptive mining method is clustering, that identifies meaningful groups embedded in the existing data such that each group shares common characteristics. For example, all tasks in a C programming course can be partitioned into two clusters based on their difficulty level - cluster 1 consists of all tasks that are easy, whereas cluster 2 holds all the difficult tasks. These clusters will be meaningful for instructors when they assign tasks to students based on their capability. In predictive data mining, the objective is to predict the value of a particular attribute (typically called as target or class label attribute) based on the values of other existing attributes. For example, a data mining model can be built to predict whether a student registered in a programming course is able to get an A grade or not, based on other attributes such as the student's age, demographics and previous grades. In this example, the target attribute can be classified as 2 classes (grade=A / grade=notA). Such a predictive mining method, where the target attribute is categorical, is known as classification. If the target attribute is real-valued, then the predictive method is known as regression.

Like many other application areas, ITS also adapt data mining techniques to perform functions such as automatically capturing student actions and making inferences on them (Wu et al., 2008; SOMYUREK, 2009; Romero & Ventura, 2010; Chaturvedi & Ezeife, 2012). Decision trees, K-nearest neighbors and Bayesian Networks (BN) are the most commonly used predictive mining methods in ITS systems (Romero & Ventura, 2010; Chaturvedi & Ezeife, 2012). Chapter 2 describes some the data mining techniques used in ITS and those that have been adapted by this thesis.

1.4 Example-based learning

Example-based learning (EBL) (Gog & Rummer, 2010; Renkl, 2014) is a well-known teaching strategy in traditional educational systems. EBL focuses primarily on students devoting their time and cognitive capacity to studying worked-out examples so that they can enhance their learning and apply it to similar problems or tasks. Figure 1.1 shows a worked-out example of a C program that finds the area of a triangle. Renkl (2014) states that irrelevant cognitive load presented to students must be reduced to improve the effectiveness of worked-out examples, so that students can devote their time and memory capacity to successfully completing assigned tasks. Cognitive overload occurs when the volume of information supplied to a student (e.g. extraneous material) exceeds his/her processing capability (Mayer & Moreno, 2003) and therefore makes learning ineffective. Following these principles, this thesis designs a framework for an EBL-based ITS that presents to students, a concise list of only those worked-out examples that can help them succeed in the assigned task.

1.5 Motivation: Why EBL-based ITS?

There is no denying the fact that there is a shift in paradigm from in-class to web-based online learning environments such as ITS. An ITS aims to provide one-on-one tutoring in real-time to each and every student in a class, irrespective of the class size. Similar to (human) teachers, ITS also adapts an instructional strategy such as example-based learning to tutor the domain. Renkl (2014), Gog and Rummer (2010) demonstrate that example-based learning strategy which promotes learning from worked-out examples is very effective. In an ITS, worked-out examples are found to be useful to both students and teachers - to students in their effort to succeed in assigned tasks or in their effort to master a LU and improve their learning; to teachers in their effort to provide effective teaching and achieve higher success rate in their course. Research on existing EBL-based ITS (Yudelson & Brusilovsky, 2005; Li & Chen, 2009; Hosseini & Brusilovsky, 2013, 2014) has gone a long way in simulating the role of a teacher in many ways, but there are still concerns about design of a formal framework that can extract features from domain examples and tasks in terms of basic learning units, represent them in an efficient and scalable manner and present a personalized list of examples to students.

1.6 Thesis Contributions

This thesis shows how domain and student data gathered from different aspects of an ITS can be analyzed and mined to improve the overall learning experience. The main goal of this thesis is to apply data mining methods for recommending course materials such as worked-out examples to students to enhance their learning. Although existing EBL-based ITS (Yudelson & Brusilovsky, 2005; Li & Chen, 2009; Hosseini & Brusilovsky, 2013, 2014) attempt to recommend worked-out examples to students, they suffer from serious limitations (discussed in chapter 2 section 2.2 and chapter 3 section 3.2). This thesis builds a framework for an EBL-based ITS called Example Recommendation System (ERS) which is partially domain-independent and easily extendable to other domains. ERS builds and manages a highly organized repository of worked-out examples to enable effective retrieval of such examples customized for each task in the domain. Thus, this thesis, and published work originating from it (Chaturvedi & Ezeife, 2012, 2013, 2014, 2015b; Chaturvedi et al., 2015c; Chaturvedi & Ezeife, 2015a) contribute to research in areas such as data mining and intelligent tutoring systems. The main functional contributions are listed in section 1.6.1 and algorithms used to achieve these functionality are listed in section 1.6.2. Table 1.1 lists all algorithms proposed in this thesis and their description. Figure 1.2 presents the map of all algorithms used in the design and evaluation of ERS.

1.6.1 Functional Contributions

The functionality that this thesis adds to existing systems include the following.

1. A simple, less resource-intensive and easily extendable automated method of knowledge extraction (KE) is proposed, that extracts learning units (LU) from given worked-out examples and task solutions using regular expression analysis. The proposed method mitigates the need for highly trained experts with complex knowledge on syntax trees (as is required by existing systems (discussed in chapter 2 section 2.2.2)). It also allows for each example / task solution to be represented uniformly as n-feature vector of binary values (1 for presence of an LU in it / 0 for absence), where n is the total number of LUs in the domain.

2. This thesis builds a knowledge customization (KC) module for ERS, that generates a customized list of worked-out examples focused towards the tasks students are assigned. Such a focused and concise list enables ERS to reduce cognitive overload on students and help them succeed in tasks with a much higher likelihood.

3. All worked-out examples in the domain of ERS are organized into non-traditional but coherent groups based on the LUs they contain. Such a list, generated by the knowledge organization (KO) module of ERS, can be very useful to students when preparing for final examination.

4. The proposed ERS (Example Recommendation System) framework clearly defines and integrates the basic components (KE, KC and KO) required by any EBL-based ITS system that aims to enhance student learning. Such modularity adds

transparency between KE and other modules (KC and KO), thereby making ERS domain-independent, once KE is executed.

5. In an attempt to evaluate ERS on selection of features from its student and domain model, this thesis builds a highly accurate predictive mining model that predicts student performance in an EBL-based ITS. Raw data from domain model and student model is engineered carefully and transformed into meaningful features that are task-focused and objective. This allows to evaluate ERS by answering questions such as "What is the likelihood that students will succeed in assigned tasks using the customized list of worked-out examples suggested by the ITS?".

6. This thesis began its journey by mining student data in online courses to validate that unsupervised instruments such as course assignments can also have a great impact on the overall student performance. This enables teachers to take informed decisions regarding their teaching strategy. For example, our study indicates that allocating 30% to assignments has a greater impact on student learning and their overall grades, as opposed to a high weight of 50% or low weight of 10%.

1.6.2 Procedural Contributions

The different procedural contributions proposed in this thesis to achieve the discussed functionality are listed below.

1. To achieve functionality (1) and integrate it with (4), we build an algorithm called KERE (Knowledge extraction using Regular expressions) that uses regular expression analysis to automatically extract LUs from task solutions and worked-out examples and store them as binary vectors of size n, where n is the total number of LUs in the system.

2. We propose an algorithm MGREPD that mines all worked-out examples in ERS's domain to select those that are most relevant for each task in the domain. The similarity function used in MGREPD is Jaccard's coefficient (Jaccard, 1901), which

caters best to the asymmetric binary vectors generated by KERE. This algorithm is used to achieve functionality (2) and (4).

3. This thesis introduces a novel algorithm called findDL that computes the expected difficulty level (DL) of any task or worked-out example in the domain, after KERE has broken them down into individual LUs. DL is then used to validate MGREPD. DL is also used as a significant feature in functionality 6. In the absence of such an algorithm, ITS domain experts are required to manually assign a DL to each task and worked-out example. Thus, having an algorithm compute the DL significantly reduces the efforts required by the ITS domain experts. Algorithm findDL is also used to achieve functionality (2).

4. To achieve functionality (3) and (4), the thesis proposes an algorithm called KOM16 that modifies two important steps of the standard k-means clustering algorithm for any dataset that is binary and asymmetric. First, it makes a neighborhood-sensitive choice of initial centroids (instead of picking them randomly, as done in standard k-means). Secondly, it recomputes its centroids using a new technique that is sensitive to the presence of LUs, both locally (in the cluster) and globally (in the entire dataset).

5. To achieve functionality (5), this thesis builds a highly accurate data mining model called PSP (Predicting Student Performance) to predict student performance in assigned tasks as a measure of evaluating ERS's student and domain model features that are carefully engineered to be task-focused and objective.

6. We propose an algorithm called MineLearning that mines the impact of unsupervised course instruments such as assignments on student performance using association rule mining. Taking this study a step further, we also proposed and implemented an algorithm called MBER (Mining Binary data Efficiently using Reduced AND operation) that mines Boolean data using matrix algebra. These two algorithms were used to achieve functionality (6).

Algorithm Name	Description	Algorithm number
KERE	Knowledge extraction using regular expressions	2
KOM16	Modified steps 1 and 2.2 of k-means - main	3
create_initial_centroids	Modified step 1 of k-means	4
recompute_centroids	Modified step 2.2 of K_means	5
GREPD	Generate Relevant Examples and Predict Difficulty of a task	6
findDL	Compute actual class label (difficulty level E/D of examples)	7
MGREPD	Modified algorithm to Generate Relevant Examples and Predict Difficulty of a task	8
ERS_main	Main algorithm of the proposed ERS system	9
MBER	Mining Boolean data using reduced AND operations	(Chaturvedi & Ezeife, 2015a)
GSE	Grade in the suggested example	10

Table 1.1: List of algorithms proposed by this thesis

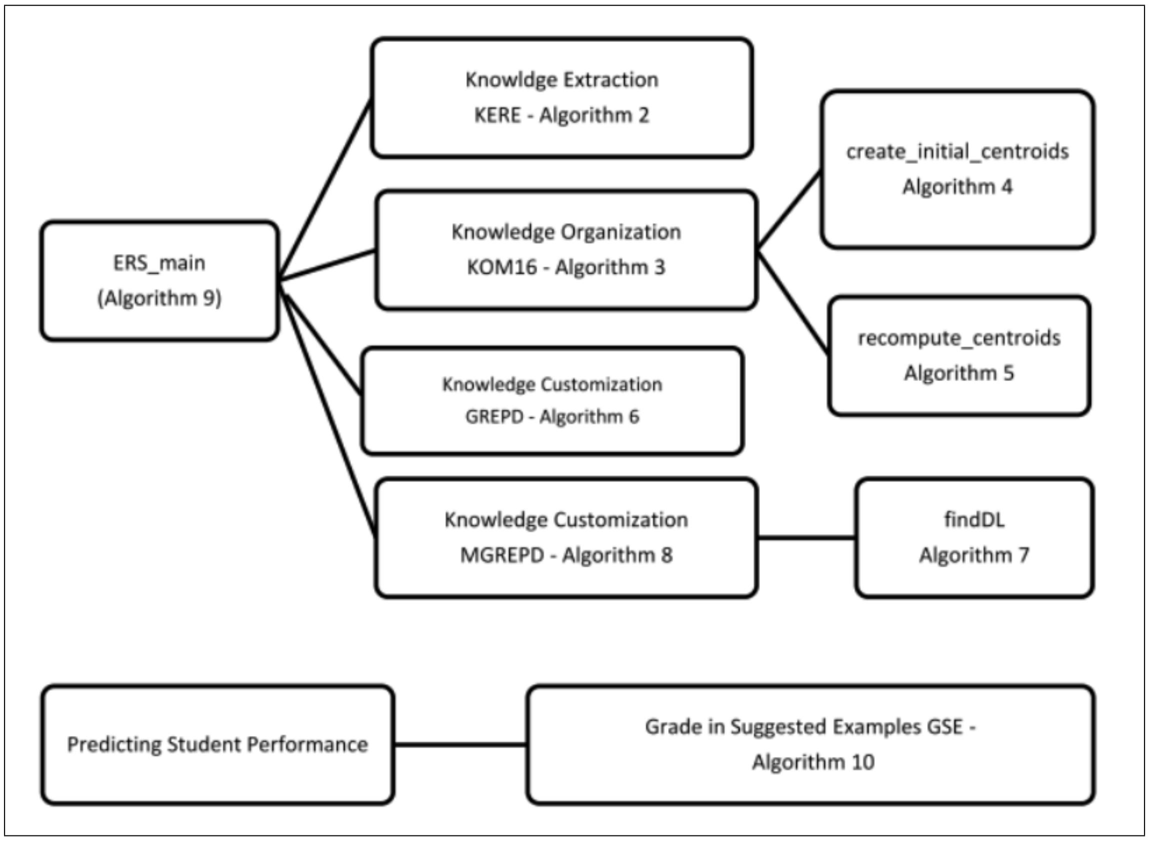


Figure 1.2: Map of all algorithms used for the design and evaluation of ERS

1.7 Thesis Outline

Figure 1.3 presents the layout and flow of chapters 1 to 8 of this thesis. Chapter 2 presents the background required to achieve the functionality of this thesis. It discusses the data mining methods that are applicable to ITS systems. It also presents a survey of the existing ITS systems that use EBL instructional strategy. Chapter 3 presents the problems we identify in the existing systems, leading to the motivation behind this research and our thesis statement. It also gives all definitions used in the thesis and the scope of the proposed ITS. Chapter 4 describes the proposed system and defines all components and algorithms proposed to achieve ERS's functionality. Chapters 5, 6 and 7 evaluate ERS in different ways - chapter 5 evaluates the different components of ERS, chapter 6 evaluates ERS as a tutor and Chapter 7 evaluates ERS using its domain and student model features. Chapter 8 presents the conclusions, limitations and future works emerging from this thesis.

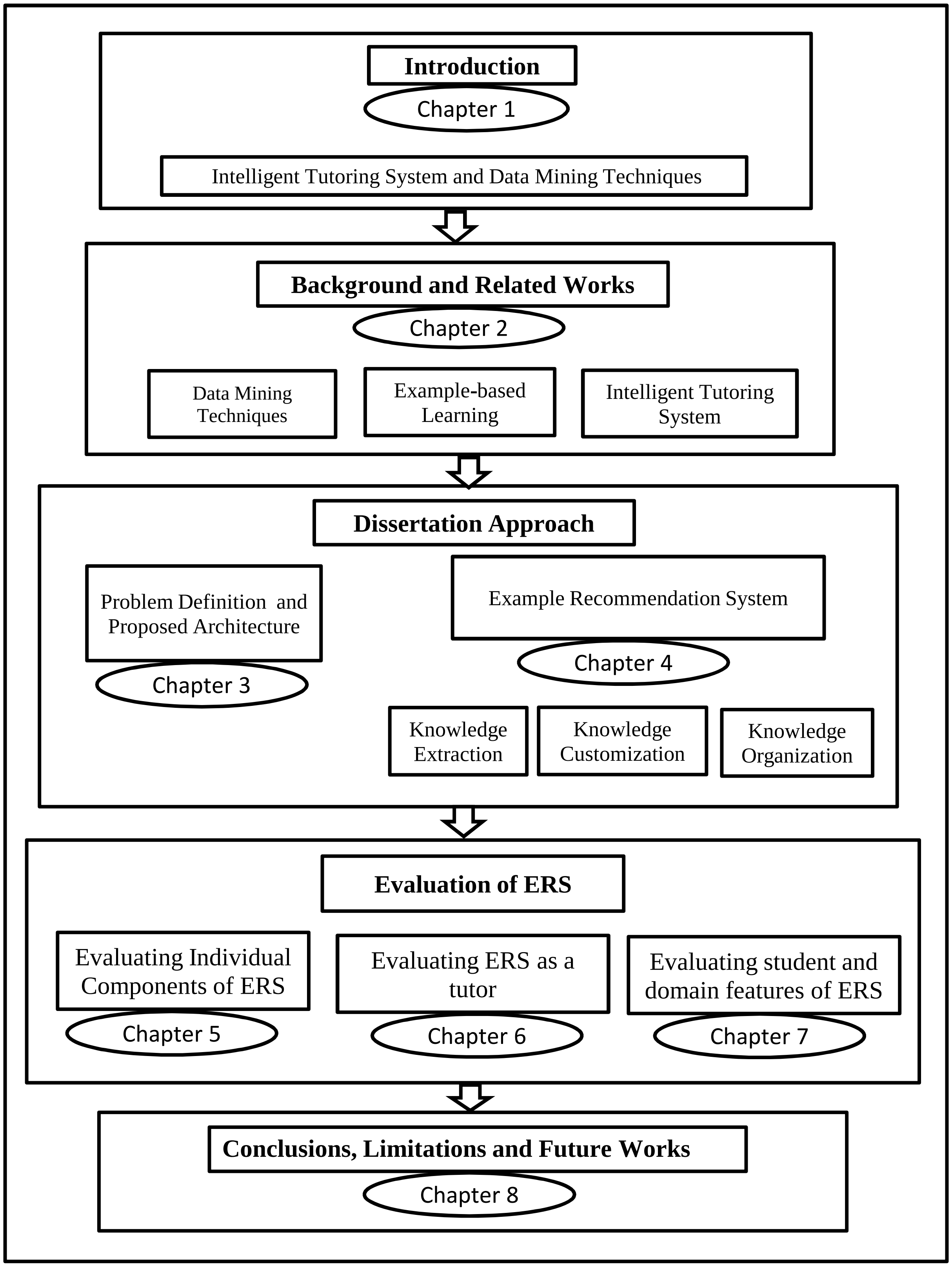

Introduction
Chapter 1
Intelligent Tutoring System and Data Mining Techniques
Background and Related Works
Chapter 2
Data Mining Techniques
Example-based Learning
Intelligent Tutoring System
Dissertation Approach
Problem Definition and Proposed Architecture
Chapter 3
Example Recommendation System
Chapter 4
Knowledge Extraction
Knowledge Customization
Knowledge Organization
Evaluation of ERS
Evaluating Individual Components of ERS
Chapter 5
Evaluating ERS as a tutor
Chapter 6
Evaluating student and domain features of ERS
Chapter 7
Conclusions, Limitations and Future Works
Chapter 8

Figure 1.3: Thesis Layout

Chapter 2

Background and Related Works

Section 2.1 of this chapter discusses the data mining techniques that are typically implemented in ITS (e.g. clustering and classification). Section 2.2 elaborates on the existing ITS that are based on Example-based learning (EBL) teaching methodology, highlighting the limitations that these systems suffer. Section 2.3 discusses the general evaluation methods used in ITS systems and the challenges ITS face when it comes to small student datasets.

2.1 Data Mining Techniques applicable to ITS

The advent of web-based courses has seen a tremendous increase in electronic datasets generated from student and teacher activities and various aspects of course design. Educational data mining (EDM) is a field that exploits various statistical and data-mining techniques over different types of educational datasets (Romero & Ventura, 2010). The objective of using these techniques is to analyze such data in order to resolve issues such as course curriculum sequencing, course instruments that impact student's learning and performance, learning strategies that suit a student's personality and other such issues. Section 1.3 of chapter 1 describes data mining as the process of discovering interesting and useful patterns and relationships in large volumes of data. It also categorizes mining activities into descriptive and predictive. Descriptive data mining focuses on finding new and non-trivial patterns from existing datasets, whereas predictive data mining uses some attributes of the existing dataset to predict unseen values of other attributes of interest. The descriptive and predictive mining techniques commonly used in EBL-based ITS and applicable to ERS are described next in sections 2.1.1, 2.1.2 and 2.1.3.

2.1.1 Clustering

Clustering is an unsupervised mining method that partitions a finite set of data samples in multidimensional space into well-defined and separate clusters using distance measures such as Euclidean distance so that (1) data samples belonging to the same cluster are similar (intra-cluster) and (2) data samples belonging to different clusters are dissimilar (inter-cluster). For example, in an educational dataset, students can be grouped into two or more clusters according to their learning styles. A simple and effective clustering algorithm called k-means (Pang-Ning et al., 2005) is described next.

K-means

K-means algorithm (Pang-Ning et al., 2005) takes as input an integer value k (where k = number of desired clusters) and n data samples, where each sample has m attributes (e.g. each worked-out example in the domain of C programming is a data sample that consists of m learning units as its attributes). It then groups its samples into k clusters (where each cluster consists of one or more data samples) such that the inter-cluster similarity of the resulting clusters is low, whereas the intra-cluster similarity is high. Each group or cluster has a representative point known as its centroid or center. Intra-cluster similarity defines how close the samples within a cluster are to each other, whereas inter-cluster similarity defines how well-separated the cluster centroids are from each other. A similarity or distance function (similarity is considered to be the inverse of distance) is used to find the closeness between a sample x and cluster centroid c or between 2 cluster centroids. For example, Euclidean distance between sample x and cluster center c_i is defined as d(x, c_i)= $\sqrt{(x_1 - c_{i1})^2 + (x_2 - c_{i2})^2 .. + (x_m - c_{im})^2}$, where m = number of attributes in a sample. Algorithm 1 shows the main steps of k-means algorithm and is explained further using example 1.

Algorithm 1 K-means algorithm (Pang-Ning et al., 2005)

```
Input: dataset of size n X m (n samples, each with m attributes), k (number of clusters),
maxIterations (threshold for maximum number of iterations)
Output: k clusters
Method
*** begin of k-means
1. Choose k samples as initial centroids
2. repeat until convergence (centroids not not change or maximum number of iterations
has been reached)
    2.1. repeat steps until all n samples are exhausted
     2.1.1. assign sample x to its closest centroid using an appropriate distance function
    2.2. recompute the centroid of each cluster based on assignment in steps 2.1
*** end of k-means
```

Example 1: Use k-means clustering to group a sample data of 4 students (A,B,C,D) with their height and weight as attributes into k=2 clusters.

Input: k=2 and a dataset S of 4 students given their height and weight.

sample data S =

	Height	Weight
A	1	1
B	2	1
C	4	3
D	5	4

Output: Dataset S organized into 2 clusters.

Solution: K-means starts by randomly picking 2 samples as initial cluster centers. For example, $c1$ = A = (1,1); center $c2$ = B = (2,1).

It then assigns samples to their closest cluster center based on its distance to the center. Euclidean distance between each sample and cluster centers $c1$ and $c2$ is calculated and stored in a matrix M_1(for iteration 1). For example, distance between data sample C (4,3) and center $c2$ (2,1) is calculated as $\sqrt{((4-2)^2 + (3-1)^2}$= 2.83.

$$M_1 = \begin{array}{ccccc} c1 & 0 & 1 & 3.61 & 5 \\ c2 & 1 & 0 & 2.83 & 4.24 \\ & A & B & C & D \end{array}$$

Now, each sample is assigned to the center to which it has the shortest distance => $A \in c1, B \in c2, C \in c2, D \in c2$.

Next, k-means finds new cluster centers by finding the arithmetic mean of existing clusters. $c1$ has 1 sample with attributes (1,1) => new mean for cluster 1 = (1,1); $c2$ has 3 samples in it; B(2,1), C(4,3) and D(5,4) => new mean for cluster $c2$ is ((2+4+5) / 3, (1+3+4) / 3)) = (11/3, 8/3). Therefore, the new centers are $c1$ = (1,1) and $c2$ = (11 / 3, 8 / 3). The above method is repeated until convergence. So samples are again compared with these new cluster centers and are (re)assigned to those clusters for which their distance is minimum. In this example, the euclidean distance between each sample A,B,C,D and these new cluster centers is calculated and stored in a matrix M_2 (iteration 2).

$$M_2 = \begin{array}{ccccc} c1 & 0 & 1 & 3.61 & 5 \\ c2 & 3.14 & 2.36 & 0.47 & 1.89 \\ & A & B & C & D \end{array}$$

Once again, each sample is assigned to the center to which it has the shortest distance => $A \in c1, B \in c1, C \in c2, D \in c2$.

Repeating this for the next iteration : $c1$ has 2 elements A(1,1) and B(2,1) =>mean for $c1$= ((1+2)/2,(1+1)/2) = (3/2,1); $c2$ has 2 elements C(4,3) and D(5,4) => mean for $c2$ is ((4+5)/2, (3+4)/2) = (9/2, 7/2). With these new centers, distances are calculated and stored in M_3

$$M_3 = \begin{array}{ccccc} c1 & 0.5 & 0.5 & 3.2 & 4.6 \\ c2 & 4.3 & 3.54 & 0.71 & 0.71 \\ & A & B & C & D \end{array}$$

Assigning each sample to the closest center ($A \in c1, B \in c1, C \in c2, D \in c2$), it is found that this assignment is the same as cluster assignment in iteration 2. So the algorithm stops. The output for this example is ($A \in c1, B \in c1, C \in c2, D \in c2$). New data can then be tested by keeping the centers fixed.

Limitation of using k-means clustering algorithm is that it is sensitive to initialization parameter k, to the initial set of centers picked and to the presence of noise or outliers. Our proposed algorithm that uses clustering to organize its worked-out examples (discussed in Chapter 4) mitigates some of these issues.

2.1.2 Classification / Prediction

Given dataset of samples, each represented as a tuple (x,y) where $x = (x_1, x_2, ..x_n)$, classification is the task of building a learning model that maps the attribute set x into attribute y, where y is termed as its class label (or target attribute). In general, data is divided into 2 subsets: training and test. Training dataset is used to build/learn the model, whereas test dataset is used to test it. First the model is built by applying a classification/learning algorithm (such as k-nearest neighbors as explained next) on the training dataset. Next, the model is applied to test dataset and their actual class labels are compared to the predicted ones to evaluate the model. Thereafter, this model can be used to classify unseen records.

2.1.2.1 K-Nearest Neighbors

K-nearest-neighbor (k-nn) classifier takes 4 inputs in order to predict a test sample's target attribute. The inputs are (1) an integer k (k=number of neighbors), (2) set of training samples whose class label y is known (e.g. set of p worked-out examples, given y = difficulty level of each example in p), (3) test sample t (e.g. t = worked-out example p+1), and (4) similarity or distance function (e.g. Euclidean distance). It then predicts test sample t's class label by performing the following steps :

a. calculate similarity between the test sample t and all training samples using the chosen similarity function

b. sort these similarity values and pick the top k samples – these are the k nearest neighbors of test sample t

c. determine the class label of each of the k nearest neighbors

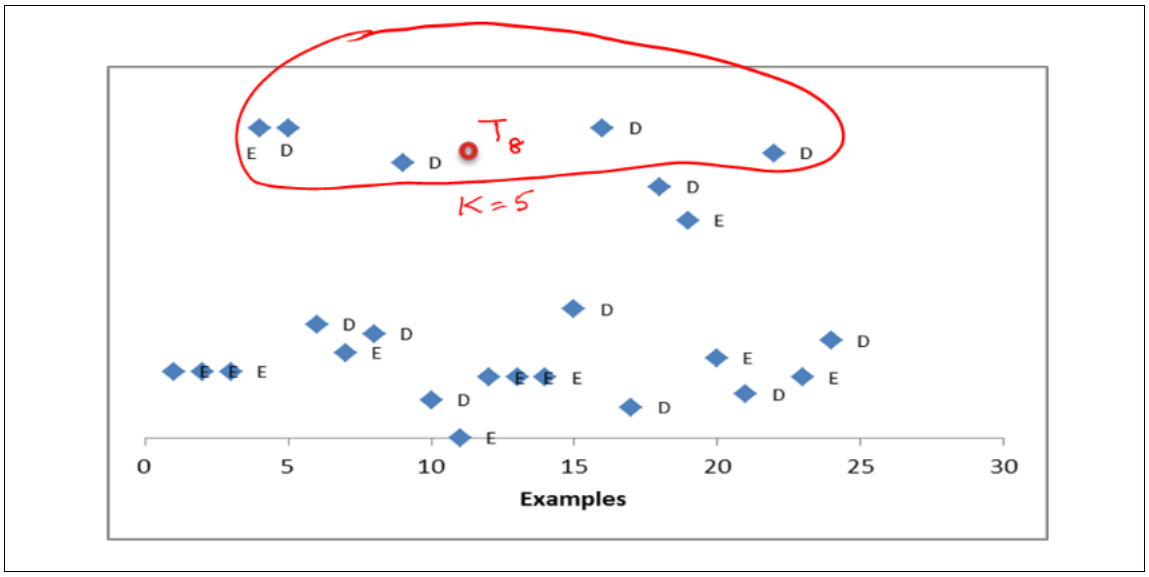


Figure 2.1: Difficulty level of task T_8 using k=5 nearest neighbors is predicted as D (difficult)

d. find the class label that gets the majority votes among its neighbors (majority implies that the number of neighbors with this class label is more than any other class label) and assign it as the class label of test sample t.

For example, if inputs to k-nn are (1) k = 5, (2) training dataset = m worked-out examples, each example e_i represented as (x,y) where $x = (x_1, x_2, ..x_n)$, n = total number of LUs, each x_i is a learning unit of e_i, class label y for each e_i is its difficulty level (class label = E for easy and D for difficult), (3) test sample t = task T_8 as shown in figure 2.1 and (4) chosen similarity function = Jaccard's coefficient of similarity(Pang-Ning et al., 2005). The k-nn algorithm first finds the 5 nearest neighbors of test sample T_8 and then predicts T_8's class label of to be D (difficult) because majority of its neighbors (4 out of 5) from the training set have a class label (difficulty level) = D.

2.1.2.2 Decision Trees

Decision Tree (DT) solves a classification problem by asking a set of carefully framed questions about the attributes in a data sample t. Each question leads to a follow-up question until a conclusion can be made on t's class label. A tree structure (DT) is used to organize these questions and their answers - each step partitions the data based on a question asked on a chosen attribute. DT is a supervised classification method that uses a greedy strategy to grow a tree by making locally optimum decisions about which attribute to use to partition data. Hunt's algorithm (Pang-Ning et al., 2005; Markov & Larose, 2007), the basis of some well known DT methods such as ID3, C4.5 and CART creates a DT recursively . Assume that a node t in the tree is associated with a set of training records D_t and (y_1,.. y_c) are their class labels, Hunt's algorithm does the following: (1) If all records in D_t belong to the same class y_c, then a leaf node is created labeled as y_c (2) If D_t belongs to more than one class, then an attribute α is selected such that it gives the best split and a test condition on α is used to partition the records in D_t to smaller subsets. Each record of D_t will now belong to one of the child nodes created for each outcome of α's test condition. This is then repeated recursively for each child node. To justify that an attribute α gives the best split, different measures such as Gini index, entropy and classification error rate can be applied to attributes so that the selection gives the best split of records - the lower the measure, the better is the split. For example, attribute α that gives the lowest Gini index is the one that gets picked and the test conditions for the current node are then applied on α. The measures are defined as :

Entropy(t) = - $\sum_{i=0}^{m-1}$ p_i $\log_2 p_i$

Gini(t) = 1 - $\sum_{i=0}^{m-1}$ ${p_i}^2$

Classification error(t) = 1 - $\max_i[p_i]$

Using any one of these measures (e.g. Gini index), the steps to create a decision tree are:

1. Find the gini index of each attribute of sample S.

2. The attribute with minimum gini index is selected as the root (α).

3. Attribute α has β number of distinct values. So root node α will have β branches coming out of it. A reduced set of samples will be now tested for each of the β branches and the gini index of each attribute except α is calculated to find the next attribute that gives the best split. This process is repeated until all samples for a given node belong to the same class or there are no samples for branch β, which is when a leaf node is created with a class label .

4. Decision rules are generated using the tree - all branches in a path from the root to the node prior to the leaf are considered as antecedents of the rule (separated by a condition) and the leaf (with class label) is considered to be the consequent.

Example 2: Use decision trees to predict the class label of a dataset S

Input : Dataset S with 2 attributes a_1 and a_2 and a class label attribute $Class$.

Let S =

	a_1	a_2	$Class$
1	T	T	$G1$
2	T	T	$G1$
3	T	F	$G2$
4	F	F	$G1$
5	F	T	$G2$
6	F	T	$G2$
7	F	F	$G2$
8	T	F	$G1$
9	F	T	$G2$

Output : classify a data row into its class label (G1 or G2)

Solution:

In this example, m=2 as attribute Class has 2 labels G1 and G2.

First, an attribute is selected for the root node, say G1. But this tree needs to be modified as the root contains records from both classes G1 and G2. To select an attribute, we use the Gini index to find the best split.

For attribute a_1, Gini index = $\frac{4}{9}[1 - (3/4)^2 - (1/4)^2] + \frac{5}{9}[1 - (1/5)^2 - (4/5)^2] = 0.3444$

For attribute a_2, Gini index = $\frac{5}{9}[1 - (2/5)^2 - (3/5)^2] + \frac{4}{9}[1 - (2/4)^2 - (2/4)^2] = 0.4889$

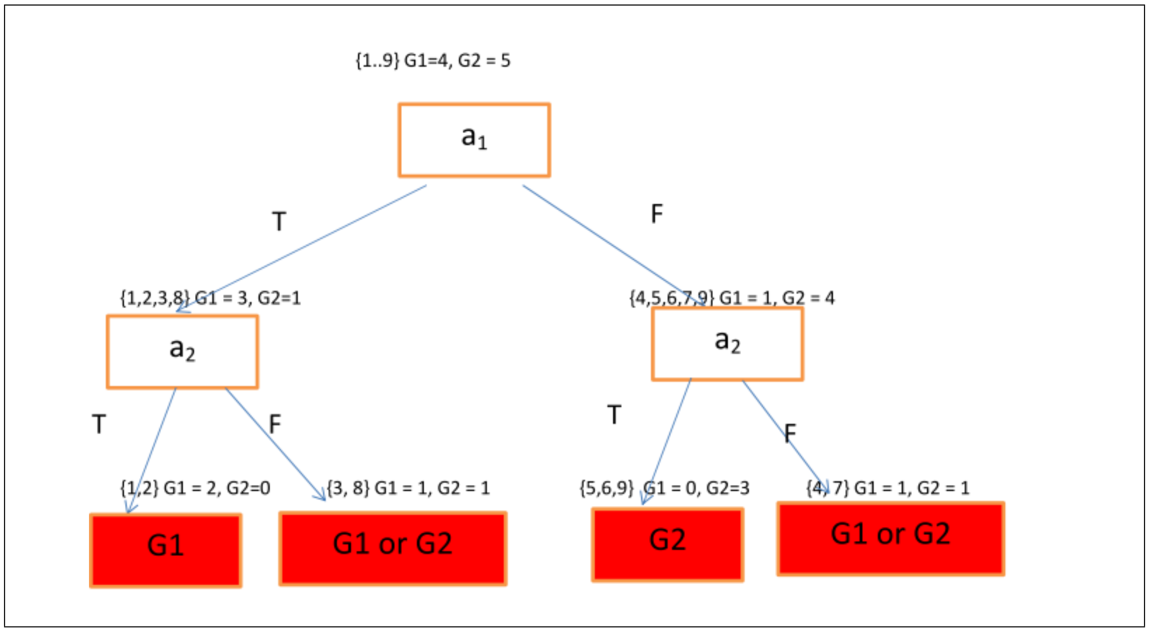


Figure 2.2: Decision tree generated for example 2 of section 2.1.2.2

Since attribute a_1 is smaller, it gives a better split. So the root is a_1. Leaf nodes with class label are shown in red.

Some decision rules generated from this tree are:

if attribute a_1= T and attribute a_2= T then classification = G1;

if attribute a_1= F and attribute a_2= T then classification = G2;

So an input record such as (F, T) will be classified as G2, according to this decision tree.

Decision tree algorithm such as ID3 (Quinlan, 2014) are not able to handle continuous attributes such as temperature. C4.5 (Quinlan, 2014), an improvement of ID3, is able to handle such attributes. Both ID3 and C4.5 use entropy and information gain as measures to select an attribute for a node. CART (Classification and Regression trees) method is a restricted version of ID3 and C4.5 because it allows only binary splits. ID3 and C4.5 also allow attributes to have multi-ways splits (e.g. an attribute can have 3 outcomes, 'True', 'False' and 'Maybe') . Unlike Id3 and C4.5, CART uses the gini index.

2.1.3 Frequent Pattern Mining

Frequent patterns (Han & Kamber, 2000; Han et al., 2004) are itemsets or subsequences that appear in a data set with frequency no less than a user-specified threshold. For example, a set of items, such as coffee and cream that appear frequently together in a transaction data set is a frequent itemset. A subsequence, such as buying a PC is followed by buying a printer, is a (frequent) sequential pattern if it occurs frequently in the shopping cart history. Finding frequent patterns plays an essential role in mining associations and many other interesting relationships among data. The most common form of pattern discovery in an unsupervised machine learning system is the discovery of association rules such as (if coffee, then cream, written more commonly as coffee => cream). Usefulness of such rules is measured by calculating their support and confidence. Support of a rule is a measure of how often that rule applies in the dataset. For example, a rule R1: "Assignment $>$ 85" $\Rightarrow$"Total $>$ 85" has a support of 5% means that "Assignment>85" and "Total>85" exist together in 5% of all the transactions being analyzed. Confidence of a rule is a measure of how often that rule is correct. For example, rule R1 has a confidence of 80% means that 80% of transactions in the dataset that contain "Assignment>85" also contain "Total>85". The Apriori algorithm proposed by Agarwal Srikant (1994) for mining frequent itemsets is based on a property which states that all non empty subsets of a frequent itemset must also be frequent (Apriori property). Each transaction in the database that is input to the Apriori algorithm consists of several attributes (items) ; a set of items is called an itemset. The problem of finding itemsets that are frequent is considered to be a non-trivial problem because the potential number of such frequent itemsets is exponential to the number of items in the database. Apriori algorithm scans the database several times (depending on the size of the largest frequent itemset) and therefore, is computationally expensive. Several variations to this algorithm intending to make it more efficient and scalable exist and are presented next.

itemset	support
{a}	3
{b}	4
{c}	3
{d}	1

Table 2.1: Output of 1st iteration of Apriori - candidate 1-itemset and their support

itemset	support
{a,b}	3
{a,c}	2
{b,c}	3

Table 2.2: Output of 2nd iteration of Apriori - candidate 2-itemsets and their support

2.1.3.1 Apriori And Its Variations

Apriori algorithm works in two phases. The first phase discovers the itemsets that are frequent (called large itemsets). By frequent, it means that they have occurred together more than given minimum support number of times.

Example 3: Given a transaction dataset D with four tuples

{(Tid1: ab);(Tid2:abcd);(Tid3:abc);(Tid4:bc)} and a minimum support of 75% or 3 transactions (indicating that if an itemset i appears in at least 3 transactions, then i is frequent), find all large itemsets (L) or frequent patterns (FP) using Apriori algorithm.

Input : D = {(Tid1: ab);(Tid2:abcd);(Tid3:abc);(Tid4:bc)}, min_support = 3

Output : Large itemsets

Solution:

Step 1: Find itemsets of size 1 and their support count. These are known as candidate 1-itemsets.

Since itemset {d} has a support count <3, it is not a large 1-itemset => Large or frequent 1-itemsets are {{a}, {b}, {c}}.

Step 2: Next, generate a list of all pairs of the frequent 1-itemsets. This is the candidate 2-itemset.

itemset	support
{a,b,c}	2

Table 2.3: Output of 3rd iteration of Apriori - candidate 3-itemsets and their support

Itemsets {a,b} and {b,c} have a support count >=3 and satisfy the Apriori property. Therefore, Large or frequent 2-itemsets are {{a,b}, {b,c}}.

Step 3: Similar to step 2, generate a list of all triplets by joining frequent 2-itemsets. This is the candidate 3-itemset.

None of the itemsets in table 2.3 have a support count of 3 or more. This implies that large 3-itemset is empty.

Therefore, the frequent itemsets generated by Apriori are {{a,b}, {b,c}}. Now, association rules can be generated from all frequent itemsets and only strong rules with confidence greater than or equal to a given minimum confidence are kept. Confidence is calculated as cardinality of the rule/cardinality of the left side of the rule. If the given confidence is 80%, example rules that can be generated R1: a -> b (confidence 3/3 = 100%); R2:b -> a (confidence 3/3 = 100%).

Variations of Apriori

To overcome issues with Apriori, such as multiple database scans and multiple joins, several variations were proposed such as hash-based (Park et al., 2005), tree_based (Sarkar et al., 2012), partition-based (Li et al., 2001; Chai et al., 2007), sampling-based, hierarchy (is-a)-based apriori (Han & Kamber, 2000) and others (Yu et al., 2008; Wang & Xiangwei, 2011).

2.2 Example-based ITS

There exist several ITS that revolve their teaching strategies around worked-out examples but they suffer from several limitations. Section 2.2.1 presents these systems as a taxonomy. Sections 2.2.2 and 2.2.3 list the main characteristics of some existing EBL-based ITS that are designed for programming and non-programming domains.

2.2.1 Taxonomy of Example-based ITS

This study surveys the existing ITS that are solely based on the theory of example-based learning to conclude that they differ from each other in at least the following ways:

1. Expertise required to create the domain model.
2. Granularity of the domain used. Granularity refers to the smallest level of detail that a domain component such as a worked-out example or task is divided into. For example, "Lesson $\supseteq$ Examples" is a course-grained system, interpreted as "each lesson consists of examples (that are treated as the smallest atomic units that cannot be further subdivided)". On the contrary, "Lesson $\supseteq$ Examples $\supseteq$ Learning Units or concepts" is a fine-grained system interpreted as "each lesson consists of examples, which are further subdivided into small indivisible concepts" (e.g. L1 consists of example E1, which has concepts {datatype, printf}).
3. Extraction of the smallest unit of granularity (SUG) (called as learning units in this thesis).
4. Data structures used to store the domain's SUG.
5. Organizing resources (such as worked-out examples).
6. Customizing resources (such as worked-out examples) based on gradable tasks students are assigned or need help in.
7. Customizing resources (such as worked-out examples) according to student performance.

Based on these criteria, tables 2.4, 2.5 and 2.6 show a tabular taxonomy of the existing ITS that use example-based learning theory of pedagogy. Most of the existing systems use a programming domain, although there are some that are designed for domains such as Math and Physics. Sections 2.2.2 and 2.2.3 discuss such systems in detail and their limitations. [1]

[1]The existing systems in this thesis are not necessarily fully functional ITS - they are mostly value-added services to existing systems. Nevertheless, they are relevant to my research as they are EBL-based.

	Authors	**ITS**	**Expertise required**	**Granularity (SUG = smallest unit of granularity)**	**Method used to extract SUG**	**Medium of storage used for SUG**	**Customizing resources based on student performance**	**Organizing resources**	**Customizing resources based on assigned task**
1	Burow, Robert, and Weber, Gerhard.	ELM_PE (Burow & Weber, 1996)	-example and task solutions -syntax trees for each example	fine-grained (SUG = domain concept e.g. function definition)	parsing	flat file	None	None	syntactic similarity using syntax trees
2	Weber, Gerhard, and Brusilovsky, Peter	ELM-ART (Weber & Brusilovsky, 2001)	-example and task solutions -syntax trees for each example	fine-grained (SUG = domain concept e.g. function definition)	parsing	flat file	simple matching	Manual (lesson-wise)	syntactic similarity using syntax trees
3	Yudelson, Michael, and Brusilovsky, Peter	NavEx (Yudelson & Brusilovsky, 2005)	-example and task solutions -syntax trees for each example	fine-grained (SUG = domain concept e.g. datatypes in C)	parsing	flat file	simple matching	Manual (lesson-wise)	None

Table 2.4: Tabular taxonomy of Example-based ITS - Part I

	Authors	**ITS**	**Expertise required**	**Granularity (SUG = smallest unit of granularity)**	**Method used to extract SUG**	**Medium of storage used for SUG**	**Customizing resources based on student performance**	**Organizing resources**	**Customizing resources based on assigned task**
4	Zhang, Y.,Capus L., and Tourigny, N.	Sphinx (Zhang et al., 2007)	- example solutions - list of domain concepts	fine-grained (SUG =	manual (given by experts)	flat file	simple matching	None	None
5	Muldner, Kasia, and Conati, C.	EA_Coach Muldner & Conati (2007)	-step-wise solution of each example - rules used in each step of the solutions	coarse-grained (SUG = step / rule)	manual	Bayesian network	probabilistic user model	None	comparing corresponding steps of examples and task
6	Li, Liang-Yi, and Gwo-Dong Chen.	PADS (Li & Chen, 2009)	- example solutions - list of domain concepts	fine-grained (SUG = domain concepts)	manual (using IOC method (Equation 2.1)	relational database	data mining - decision trees	None	None

Table 2.5: Tabular taxonomy of Example-based ITS -(continued) - Part II

	Authors	**ITS**	**Expertise required**	**Granularity (SUG = smallest unit of granularity)**	**Method used to extract SUG**	**Medium of storage used for SUG**	**Customizing resources based on student performance**	**Organizing resources**	**Customizing resources based on assigned task**
7	Mokbel et al.	Mokbel et al. (2013)	-example and task solutions -syntax trees for each example	fine-grained (SUG = domain concepts)	Parsing and spectral clustering	relational database	Tree edit distance	None	None
8	Hosseini and Brusilovsky	JavaParser Hosseini & Brusilovsky (2013)(2014)	-example and task solutions -syntax trees for each example	fine-grained (SUG = domain concepts)	Parsing	flat file	Matching using TFIDF and cosine similarity	lesson-wise	None

Table 2.6: Tabular taxonomy of Example-based ITS -(continued) - Part III

2.2.2 Example-based ITS for Programming Domain

Research has shown that Example-based learning is an effective teaching method (VanLehn, 1998; Gog & Rummer, 2010; Renkl, 2014). One of the earliest ITS systems in the domain of programming that used examples for effective learning was developed by Burows and Weber (1996) [2]. Their domain model consists of all examples in the example database that are pre-analyzed and stored as syntax trees. These trees store concepts and sub-concepts that occur in the solution as interior nodes and leaves as constant values. When a student $s1$ asks for an example while working on a task, the authors compare the syntax trees of the task that is currently assigned to the student with that of each example. The best example that matches the task tree is then presented to the student. The authors use 3 features to match the entries in a task with that of an example : concept/sub-concept similarity, syntactic similarity and organizational similarity. When a concept/sub-concept of a task is matched to that of an example, the highest matching value is given to a perfect match (e.g. float as the return value in the task and float as return value in the example) ; a lower value is given to a match that is generic (e.g. int as a return value in the task and float as the return value in the example is a generic match) and the lowest is given when there is no match. Syntactic similarity is measured by the number of arguments that the concept/sub-concepts in the task and the example use - if they are same, a higher matching value is assigned to it. Organizational or hierarchical similarity is measured by the placement of a concept/sub-concept in the syntax tree. A concept placed high in the tree is a strong indication of the importance of the concept. Therefore, the higher a concept is in this tree, the higher is its similarity value.

Weber and Brusilovsky (2001) used a similar approach to develop an ITS called ELM-ART to teach the programming language LISP. The domain model of ELM-ART consists of each task solution stored as a syntax tree. ELM-ART compares the syntax trees of student solutions with task solutions, in order to assist students in solving a task. Weber and Brusilovsky (2001) proposed yet another web-based teaching and learning tool called WebEx to support learning from examples. WebEx was created for a course on

[2]The existing systems in this thesis are not necessarily fully functional ITS - they are mostly value-added services to existing systems. Nevertheless, they are relevant to my research as they are EBL-based.

'Data Structures and Programming Principles' and was used for interactive exploration of programming examples in language C in an adaptive way. The idea was to avoid overloading students with detailed explanations for each line of the example's code and allow them to see these explanations of the code at their own pace. Student feedback of the WebEx system motivated the authors to design an improved system called NavEx (Yudelson & Brusilovsky, 2005) that provided adaptive navigation support and personal guidance to students. Although NavEx is similar to WebEx in its emphasis on the importance of using examples when teaching programming courses, it allows for more personalized support by adapting examples and explanations to student's current level of knowledge. NavEx's domain model consists of all example solutions represented as syntax trees and the sequence in which lessons are taught. All examples are grouped according to lessons to which they belong. NavEx extracts the basic concepts of each example using syntax trees (similar to ELM-ART and WebEx) and uses them to automatically derive the learning goals of each lesson. Each lesson requires a set of prerequisite concepts to achieve its learning goal, represented by a set of outcome concepts. For example, domain model of NavEx specifies that lesson 1 (L1) has 3 examples E1, E, E3 and lesson 2 (L2) has 3 examples E4, E5, E6. Concepts extracted for these examples as given in small brackets are E1 (C1), E2 (C1, C2), E3 (C1, C2), E4 (C1, C2, C4, C6), E5 (C1, C2, C4) and E6 (C1, C4). Lesson L1 has no prerequisites since it is the first lesson. But it has an outcome of (C1, C2), since by the end of lesson 1, students must have seen examples E1, E2 and E3 and therefore must have learnt these concepts. C1 and C2 are now prerequisites to lesson L2 and after browsing examples E4, E5 and E6 of L2, its outcome concepts are C4 and C6. NavEx uses a d-dimensional vector of Boolean values ('known' / 'not known') to store student's knowledge on each concept c_i(i=1..d), where d is the total number of concepts in the domain. It then simply matches the student's knowledge on an example's prerequisite concepts to decide if this example should be presented to the student or not (e.g. if student s has finished lesson1, then values of C1 and C2 for s are set to 'known'. Therefore, s can be presented with examples E4, E5 and E6). A value of 'unknown' for a concept c_i for student s is changed to 'known' if s has clicked on an example with concept c_i enough number of times, decided by a threshold value computed as : $threshold =$

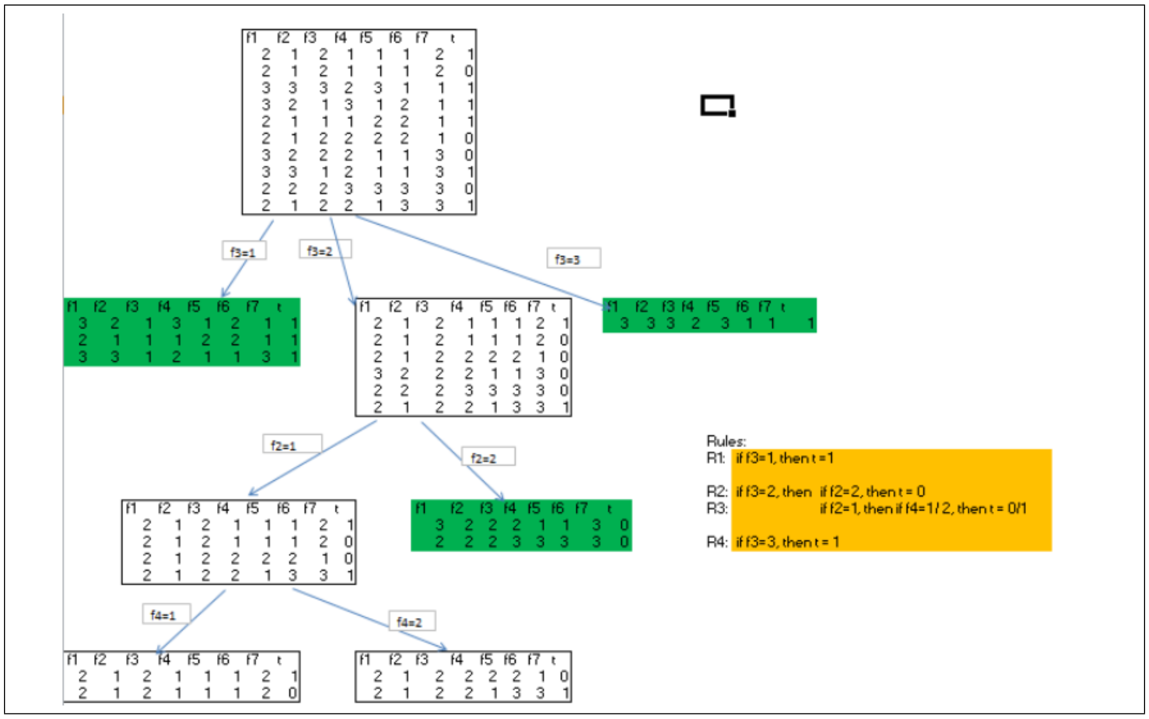
$0.8 * (all\_concepts - prereq\_concepts)/all\_concepts * total\_number\_of\_clicks\_possible$, where $total\_number\_of\_clicks\_possible$ is a value given by an expert. For example, $all\_concepts$ for E5 of lesson L2 is {C1, C2, C4}, mastered concepts for E5 are (C1, C2) and the $total\_number\_of\_clicks\_possible$ for this example (given by expert) is 10. So the threshold is 0.8 * (1/3) * 10 = 2.6 => the student has to make at least 3 clicks to master this example. Once this threshold is reached, the system concludes that the student has mastered example E5 and hence the student model can be updated accordingly.

Li and Chen (2009) [3]use decision trees to select programming exercises to suit the student's knowledge level - the authors call it as the student's zone of proximal development (ZPD). Their system offers challenge and assistance to the students by offering a personalized assistance dispatching system (PADS) to select exercises based on the difficulty level of the exercise and on student's current knowledge. PADS extracts concepts covered by each exercise using a manual method called IOC (Rovinelli & Hambleton, 1976), in which nine experts gave their opinion of whether or not a concept should belong to an exercise. Then, an index value is calculated for each concept i in exercise k as

$$I_{ik} = \frac{(n-1)\sum_{j=1}^{q} X_{ijk} + n\sum_{j=1}^{q} X_{ijk} - \sum_{j=1}^{q} X_{ijk}}{2(n-1)q} \tag{2.1}$$

where n = total number of LUs, q = total number of experts and X_{ijk}= the rating (1, 0, -1) of concept i on exercise k by expert j. An index value $I_{ik} > 0.8$ indicates that concept i is required by exercise k, a value between 0.5 and 0.8 indicates that i is a sub-concept of k and value of < 0.5 indicates that i is not a concept of j. PADS stores student's proficiency in each extracted concept in its student model and uses it to derive fours of its features used in the decision tree model. Two other features are manually given by experts. Feature f7 is captured from the weblogs. The seven feature attributes PADS uses are student's proficiency level in the main concept (f1), student's proficiency level in the sub-concepts (f2), complexity level of the algorithm (f3), number of lines in the code for completing the exercise (f4), student's proficiency level in algorithm analysis (f5), student's grade in the last assignment (f6) and the number of logins and time spent on

[3]The existing systems in this thesis are not necessarily fully functional ITS - they are mostly value-added services to existing systems. Nevertheless, they are relevant to my research as they are EBL-based.



f1	f2	f3	f4	f5	f6	f7	t
2	1	2	1	1	1	2	1
2	1	2	1	1	1	2	0
3	3	3	2	3	1	1	1
3	2	1	3	1	2	1	1
2	1	1	1	2	2	1	1
2	1	2	2	2	2	1	0
3	2	2	2	1	1	3	0
3	3	1	2	1	1	3	1
2	2	2	3	3	3	3	0
2	1	2	2	1	3	3	1

f1	f2	f3	f4	f5	f6	f7	t
3	2	1	3	1	2	1	1
2	1	1	1	2	2	1	1
3	3	1	2	1	1	3	1

f1	f2	f3	f4	f5	f6	f7	t
2	1	2	1	1	1	2	1
2	1	2	1	1	1	2	0
2	1	2	2	2	2	1	0
3	2	2	2	1	1	3	0
2	2	2	3	3	3	3	0
2	1	2	2	1	3	3	1

f1	f2	f3	f4	f5	f6	f7	t
3	3	3	2	3	1	1	1

f1	f2	f3	f4	f5	f6	f7	t
2	1	2	1	1	1	2	1
2	1	2	1	1	1	2	0
2	1	2	2	2	2	1	0
2	1	2	2	1	3	3	1

f1	f2	f3	f4	f5	f6	f7	t
3	2	2	2	1	1	3	0
2	2	2	3	3	3	3	0

f1	f2	f3	f4	f5	f6	f7	t
2	1	2	1	1	1	2	1
2	1	2	1	1	1	2	0

f1	f2	f3	f4	f5	f6	f7	t
2	1	2	2	2	2	1	0
2	1	2	2	1	3	3	1

Figure 2.3: Example decision tree generated by PADS (Li & Chen, 2009)

the logins in the last two weeks (f7). To train the model, the target variable selected is difficulty level of an exercise. It is measured by using expert knowledge (on parameters such as the operators used in the exercise, and number of steps required to solve it) and by student feedback on the exercise. At the end of each exercise, students are asked to answer a question : How did you complete this exercise? The choices given to the students were: 1) I completed this exercise without assistance 2) I completed it with assistance of related materials 3) I completed it with peer collaboration 4) I couldn't complete it. If the answer is either a 1 or a 4, then the target variable is set to 0; otherwise it is set to 1 (=>appropriate difficulty level) . Once trained, this model is used to predict the target t (difficulty level of the exercise) for future students. For example, let S be a sample dataset with 10 tuples and 8 attributes - 7 feature attributes (f1-f7) and 1 target attribute t as shown in the root of figure 2.3. Now if a new student s has the following 7 feature attributes (2 2 2 1 1 1 2) for exercise e$_i$, then it is predicted by rule R2 that e$_i$ is not at an appropriate difficulty level for s.

Mokbel et al. (2013) also use syntax trees to extract concepts from worked-out example solutions. They then divide their solution's syntax trees into sub-graphs using spectral clustering and then measure the proximity between solution parts using TFIDF weights. Hosseini and Brusilovsky (2013), in an attempt to create an indexing tool called JavaParser for Java problems, use the same method of syntax trees used by ELM-ART (Weber & Brusilovsky, 2001) and NavEx (Yudelson & Brusilovsky, 2005) to extract concepts from the given example solutions. After extracting the concepts, their system finds sequence of problems that can assist students to fill gaps in their knowledge, especially when preparing for final examination. Hosseini and Brusilovsky (2014) **??**, also propose to analyze concepts extracted from a domain's learning content in an effort to find similar problems or examples by comparing their syntax trees using measures such as tree edit distance (Zhang & Shasha, 1989).

Limitations

After doing a systematic literature survey of existing ITS, this thesis divides the entire process of designing an ITS that is based on EBL teaching method into different phases such as knowledge extraction (KE), knowledge organization (KO) and knowledge customization (KC). Knowledge extraction (KE) is defined as a process that extracts knowledge from given inputs (e.g. extract learning units (LUs) from given C programs). Knowledge extraction in many of the systems such as ELM-ART (Weber & Brusilovsky, 2001), NavEx (Yudelson & Brusilovsky, 2005) and JavaParser (Hosseini & Brusilovsky, 2013, 2014) is done using syntax trees. This entails a complex expert knowledge using complex automated methods such as parser and syntax tree generation that is not easily defined or maintained, nor can it be applied or extended to other subject domains. Other systems such as PADS (Li & Chen, 2009) use a manual method (IOC) of extraction. This study defines knowledge customization (KC) as a process that generates resources customized to the needs of students using the ITS. A limitation of customization in the existing systems is that although they recommend examples to students, this recommendation is more often independent of any task that they are assigned. Such a list is not focused and may cause cognitive overload. Knowledge Organization (KO) is a process that enables ITS to present its resources (such as worked-out examples) in an organized way so that it helps students use these resources effectively. NavEx (Yudelson & Brusilovsky, 2005) organizes all its examples into the lessons to which they belong. JavaParser (Hosseini & Brusilovsky, 2013, 2014) attempts to develop an automatic indexing tool for Java problems, indexed on concepts, but still uses syntax trees to store the knowledge. PADS does not present its examples in any order. The proposed system attempts to organize all worked-out examples into coherent groups based on their learning units (instead of lessons). This type of organization can be very helpful to students, especially at the time of final exam preparation. For example, in existing systems such as NavEx (Yudelson & Brusilovsky, 2005), if a student $s1$ wants to study examples on lesson 5 ("for-loops"), then $s1$ must navigate to lesson 5. But there may be examples on "for-loops" in other lessons (e.g. in lesson 8 on functions), which $s1$ will not see unless he/she navigates to lesson 8. This thesis proposes a clustering method that will allow all worked-out examples with "for-loops" to be in a single cluster.

2.2.3 Example-based ITS for non-programming domain

Muldner and Conati (2007) proposed a method called EA-Coach (example-analogy coach) for selecting examples that tailor the needs of individual students based on an example-based learning method known as analogical problem solving (APS). The authors claim that their approach encourages meta-cognitive skills such as self-management and understanding and doing the task instead of just copying it from the suggested examples. EA-Coach takes 3 inputs : a student model represented as a Bayesian Network (BN) (Muldner & Conati, 2007) describing attributes such as student's domain knowledge and any indication of good meta-cognitive skills, solution of the current problem/task stored as a sequence of steps and solutions to each example also stored as a sequence of steps. Solution of problem/task and examples are stored as a sequence of steps, each step derived using a rule. To retrieve the most appropriate example, solutions of the task and examples are compared stepwise. Two corresponding solution steps (of the task and the example) are defined to be structurally different, if they are not generated by the same rule and identical otherwise. Two identical steps are defined to be superficially different in 2 ways : trivial and non-trivial. They are considered identical and superficially different in a trivial way, if by replacing a constant value in example step by a constant value of a problem step generates a correct solution step of the problem. In simple words, if the student can just copy a value from the example and replace it in the problem to get its solution, it is called trivial. If it doesn't allow a simple copy-and-replace, then it is non-trivial. For example, Pstep1 and Estep1 in figure 2.4 are trivially superficially different - different because the constants used in PStep1 and EStep1 are different; trivial because PStep1 can be transformed to EStep1 just by a copy of the constant values from PStep1 to EStep1. A BN is a directed acyclic graph (DAG) of random variables (such as concepts, e.g., Add 2 fractions) that uses Bayes theorem (Pang-Ning et al., 2005) to depict the relationships (probabilities) between these variables. Bayes theorem states that conditional probability P(H|X) (defined as the probability that event H occurs, given X) of a variable H on X can be computed as P(H|X) = P(X|H)*P(H) / P(X), where H is the hypothesis and X is the data sample. P(X|H), P(H) and P(X) are estimated using training data. BNs used in student models typically have their DAGs designed by experts and the probabilistic relationships between variables are estimated by using some training data. In EA-Coach, BN representing the student model consists of nodes for each rule, where each node stores the probability of the student's knowledge of that rule. It also has nodes that indicate the student's attitude towards a problem (e.g., whether the student likes to copy-and-replace from an example to reach the task's solution) and a similarity value between corresponding rules of the task and each example. Based on these values, it simulates the student's behavior. Each node in a BN has a table called conditional probability table (CPT) that stores the probabilities of that node, conditioned on its parent nodes (also called ancestor nodes). For example, at time t, BN has 2 nodes for rule 1 and rule 2, a fact node to represent x=y+z with the fact node having the 2 rule nodes as its parents. If the probability of node for rule 1 is 0.8 and for rule 2 is 0.5, then the probability that the student knows the fact x = y + z if he knows both the rules (P(Fact | Rule1, Rule2) = 0.95) is 0.95 as opposed to 0.5 if he knows only rule 2 (P(Fact | ~Rule1, Rule2) = 0.5). In fact, the fact node's CPT will have 2 other probabilities P(Fact | Rule1, ~Rule2) and P(Fact | ~Rule1, ~Rule2). The probability that a rule is

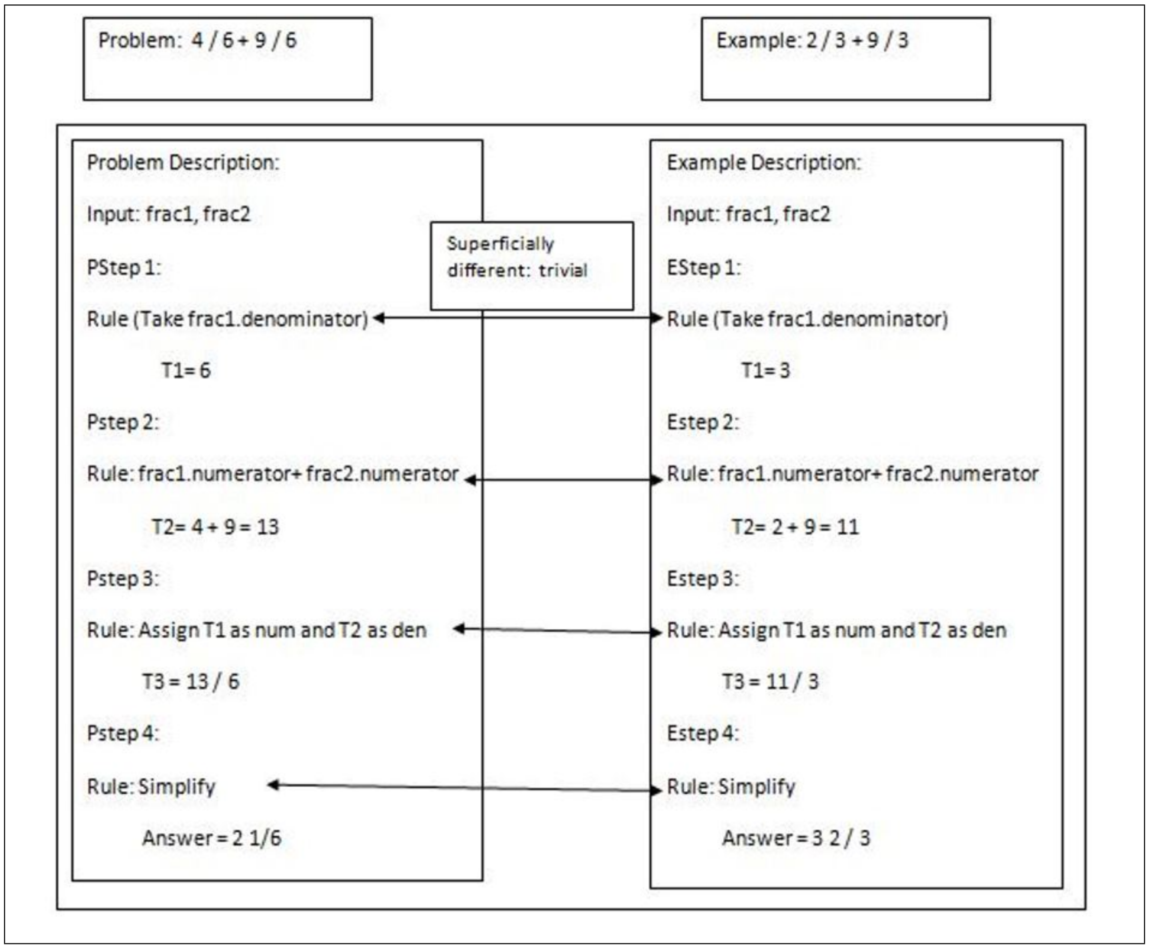


Figure 2.4: Relation between corresponding steps of a problem/task and an example as proposed by Muldner and Conati (2007)

known to the student is called its utility value. At the end, the utility values of each rule are summed up (assuming that each rule has the same weight). The example with the highest utility value is then picked for the student.

Limitations

Although the strength of EA_Coach (Muldner & Conati, 2007) is that examples are selected adaptive to the student's knowledge and his/her meta-cognitive skills, but the domain model that consists of each solution (problem and example) represented as a sequence of steps or rules is done manually and requires a tremendous time and effort of highly trained experts.

2.3 General evaluation methodology for any ITS

An Intelligent Tutoring System (ITS) is a blend of education and technology and therefore, belongs to a category different from Education alone or Technology alone (Woolf, 2010). In general, evaluating an ITS consists of 3 steps (Shute & Regian, 1993):

1. Establish goals of the tutor: Every tutor is built upon a learning theory (such as Example-based learning (EBL)) to achieve its prime goal, which is to accomplish the learning outcomes of the domain it teaches. For example, the prime goal of ANDES (Schulze et al., 2000) is to teach introductory Physics to Undergraduate courses and the primary goal of ERS is to teach C programming as an introductory course to Undergraduate students (Chaturvedi & Ezeife, 2015b).

2. Identify goals of evaluation: Shute and Regian (1993) argue that although the prime goal of an ITS, like any other tutor, is to focus on improving the learning outcomes for the domain they teach, they may have alternate goals as well. For example, an alternate goal of ANDES (Schulze et al., 2000) is to provide relevant hints to students and that of ERS (Chaturvedi & Ezeife, 2015b) is to increase the likelihood of student success in graded tasks, where success is measured in terms of higher marks.

3. Build an evaluation methodology : An evaluation methodology dictates how the the prime and alternate goals of the ITS are measured. Building such a methodology relies on answers to 3 basic questions:

    (a) **What is evaluated?** This questions refers to the prime and alternate goals of the tutor identified in steps 1 and 2. In general, an ITS is evaluated in 3 different ways:

        i. as a tutor or educator that measures the educational impact of the ITS (Shute & Regian, 1993; Woolf, 2010).

        ii. in terms of its individual components and their algorithms (e.g. evaluate modules KE, KC and KO of ERS (Chaturvedi & Ezeife, 2015b)).

iii. in terms of different features used in the ITS (Mark & Greer, 1993; Greer & Mark, 2016).

(b) **What is the type of evaluation methodology (Formative / Summative)?** Formative evaluation is a method in which a group of users test the system while it is still in the developing stage, so that they can assist in the process of development, if necessary. This is not a very common method to evaluate ITS because of their complex and dynamic nature (Mark & Greer, 1993; Greer & Mark, 2016). However, it can be applied to evaluate its individual components (e.g. ERS's experts use it to validate its knowledge extraction module). Summative evaluation assesses whether the end product of the ITS accomplishes its goals (e.g. do students using ERS tend to score higher marks in the assigned tasks). It evaluates the complete system and attempts to prove the formal claims made about the system (Mark & Greer, 1993). It provides objectivity and makes it possible to measure factors such as the tutor's educational impact. Experimental Design, a commonly used method of evaluation in areas such as Psychology, facilitates an ITS's summative evaluation. It starts by forming a research question (R) and making a hypothesis based on that. Once a hypothesis (H) is made, research is designed (D) to examine H. Then the study is completed using D and its data is analyzed to prove that results conform to H. Most of the existing ITS (e.g. (Arroyo et al., 2003; Yudelson & Brusilovsky, 2005; Li & Chen, 2009)) use this method for a summative evaluation of their goals. ERS is evaluated as a tutor in chapter 6.

(c) **How is an ITS evaluated?** Is the tutor evaluated as standalone or against other existing tutors? Woolf (Woolf, 2010) summarized six different ways in which ITS tutors are evaluated:

i. C1: Tutor alone: In this method, one group of students use the ITS and a test at the end determines if students meet the ITS's learning outcomes. A disadvantage of this method is that its learning outcomes cannot be compared to any other system and therefore it is difficult to identify the

ITS features that make learning effective. PADS (Li & Chen, 2009) uses C1 to meet its goals.

ii. C2: Tutor verses non-interventional control: Students, in this method, are divided into 2 groups. One group of students uses the ITS, whereas the other group receives no teaching at all. This method is useful only if the ITS is designed to prove whether a web-based electronic tutoring system is better than students learning on their own. This method is clearly biased towards group 1 students that use the ITS. It is also difficult to identify the ITS features that could lead to better learning.

iii. C3: Tutor verses traditional classroom teaching: In C3, students are divided into 2 groups. One group of students uses the ITS, whereas the other group receives classroom teaching. Similar to C2, this method is useful only if the ITS is designed to prove whether a web-based electronic tutoring system is better than human tutors. This method also has disadvantages similar to C2.

iv. C4: Tutor_1verses Tutor_2: C4 divides students into 2 groups. Group 1 of students works with version 1 of the ITS (Tutor_1), whereas group 2 works with a different version of the same tutor (Tutor_2). This method is very effective but it is very time-consuming and resource-intensive to build more than versions of the same tutor and test them. An example of C4 is WebEx as Tutor_1 (Brusilovsky, 2001) and NavEx as Tutor_2(Yudelson & Brusilovsky, 2005). NavEx was developed by the same group of researchers to improve upon the limitations of WebEx.

v. C5: Tutor verses ablated tutor: Once again, students are divided into 2 groups. Group 1 of students works with a complete version of the ITS, whereas group 2 works with the same tutor minus one or more features of that ITS (e.g. group 1 is allowed to visit all worked-out examples in ERS's database, whereas group 2 is allowed to browse only selected examples that ERS recommends). According to Woolf (2010), ablation experiments compare learning performances to identify those features that improve learning by removing certain design features of the ITS. C5 is very effective but it does suffer from the risk of being biased towards students in group 1.

vi. C6: Tutor A verses Tutor B: In C6, students are divided into 2 groups. Both groups work with entirely different ITS that are developed by different teams and may even be built on different learning theories. Such a methodology may be useful when the goal is to measure and compare ITS design and how students interact with them. A disadvantage of such a method is that it is difficult for a research team to find 2 tutors that cover exactly the same topics and are built for the same learning outcomes. Hosseini and Brusilovsky (2013) use this method of evaluation to compare 2 tutors that teach Java Programming, Knowledge Maximizer (Hosseini et al., 2013) (Tutor A) and Quizguide (Tutor B).

Table 2.7 illustrates these stages for some ITS systems based on EBL learning theory. The most challenging aspect of evaluating the goals of an ITS is the size of the student model component. In some situations, the class sizes are small, whereas in others, students are reluctant to participate in a research study. As the table shows, (summative) evaluation in both WebEx (Brusilovsky, 2001) and NavEx (Yudelson & Brusilovsky, 2005) was solely on the usage of their resources. WebEx was offered to students registered in an introductory programming course for three semesters in their study (spring 2002, fall 2002 and spring 2003) but only 18 students filled the questionnaire form that was required for WebEx's subjective evaluation. NavEx (Yudelson & Brusilovsky, 2005) used 34 active students in their study - 23 students from Spring 2004 term and 11 from Fall 2004 term. PADS (Li & Chen, 2009) included 50 first-year students and measured the total number of tasks students can complete in a given period, if the tasks are from their zone of proximal development (Vygotsky, 1987). KM (Hosseini & Brusilovsky, 2013) used 14 students in their study from one term of an undergraduate Java Programming course.

	WebEx Brusilovsky (2001)	NavEx (Yudelson & Brusilovsky, 2005)	PADS (Li & Chen, 2009)	KM Hosseini & Brusilovsky (2013)
Goal(s) of Tutor	To motivate students to use an online system so that they can optimize the use of comments in C programs	To improve the quality of adaptive navigation support for the system developed for WebEx	To allow students to spend less time on given tasks so that they can do more tasks in a given time period	To support Java Programming students in preparing for final exam
Goal(s) of evaluation	To improve frequency of use of examples and improved course coverage	To improve the adaptive support of WebEx to improve frequency of use of examples and improved course coverage even further	To present students with tasks of appropriate difficulty level (using zone of proximal development)	To provide students with a sequence of questions that help him/her to fill in gaps in java knowledge towards final exam preparation
Evaluation Style Formative/ Summative	Summative	Summative	Summative	Summative
Evaluation Methodology	C1 (Tutor alone)	C4 (Tutor$_1$vs Tutor$_2$)	C1 (Tutor alone)	C6 (Tutor A verses Tutor B)
Number of students used in the study	18	34	50	14

Table 2.7: Evaluation goals and methodology used in ERS and other EBL-based ITS

2.4 Chapter 2 Overview

Chapter 2 presents the data mining techniques used commonly in ITS (e.g. Clustering, Classification, Association Rule Mining). It then presents a taxonomy of existing ITS that are built on EBL (Example Based Learning) strategy. The techniques used in such existing systems (both in the programming and non-programming domain) and their limitations are described. It also presents the general evaluation methodologies used in ITS and briefly highlights the methods used to evaluate the proposed system.

Chapter 3

Proposed Architecture for an EBL-based ITS

An Intelligent Tutoring System (ITS) provides direct customized instruction or feedback to students when they perform a task in a tutoring system without the intervention of a human. As described earlier in chapter 1, one of the main functions of an ITS system is to present its students with course materials that are most useful to them in terms of their own knowledge on the domain and in terms of other resources in the ITS such as tasks assigned to them. ITS typically compare and analyze student model (SM) components to estimate student's current knowledge or mastery on the basic topics of the domain (e.g. scanf is a topic in the domain of C programming) and use such estimates to recommend resources that the ITS thinks will help students to succeed in learning the domain. This thesis proposes a framework called Example Recommendation System (ERS) that is built upon EBL and that uses state-of-the-art mining methods in order to recommend a focused, organized and customized list of worked-out examples with the overall objective of increasing the likelihood of student success in the ITS's domain. Example-based learning (EBL) is a well-known teaching strategy in traditional educational systems (VanLehn, 1998; Gog & Rummer, 2010; Renkl, 2014) and worked-out examples are an important and an almost an inherent part of this teaching methodology. Although research in ITS has gone a long way in simulating the role of a teacher in many ways, not much progress has been made on designing ITS based on EBL methodology. Existing systems such as NavEx (Yudelson & Brusilovsky, 2005), PADS (Li & Chen, 2009) and those used by Hosseini and Brusilovsky (2013) use EBL in their ITS but they have several limitations as described in sections 2.2 and 2.3 and also listed in section 3.2. This thesis proposes to alleviate the limitations that exist in many key components involved in the design of an EBL-based ITS.

3.1 Definitions

Key definitions of the basic entities that play a major role in this research are listed here.

Definition 1: Domain of an ITS refers to the subject or course that the ITS is designed to teach (mostly within a limited scope) (e.g. domain of C programming).

Definition 2: A task in an ITS is defined to be a gradable question or instruction assigned to students (e.g. task T1: 'There are 2.54 centimeters to 1 inch. Write a C program that asks a user to enter the value of his/her height in inches and then displays the height in centimeters.').

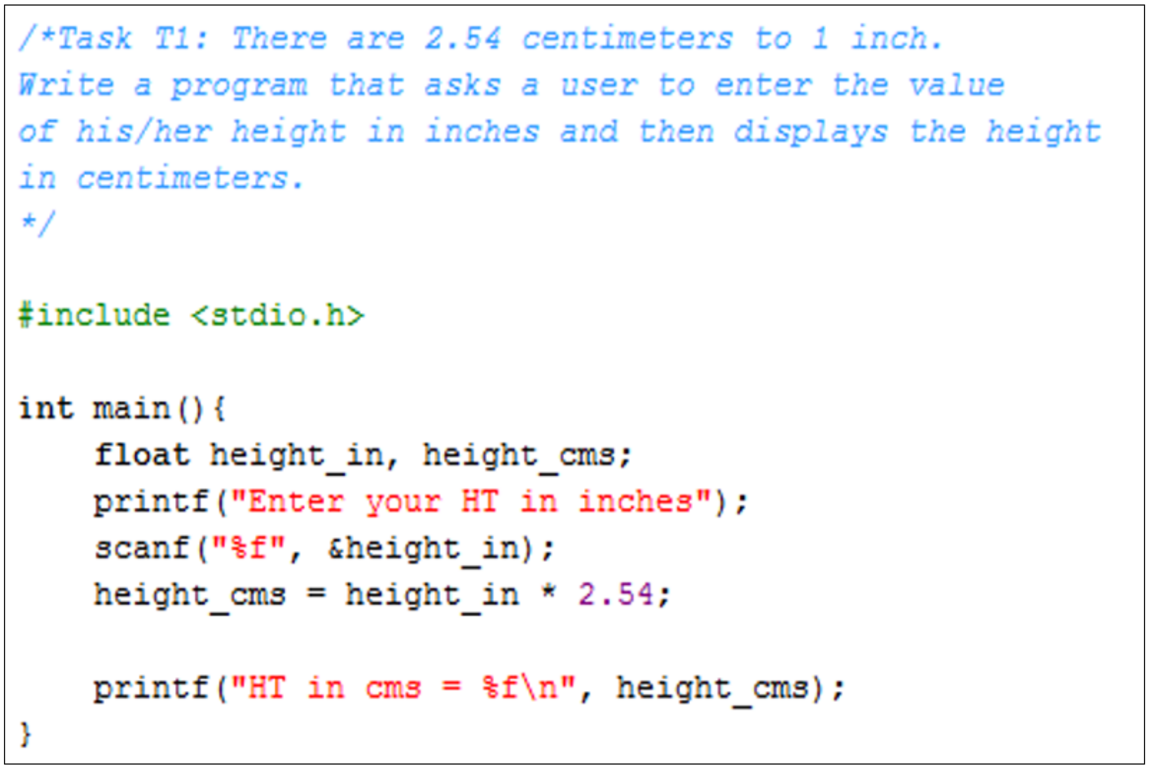
Every course that is offered for credits (be it in traditional classroom teaching or online teaching environment) requires some gradable instruments such as tasks, assignments, quizzes and tests, so that student performance on the domain can be measured objectively. ERS (proposed ITS) uses marks scored by students in tasks to measure their performance in the course. Tasks and their marking schemes are provided by domain experts.

Definition 3: A task solution is the correct solution of the question or instruction asked in a task (e.g. task solution for T1 is shown in figure 3.1).

Task solutions are provided by domain experts.

Definition 4: A worked-out example (WE) refers to a complete or partial worked-out solution of a question or instruction (similar to examples in textbooks).

Worked-out examples are provided by domain experts. The proposed method uses the same structure for task solutions and worked-out examples. Figure 1.1 in chapter 1 shows



```
/*Task T1: There are 2.54 centimeters to 1 inch.
Write a program that asks a user to enter the value
of his/her height in inches and then displays the height
in centimeters.
*/

#include <stdio.h>

int main(){
    float height_in, height_cms;
    printf("Enter your HT in inches");
    scanf("%f", &height_in);
    height_cms = height_in * 2.54;

    printf("HT in cms = %f\n", height_cms);
}
```

Figure 3.1: Sample Task Solution of task T1: 'Write a program that asks a user to enter the value of his/her height in inches and then displays the height in centimeters'.

a worked-out example of a C program that computes and prints the area of a triangle, given its base and height.

Definition 5: A learning unit (LU), also referred to as a topic or concept, is the smallest basic unit of knowledge in that domain that a worked-out example or a task solution is divided into. For example, "scanf" is an LU in the domain of C programming. Similarly, "fraction" is an LU in the domain of Math.

Domain experts define all learning units that an ITS must have, depending on the scope of the domain. For example, if the domain is Math and its limited scope is to teach addition of 2 fractions, then the learning units required for this scope are addition, multiplication, division, numerator of a fraction, denominator of a fraction, least common multiples of 2 whole numbers and reducing fractions.

3.2 Problems identified in existing ITS based on Example-based learning

This thesis identifies several problems in existing ITS systems that are designed using example-based learning method (discussed in chapter 2).

1. Domain Knowledge Acquisition and Extraction: Existing systems (chapter 2 section 2.2) extract knowledge (in terms of LUs) from tasks and worked-out examples using techniques that are either manual, or automatic but so domain-specific that they cannot be easily adapted to other domains and require highly trained experts. Many existing ITS (Yudelson & Brusilovsky, 2005; Hosseini et al., 2013; Mokbel et al., 2013; Hosseini & Brusilovsky, 2014) require experts to write grammar rules and create syntax trees for resources such as worked-out examples. Others require experts to manually list the learning units that each resource consists of (Li & Chen, 2009). These methods not only limit the capabilities of domain knowledge extraction as extendable in scope of the current domain or to newer domains, they also require significant effort by experts.

2. Task-independence: Existing systems surveyed in section 2.2 recommend worked-out examples independent of any task assigned. The onus lies on students to fetch examples that will help them succeed in a task or test.

3. Customization methods: Existing systems such as NavEx (Yudelson & Brusilovsky, 2005) lack in use of state-of-the-art mining methods to search for relevant worked-out examples. Others such as PADS (Li & Chen, 2009) build models based on features that are very domain-specific and subjective. Such limitations are a direct consequence of lack of a robust pre-processing module that represents student and domain data in a way that enables state-of-the-art mining methods.

4. Organization style and methods: Some existing systems such as PADS (Li & Chen, 2009) do not present their worked-out examples in any organized way. In other systems, such as NavEx (Yudelson & Brusilovsky, 2005), experts organize their

examples in the traditional way found in textbooks (lesson-wise). Hosseini and Brusilovsky (Hosseini & Brusilovsky, 2013) propose methods to index their worked-out examples but do not organize them as groups. Organizing worked-out examples using automated methods into groups such that all those with related LUs are placed in one group can be very useful to students when they prepare for exams (e.g. final examination) and for domain experts in managing their resources.

5. Lack of modularity: None of the existing systems surveyed in section 2.2 present a modular framework to accommodate the various functionalities of a complex ITS system. This makes them less extendable to other domains.

6. Incomplete evaluation methods: An ITS is typically a complex system that consists of domain and student model data and various components such as KE, KC and KO. Therefore, it must be evaluated not only on its prime role as a tutor, but also on its alternate goals such as evaluation of its individual components and features. Existing EBL-based ITS are evaluated only on their educational impact using data that is subjective and usage-driven (e.g. how many times a student clicks on a worked-out example).

3.3 Research Questions

In light of the problems identified in section 3.2, we put forth the following research questions to be answered by this dissertation:

Research Question RQ1 Is it possible to design knowledge extraction methods that are (1) simple, (2) efficient, (3) can extract LUs from worked-out examples and task solutions correctly and (4) are extendable to new domains without the need of highly trained experts?

Answer: Yes, it is possible. An ITS that supports example-based learning (EBL) does not require to consider and verify syntactic relationships between the extracted LUs of task solutions or worked-out examples in order to perform its functions - it just needs to identify the existence of LUs in them. Therefore, following the principle of Occam's razor

(Domingos, 1999), which states that if there are two models performing the same function, the simpler one should be preferred, we propose to use regular expression analysis to extract LUs, instead of complex methods such as syntax trees (as used in many existing systems (Yudelson & Brusilovsky, 2005; Mokbel et al., 2013; Hosseini & Brusilovsky, 2014)). Chapter 5 section 5.2 validates our proposed algorithm's correctness and extendibility.

Research Question RQ2 What impact does a focused and concise list of worked-out examples have on student performance and learning?

Answer: This question has 2 parts: (1) Can data mining methods be applied on ITS data to generate list of worked-out examples focused towards an assigned task? (2) Does a focused list improve student learning?. Chapter 5 section 5.3 validates our proposed algorithm that builds a highly accurate mining model to generate such a focused list of examples for each assigned task. Chapter 6 compares student performance in different scenarios and demonstrates that it does improve student learning.

Research Question RQ3 Can inclusion of global relevance of LUs in a set of worked-out examples impact cluster formation?

Answer: Yes it can. We propose to organize all worked-out examples in the ITS in order to assist students with final exam preparation. For binary data, standard k-means algorithm uses mode to compute and recompute its centroids in each iteration, and therefore clusters formed are biased towards those LUs that are present in very few worked-out examples. Including the global relevance of such LUs while computing cluster centroids forms better clusters that are well-separated and compact. Chapter 5 section 5.4 answers this question.

Research Question RQ4 Is there a way to clearly define and integrate different modules of an ITS and if so, how?

Answer: Yes, there is. To answer this question, we present an architecture for the proposed ITS called ERS in chapter 4 in which knowledge extracted from the domain model is used to represent each worked-out example and task solution in vector space and is then integrated into the knowledge customization (KC) and knowledge organization (KO) modules. Chapter 5 presents the experiments done for each module of ERS.

Research Question RQ5 How does selection of features from domain and student models of an ITS impact the accuracy of predicting student performance?

Answer: We claim that designing a predictive model using features that are better informed about the assigned tasks and its components improves its accuracy (e.g. features such as difficulty level of the task, student's average marks on LUs of a task). Chapter 7 section 7 validates our claim.

3.4 Thesis statement

Given (i) a domain model that consists of:

- set L of basic LUs taught by the ITS
- set W of worked-out examples constructed using one or more of the LUs defined by the expert
- set T tasks created with the intent to measure student performance in the domain objectively
- set TS that consists of solutions of all tasks in T

and (ii) a student model describing for each student s

- s's static information such as ID and name
- s's knowledge on the LUs in L
- s's learning behavior captured by s's interaction with ERS

the problem of motivating students to study recommended course materials (such as worked-out examples) towards meeting their short-term goal of succeeding in the current task and their long-term goal of learning the domain can be formulated as an integration of three independent but cohesive components:

1. Knowledge Extraction (KE) : is defined as the process of extracting LUs from worked-out examples and task solutions. This requires the domain experts to define the LUs in the domain, create all resources for the domain model such as task solutions and worked-out examples and describe effective, efficient and extendable algorithms to extract LUs from them. The aim of KE in this study is to design methods that can be easily extended to other domains without the need of highly trained experts.

2. Knowledge Customization (KC) : is defined as the process of generating a concise list of worked-out examples customized towards the tasks assigned to students by ERS and to their current knowledge on the domain.

3. Knowledge Organization (KO) : is defined as the process of organizing all worked-out examples into coherent groups based on the LUs they contain. Such a list can be very helpful to students at the time of final exam preparation.

We achieve this integration of components KE, KC and KO by designing a framework called Example Recommendation System (ERS), whose architecture is described next.

3.5 Basic Architecture of ERS

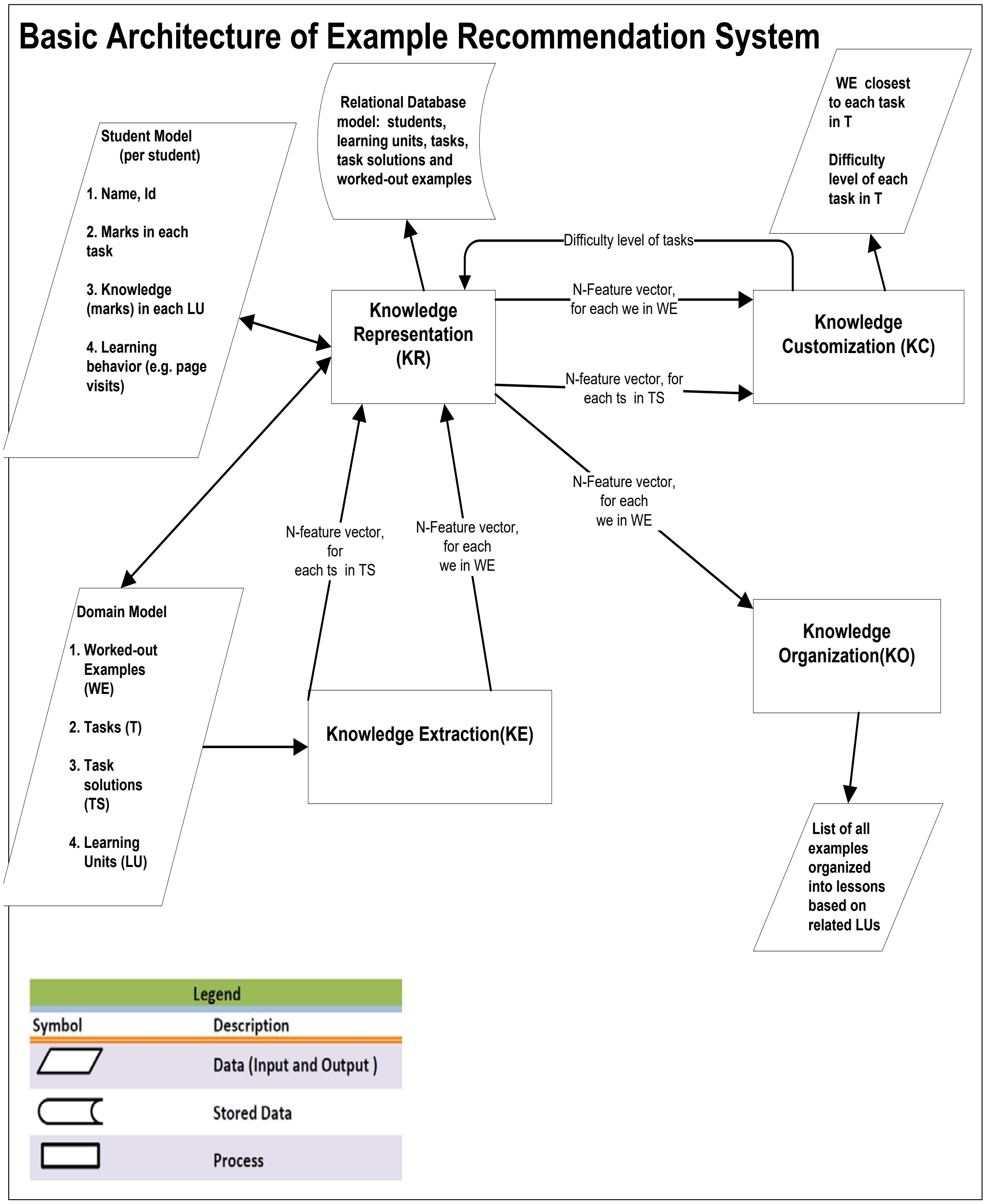
The proposed ERS framework defines four main modules as shown in figure 3.2. Knowledge extraction (KE) module mainly interacts with domain experts to automatically identify all learning units (LU) that the task solutions and worked-out examples in the domain model contain and represent them as n-feature vectors, where n is the total number of LUs in ERS. These are then conveniently stored and represented in a relational database by the knowledge representation (KR) module. Anytime a new task or worked-out example is added to ERS's domain model, it goes through the knowledge extraction module and and is transformed into an n-feature vector. Similarly, each student's information is also stored in the relational database. Knowledge customization (KC) uses the feature vectors of task solutions and worked-out examples resulting from KE to build a data mining model that generates the list of worked-out examples that are closest to the assigned task. Knowledge organization (KO) also uses these vectors to cluster all worked-out examples that are closest in terms of the LUs they contain (instead of the lessons they belong to). Chapter 4 describes each module in detail.

3.5.1 Methodologies for evaluating ERS

ERS is a complex system that consists of several modules with different functionalities and diverse datasets emerging from its student and domain models. We perform a rigorous evaluation of ERS using techniques described in chapter 2 section 2.3. ERS is evaluated using a combination of formative and summative evaluation methods and a modified C5 (tutor verses ablated tutor) to evaluate its prime goal of improving student learning and alternate goals of building state-of-the-art models for its individual components using effective student and domain model features. It is evaluated in three distinct ways: (1) EM1: evaluate its individual components and their algorithms (discussed in chapter 5) (2) EM2: evaluate it as a tutor (discussed in chapter 6) and (3) EM3: evaluate its student and domain model features (discussed in chapter 7).

3.6 Motivation for choosing the domain

The proposed ERS system is designed and validated for Programming in C domain. KE component of ERS is also validated on a second domain (Programming in Miranda). University of Windsor offers an online version of a course on Programming in C for first year undergraduate students. It also offers an undergraduate course on Programming in Miranda. Although we have the best teachers to teach these courses, it becomes impossible for them to provide one-on-one assistance to students, either because of large class sizes or due to time conflicts between student and teacher schedules. University of Windsor provides a robust learning management system (LMS) to host course materials for each course offered such as lecture slides, examples and assignments. The LMS used until Fall 2015 was CLEW (Collaborative and Learning Environment Windsor) (Windsor, 2014b). Although CLEW has been replaced by Blackboard (Windsor, 2014a) from Winter 2015, this thesis uses CLEW for its research. CLEW is used to support teaching and



Basic Architecture of Example Recommendation System
Relational Database model: students, learning units, tasks, task solutions and worked-out examples
WE closest to each task in T
Difficulty level of each task in T
Student Model (per student)
1. Name, Id
2. Marks in each task
3. Knowledge (marks) in each LU
4. Learning behavior (e.g. page visits)
Difficulty level of tasks
N-Feature vector, for each we in WE
Knowledge Representation (KR)
Knowledge Customization (KC)
N-feature vector, for each ts in TS
N-Feature vector, for each we in WE
N-feature vector, for each ts in TS
N-Feature vector, for each we in WE
Domain Model
1. Worked-out Examples (WE)
2. Tasks (T)
3. Task solutions (TS)
4. Learning Units (LU)
Knowledge Extraction(KE)
Knowledge Organization(KO)
List of all examples organized into lessons based on related LUs
Legend
Symbol
Description
Data (Input and Output)
Stored Data
Process

Figure 3.2: Architecture of the proposed ERS system

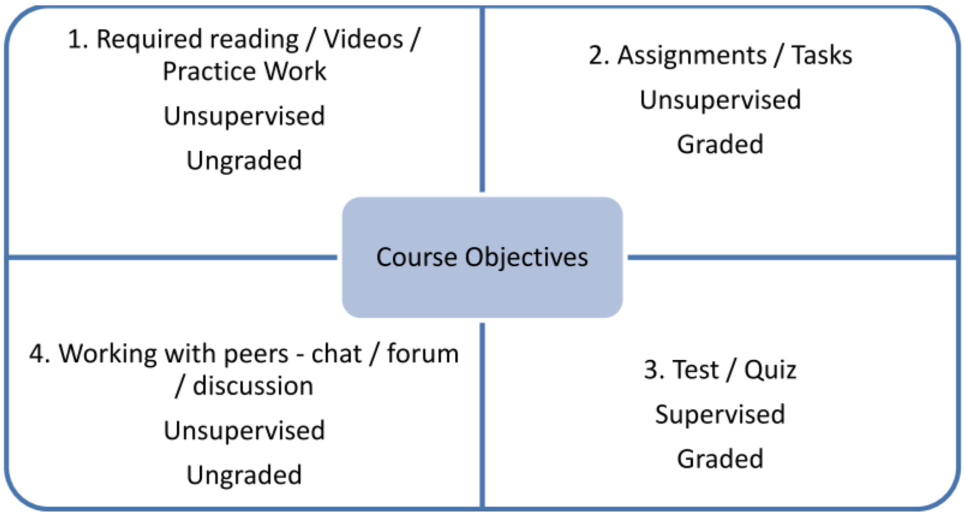


Figure 3.3: Course Structure on CLEW

learning in face-to-face, distance education and blended courses (www.uwindsor.ca/clew). A typical course structure used on clew is shown in figure 3.3. Each course has a set of objectives, required reading assignment and a set of assessment instruments - both supervised (e.g. graded tests) and unsupervised (e.g. assignments). Students use the discussion board and chat rooms to collaborate with their peers or post concerns related to the course. Students registered in Winter 2015 were offered to use ERS, whereas students registered in Fall 2015 were required to use ERS, in addition to using clew and its resources.

3.7 Scope of ERS domain

ERS uses 2 domains for its experiments - domain D1 is on programming in C and domain D2 is on Programming in Miranda. Domain experts of ERS define its scope in terms of learning units (LUs) that ERS teaches. Each worked-out example and task solution in ERS is represented in terms of these LUs and the hierarchy they follow is:

Domain $consistsOf$ Task solutions $consistsOf$ Learning Units

Domain $consistsOf$ Worked Out Examples $consistsOf$ Learning Units

LU1	LU2	LU3	LU4	LU5	LU6	LU7	LU8
datatype	variable	assignment	arithmetic expression simple	arithmetic expression compound	printf - constant messages only	printf - variables only	printf - mixed

(a) Learning units LU1 - LU8

LU10	LU11	LU12	LU13	LU14	LU15	LU17	LU18
scanf - single input	scanf - multiple input	relational expression	logical expression	mixed expression relational logical	mixed expression (all types)	compound statement	while loop simple

(b) Learning Units LU10 - LU18

LU19	LU20	LU22	LU23	LU24	LU25	LU26	LU27
for loop simple	nested loops	if /else statement	nested if / else	switch	function prototype	function definition - call by value	arrays

(c) Learning Units LU19 - LU26

Table 3.1: Scope of ERS's Domain D1 (Programming in C)

3.7.1 Domain D1: Programming in C

Table 3.1 lists the LUs in the scope of ERS's domain D1.

3.7.1.1 How are domain model's resources built and refreshed?

Source 1: Textbook written by Dr. C. I. Ezeife (2010).

Source 2: Resources developed for a course on C Programming offered by University of Windsor to first-year Undergraduate non-Computer Science majors (Windsor, 2014b).

Dataset for domain D1 is further described in section 5.1.

3.7.2 Domain D2: Programming in Miranda

Table 3.2 lists the learning units in the scope of ERS's domain D2. Learning Units for domain D2 start at LU41 and end at LU56.

LU41	LU42	LU43	LU44	LU45	LU46	LU47	LU48
Arithmetic Operator	Foldr	Map	Empty List	Function Composition	If Statement	List Comprehension	List - Many Elements

(a) Learning units LU41 - LU48

LU49	LU50	LU51	LU52	LU53	LU54	LU55	LU56
List One Element	List Operator	Primitive Type	Recursion	Relational Operator	Tuple	String	Logical Operator

(b) Learning units LU49 - LU56

Table 3.2: Scope of ERS's domain D2 (Programming in Miranda)

3.7.2.1 How are domain model's resources built and refreshed?

Source: Resources developed for a course on Key concepts in Computer Science (includes Miranda Programming) offered by University of Windsor to first-year Undergraduate students with Computer Science major (Frost, 2015).

Dataset for domain D2 is further described in section 5.1.

3.8 Chapter 3 Overview

Chapter 3 begins by defining all elements such as tasks, task solutions, worked-out examples and learning units that play a key role in the framework of the proposed EBS_based ITS called ERS. The main elements that connect ERS with students are tasks and worked-out examples. This chapter then presents the research questions and defines the thesis statement of this dissertation. It also presents the basic framework and scope of ERS.

Chapter 4

Example Recommendation System (ERS)

This chapter describes each module of the ERS framework along with the techniques and algorithms used in them. Sections 4.1, 4.2, 4.3 and 4.4 present the core algorithms and methods used in knowledge extraction from ITS resources, their representation, customization of resources to student needs and knowledge organization. Section 4.5 demonstrates an example that clearly integrates the 3 modules to accomplish the objectives of ERS.

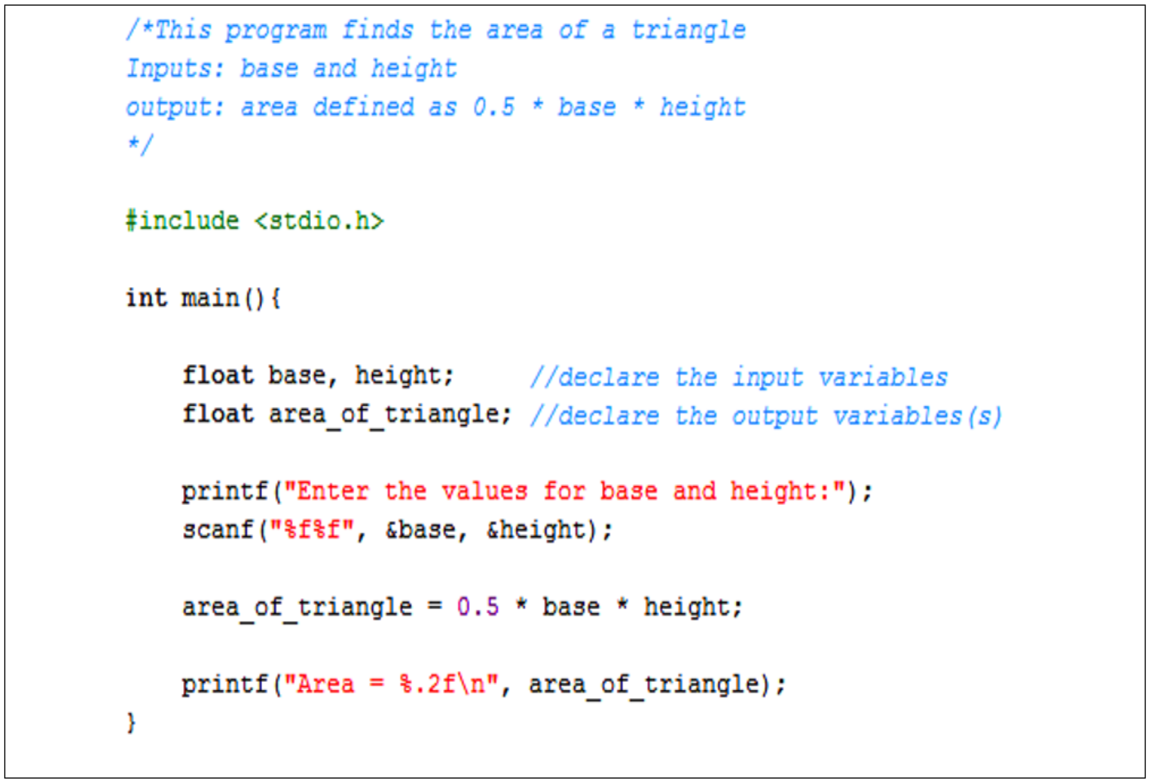
4.1 Knowledge Extraction in ERS

The goal of knowledge extraction (KE) module of ERS is to automatically transform each task solution and worked-out example in its domain to a binary vector of size n, where n = number of LUs in ERS. The main motivation to design and implement new KE algorithms for EBL-based ITS such as ERS is the lack of its extendibility in the existing systems to other domains. This section explains the novel algorithms proposed by ERS for KE that are simple, efficient and easily extendable to other domains.

4.1.1 Domain Model

A domain model of an ITS defined by experts in the domain stores the correct knowledge of what is taught within the scope of the ITS. Domain model of ERS is built by a pool of Computer Science instructors and students (graduate and undergraduate majors) and consists of the following items:

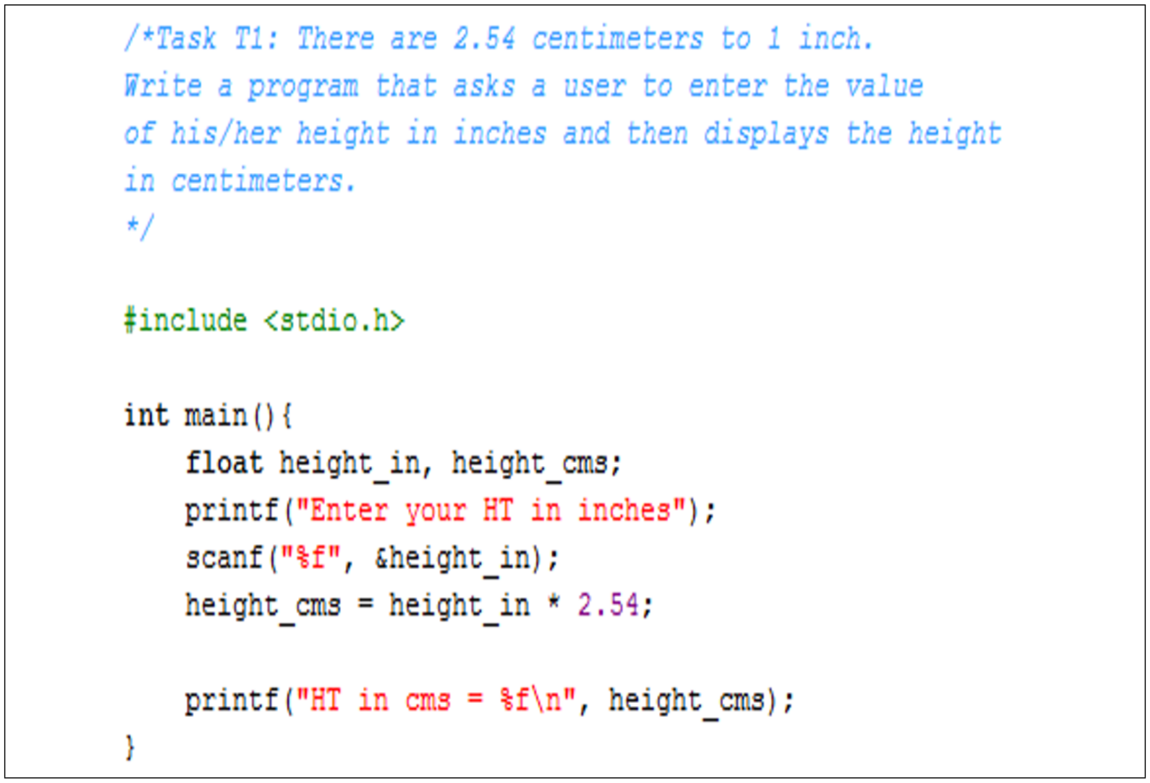
1. Worked-out examples: Figure 1.1 in chapter 1 shows an example solution (repeated below for convenience). Table 4.3a shows several other worked-out examples.



```
/*This program finds the area of a triangle
Inputs: base and height
output: area defined as 0.5 * base * height
*/

#include <stdio.h>

int main(){

    float base, height;      //declare the input variables
    float area_of_triangle; //declare the output variables(s)

    printf("Enter the values for base and height:");
    scanf("%f%f", &base, &height);

    area_of_triangle = 0.5 * base * height;

    printf("Area = %.2f\n", area_of_triangle);
}
```

2. Tasks assigned to students: An important function of any tutoring system is the ability to measure its students performance objectively. ERS asks its students to perform several tasks to assess them objectively on the learning units taught by ERS. For example, Task T1, as given in chapter 3, is 'T1: There are 2.54 centimeters to 1 inch. Write a C program that asks a user to enter the value of his/her height in inches and then displays the height in centimeters'.

3. Task solutions: domain experts provide solutions to every task that is in ERS's database. Task solutions share the same structure as a worked-out example. Figure 3.1 shows a solution for task T1 and is repeated here for convenience.



```
/*Task T1: There are 2.54 centimeters to 1 inch.
Write a program that asks a user to enter the value
of his/her height in inches and then displays the height
in centimeters.
*/

#include <stdio.h>

int main(){
    float height_in, height_cms;
    printf("Enter your HT in inches");
    scanf("%f", &height_in);
    height_cms = height_in * 2.54;

    printf("HT in cms = %f\n", height_cms);
}
```

4. Learning units (LUs) : defined by experts for the scope of ERS. Experts are required to list all LUs in the ascending order of difficulty such that LU1 represents the simplest LU while LU50 has a higher level of difficulty and LU100 has even higher level of difficulty than LU50. They are also required to partition the complete list of LUs into 2 disjoint sets of simple (S) and complex (C) LUs based on material difficulty. For example, experts on C define the list of LUs in order of difficulty as Datatype < Variable < scanf < BE < for loop < function definition (only six LUs are shown here) and partition them as S = {Datatype, Variable, scanf, BE} C = { for loop, function definition}.

5. Algorithms for extracting LUs from task solutions and worked-out examples: ERS's domain experts provide regular expressions for each LU in its scope and algorithms to extract one or more of these LUs from given task solutions and worked-out

examples. Section 4.1.2 defines regular expressions and the process of using regular expressions for KE module of ERS.

4.1.2 Knowledge Extraction using regular expressions

In order to facilitate the functionality of ERS (such as customization in KC discussed in section 4.3) that compares task solutions and worked-out examples), our system divides each worked-out example and task solution to one or more LUs they require or cover. This is achieved by pattern matching using regular expressions. A regular expression (RE) is defined as a set of characters that describe a pattern (Friedl, 2006). A RE is made up of constants and symbols that have a special meaning and are known as metacharacters (e.g. symbols such as \, ?, *, + and |). Some commonly used metacharacters are explained below with examples.

- +: this symbol matches the preceding character one or more times. For example, a RE $re1$ given as *Joh+n* matches strings that start with Jo followed by 1 or more h, followed by n. Input strings such as *John* and *Johhhhn* will be recognized by $re1$ but not *Jon*.

- *: this symbol matches the preceding character zero or more times. For example, a RE $re2$ given as *Joh*n* matches strings that start with Jo followed by 0 or more h, followed by n. Input strings such as *Jon*, *John* and *Johhhhn* will be recognized by $re2$.

- \: symbol \ is used when it is required to override the meaning of the used metacharacter symbol with its literal meaning. For example, a RE that identifies strings such as 1 + 1 = 2 needs to override the meaning of metacharacter + with the literal meaning of +. RE $re3$ must be written as d \+ d = d (where d is any single digit from 0 .. 9) in order to identify strings such as 1 + 1 = 2 (RE written incorrectly as d + d = d will identify strings such as 1234 = 6 and 1 = 5).

- ?: symbol ? makes the preceding character in the regular expression optional. For example, RE $re4$ given as *Joh?n* matches both strings *John* and *Jon*.

	Regular Expression	Explanation - Regular expression in column 1 matches
1	\s	any whitespace character
2	\s*	0 or more whitespace characters
3	\a	character a literally
4	.	any character except newline
5	.*	any character (except newline) 0 or more times greedily (returns the longest match)
6	.*?	any character (except newline) 0 or more times lazily (returns the shortest match)
7	(a|b)	either a or b
8	[^a]	any character except a
9	\d	digit 0 .. 9

Table 4.1: Few Regular expressions with metacharacters and their explanations

- |: symbol | matches either the preceding character or the proceeding one but not both. For example, RE *re*5 *Joh*|*n* matches *Joh* and *Jon* but not *John*.

Table 4.1 describes a few regular expressions using metacharacters and their meaning. RE *s* matches any whitespace character (newline, blank space or tab), whereas *s** matches 0 or more whitespace characters in an input string. Symbol . matches any character except newline. Similarly, .*? matches any character (except newline) 0 or more times but stops as soon as it finds its first occurrence. For example, if RE *re*6 is given as \"*.*?*\" and is searched for in an input string *s* = "*Value of* "*s*" *is* %*s*", then the resulting match is *Value of*, whereas if *RE re*6 is given as \"*.** \", then the resulting match is *Value of* "*s*" *is* %*s* (since .* continues its search until it finds its last occurrence).

Domain experts of ERS define RE for each LU listed in table 3.1, although only four of them (LU2, LU6, LU7 and LU8) are shown in table 4.2 to illustrate the power of RE. RE for LU2 states that a variable in C starts with a letter and can be followed by zero or more letters and digits. ERS experts identify 3 different learning units for printing in C (LU6, LU7 and LU8 as shown in table 3.1). RE for LU6 is simple - it allows all characters within quotes except a % symbol. This is a limitation of ERS that it imposes on its experts (RE for LU6 does not allow task solutions and worked-out examples to print constant messages that includes a % symbol). For example, *relu*6 will correctly identify statements such as printf("Welcome"); or printf (" wel come "); but will fail

Learning Units	Regular expressions
LU2 (variable)	*relu*2: [a-zA-Z_][a-zA-Z0-9_]*
LU6 (print constant messages only)	*relu*6: printf \s* \(\s* \" [^%]+ \" \s* \) ;
LU7 (print variables only)	*relu*7 : printf \(\" (\%[dcf])+ \" \, [a-zA-Z_][a-zA-Z0-9_]* (\, ([a-zA-Z_][a-zA-Z0-9_]*))+ \) ;
LU8 (print mixed constant messages and variables)	*relu*8: printf(\ |\s)*\((\ |\s)*\".*\"((\ |\s)*,(\ |\s)*([a-zA-Z_][a-zA-Z0-9_]*|[0-9]+))*(\ |\s)*\)(\ |\s)*;

Table 4.2: Regular expressions for an identifier and printf used in ERS

to identify statements such as printf "Welcome" or printf("Wel%come");. RE for LU7 (*relu*7) identifies printfs with variables and format specifiers only (e.g. printf("%d%c", area, option);). RE for LU8 (*relu*8) identifies printfs with both messages and variables (e.g. printf("Area of a triangle = %f", area);). A limitation of using RE for identifying LUs in given task solutions and worked-out examples is that RE do not verify if the number of variables matches the number of format specifiers in a printf statement. But since the worked-out examples and task solutions are provided by the domain experts, this limitation can be handled if ERS assumes that experts are always right and they provide the correct solution.

4.1.3 Proposed algorithm for knowledge extraction (KERE)

This study asserts that, to present relevant worked-out examples for a specific task in ERS, it is sufficient to extract the basic learning units (LU) of the task (from its solution) and evaluate if these LUs match closely to the LUs of existing examples in ERS's database, thus eliminating the need for a complex syntax tree representation of each example and

Algorithm 2 Knowledge extraction using regular expressions (KERE)

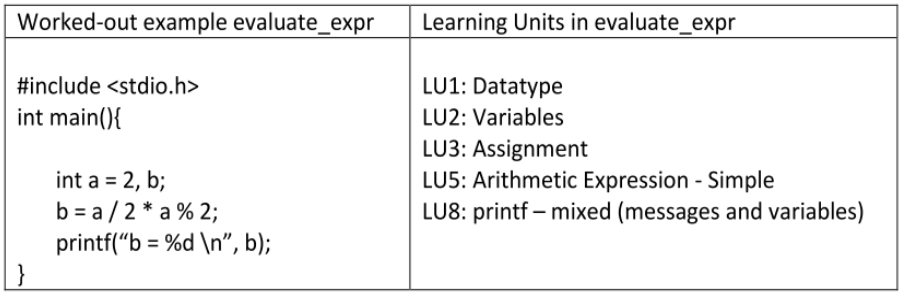
```
Inputs:
1. learning Units (LU) in the domain
2. set of regular expressions {RE} - one for each concept in LU
3. string $TE_{soln}$ of task / example solution
Output: LU_TE: list of {LUs that $TE_{soln}$ covers, number of times a LU occurs in $TE_{soln}$}
Method:
***begin of KERE
1. LU_TE=[]
2. Clean $TE_{soln}$ by removing comments, header files (including main's header)
3. for each concept $C_n$in LU
    3.1. let $re = RE[C_n]$
    3.2. $C_n.count$ =number of times a string in $TE_{soln}$ matches $re$
    3.3. if $C_n.count >= 1$, then
        3.3.1.append $(C_n, C_n.count)$ to $LU\_TE$
***end of KERE
```

task as done by many existing systems such as NavEx (Yudelson & Brusilovsky, 2005). The proposed system defines an algorithm called KERE (Knowledge Extraction using Regular Expressions) (Chaturvedi & Ezeife, 2015b) to identify the presence of one or more LUs in all those task solutions and worked-out examples that are written as partial or complete C programs from a domain. KERE (algorithm 2) takes as input a task solution or a worked-out example represented as a string (TE_{soln}) and outputs its LUs. It first cleans TE_{soln} by removing the comments and header files in it. Then it does a string pattern matching of the clean TE_{soln} against the regular expressions of each LU to find the presence of a LU in TE_{soln} and returns the number of times the LU appears in it.

Figure 4.1 shows a worked-out example and the LUs extracted from it by KERE. Similarly, worked-out example in figure 1.1 contains LUs {LU1, LU2, LU3, LU5, LU6, LU8, LU11}.

4.2 Knowledge Representation (KR) in ERS

Knowledge in ERS, refers to both domain knowledge and student knowledge (section 3.4).



Worked-out example evaluate_expr	Learning Units in evaluate_expr

`#include <stdio.h>`
`int main(){`
`    int a = 2, b;`
`    b = a / 2 * a % 2;`
`    printf("b = %d \n", b);`
`}`
	LU1: Datatype
	LU2: Variables
	LU3: Assignment
	LU5: Arithmetic Expression - Simple
	LU8: printf – mixed (messages and variables)

Figure 4.1: Worked-out example evaluate_expr and its LUs extracted by KERE

$$Knowledge_{ERS} = Domain \quad Knowledge \bigcup Student \quad Knowledge$$

$$= \{\{LUs, Task, Task \quad Solutions, WorkedOut \quad examples\} \bigcup$$

$$\{\{Static \quad information\} \bigcup \{Marks \quad in \quad individual \quad LUs\} \bigcup$$

$$\{Marks \quad in \quad assigned \quad tasks\} \bigcup \{learning \quad behavior\}\}$$

Information about these diverse entities and their relationships are best represented using an Entity-relationship model (ER) (Chen, 1976). An ER model is a graphical representation of entities and their logical relationships such that they can be easily transformed into an implementation model such as a relational database model. The motivation for storing ERS data in a relational database is that it facilitates data integration of multiple entities playing a role in ERS, and allows for easy and convenient addition, deletion and modification of the data stored in the relational model.

4.2.1 Creation of basic building blocks of ERS's relational model

Each entity type in the ERS's ER model 4.2 is translated to a table in a relational database model. For each binary M:N relationship shown in the ER model (such as attempts, ELU and TLU), a new table is created with attributes that are the primary key attributes of the M and N side of the relationship and any relationship attribute, if any. For example, a new table called ELU is created with attributes (EID, LUID). For each 1:N relationship, the primary key of the N side of the relationship is added as an additional attribute to

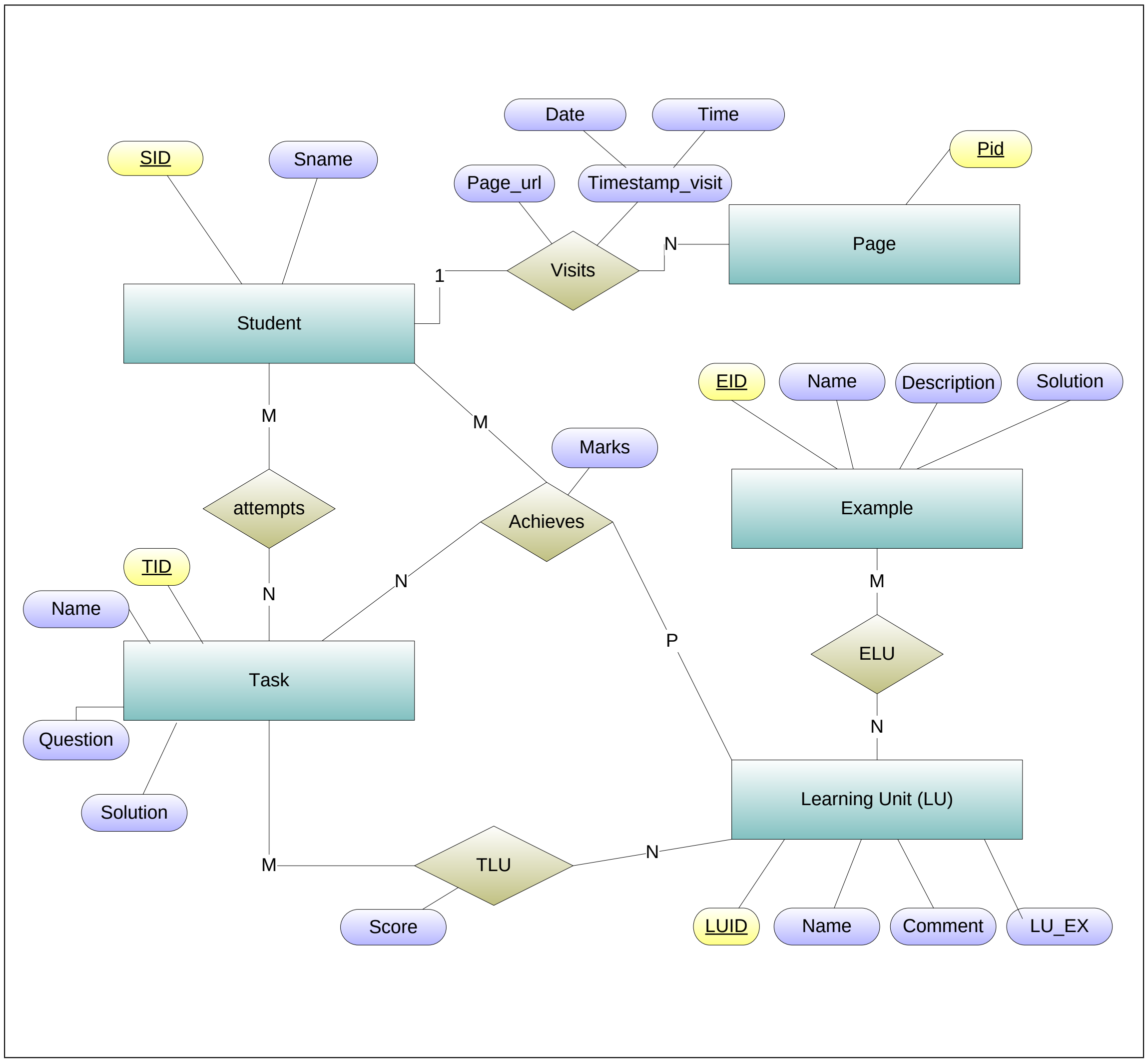


Figure 4.2: Relationships between students, tasks, examples and LUs represented as an entity-relationship model

the table on the 1 side. For example, table Page in the relational model has an additional (foreign) attribute called SID (indicating the pages that a student visits).

- Table LU, consists of the basic learning units in ERS's domain model3.1. Table LU has attributes (LUID : varchar2(3) primary key, Name varchar2(20), Comment : varchar2(2000), LU_EX varchar2(2000) that stores an example of the LU, LU_REG_EXP varchar2(1000) that stores the regular expression of the LU). For example, a statement to insert a new LU called LU30 into this table is : insert into LU values<'LU30', 'digits', 'one of more digits from 0..9','43', '\d+' >.

- Table Task stores details on each task in the ITS and has attributes (TID : varchar2(5), Name varchar2(20), Question varchar2(1000), Solution varchar2(2000)) . For example, a new task T25 is inserted as :

  insert into task values<'T25', 'task_to_find_sum_prod', 'Write a program in C to read 3 integer numbers from the users and display their sum and product.', '#include <stdio.h> int main(){ int x, y, z; int sum, product; scanf("%d%d%d", &x, &y, &z); sum = x + y + z; product = x * y * z; printf("%d\n", sum); printf("%d\n", product); return 0; }'>;

- Table Example stores details on each example offered by the ITS and has attributes (EId : varchar2(5), Name varchar2(20), Description varchar2(1000), Solution varchar2(2000)). For example, a new example E5 is inserted as :

  insert into example values<'E5','calculate_temperature', 'Write an equation in C to calculate temperature in Fahrenheit, given temperature in Celcius' , 'scanf("%f", & celcius); fahrenheit = 9 / 5 * celcius + 32; ' >;

- Table Student : Each student 's student model has a static content such as id, name and a dynamic content such as knowledge level on each LU. Student has static attributes (SId: varchar2(5), Sname: varchar2(50)). The dynamic attributes are stored in other tables such as Page and Achieves discussed below.

- Table attempts is created from the M:N relationship between tables Student and Task. Students marks achieved in all assigned tasks and the number of attempts made by the student are stored in this table. Its attributes are (<u>SID, TID</u>, marks, numberAttempts)

- Table TLU is created from the M:N relationship between tables Task and LU, and therefore its attributes are (<u>TID, LUID</u>, Score). For example, a task T1 has 2 learning units: LU12 with a weight or score of 3 and LU20 with a score of 4; the rows inserted in TLU are ('T1', 'LU12', 3) and ('T1', 'LU20', 4);
- Table ELU is created from the M:N relationship between tables Example and LU, and therefore its attributes are (<u>EID, LUID</u>). Students are not graded on worked-out examples, and therefore there is no attribute score in this table.
- Table Page logs the browsing history of students visiting a worked-out example or task. Its attributes are (<u>PID</u>, SID, page_url, visited_timestamp). ERS defines functions to retrieve the task or worked-out example visited from the page-url stored in this table.

There is also a ternary relationship between Task, Student and LU that stores the marks achieved by a student in a task's LU. It is converted to a table called achieves with attributes (<u>SID, TID, LUID</u>, marks). For example, student $S1$ achieves 2 marks in LU2 from task T1, and 4 marks for LU2 from task T5 will create 2 rows as ('S1', 'T1', LU2, 2) and ('S1', 'T5', 'LU2', 4).

4.3 Knowledge Customization in ERS

The prime goal of knowledge customization (KC) module in ERS is to build a list a worked-out examples that are most appropriate to the tasks assigned to students. The motivation behind customizing domain knowledge towards student needs is to avoid cognitive overload for students so that they can focus on the material suggested to them and succeed in assigned tasks with a much higher likelihood.

4.3.1 Computing the similarity between worked-out examples and task solutions

The core algorithms of this module and the following knowledge organization module described in section 4.4 are driven by similarity functions such as Euclidean distance (Pang-Ning et al., 2005). We define similarity function as a function that computes the similarity between assigned tasks and worked-out examples or between different worked-out examples. The choice of a similarity function depends on the type of data attributes used such as continuous, categorical and binary. Continuous data attributes are those that can be measured (e.g. weight of a person). Categorical data attributes define different categories of data but cannot be measured (e.g. gender, that has two categories 'F' and 'M'). Binary data attributes are a special case of categorical data, that can have only one of the two values 1 or 0. Binary data can be further categorized as symmetric and asymmetric data. A symmetric binary attribute is one in which the presence of a 1 is regarded as equally significant as its absence (0). For example, if gender is a binary attribute, where 1 represents female and 0 represents male, then a 1-1 match in two different classrooms (indicating a female-female match) is as significant as a 0-0 match (indicating a male-male match). Therefore gender is a symmetric attribute. An asymmetric binary attribute is one in which the presence of one of the values (e.g. 1) is regarded as more significant than the other. For example, if 'LU' is a binary attribute, where a value of 1 indicates the presence of an LU and 0 its absence, then a 1-1 match of a LU in two worked-out examples is significant, whereas a 0-0 match has no significance (since 0 implies that the LU is not present) and must be ignored. Therefore, LU is an asymmetric binary attribute. The most commonly used distance function is Euclidean distance, although it is meaningful only when data is measurable and its magnitude is significant. Euclidean distance between 2 data points x and y of dimension n is measured as

$$d(x,y) = \sqrt{\sum_{j=1}^{n}(x_j - y_j)^2} \tag{4.1}$$

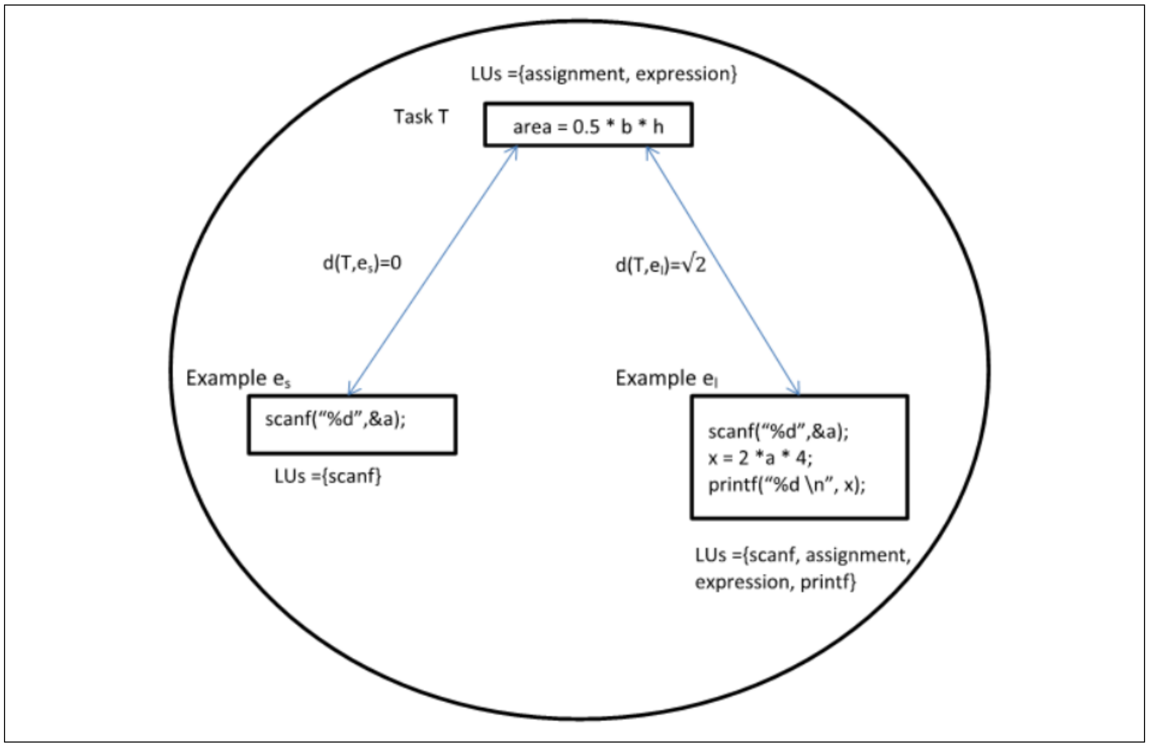


Figure 4.3: Distance between task T and examples e_l, $e_s$using number of LUs as magnitude - not an appropriate distance measure for ERS

where n = length of the vectors greatly depends on the length of the vectors being compared (Markov & Larose, 2007). For example, if a task T and two worked-out examples e_s (small) and e_l(large) are compared based on the number of concepts covered (magnitude), then d(T,e_l) will be much higher than d(T,e_s). This could be misleading since they both could be equally relevant to the LUs of T or example e_s can even be more relevant to T than e_l. Figure 4.3 shows a scenario that suggests that the length of an example (in terms of LUs covered by it) is not a good measure to compare 2 worked-out examples or an example with a task.

The dataset extracted by KERE was first proposed to be non-binary and sparse (Chaturvedi & Ezeife, 2014) (e.g. an example in which LU1 occurs once, LU2 occurs two times, and LU4 occurs once is represented as [1, 2, 0, 1]). We then normalized these non-binary values using a TFIDF weighting mechanism (Markov & Larose, 2007) used commonly in the field of information retrieval so that they fall in the range of 0 and 1. For example, after normalization, each worked-out example e_i is represented as a vector of weights $e_i = [w_{i1}, w_{i2}, \dots w_{in}]$, where each $w_{ij}, j = 1..n$ represents the weight assigned to this example's LU j. TFIDF uses term frequency (TF) combined with inverse document frequency (IDF) to compute a weight for every LU contained in a worked-out example. TF is computed as

$$TF_{ij} = n_{ij} \tag{4.2}$$

where n_{ij} is the number of times LU j occurs in example i. IDF for a LU j, is obtained by dividing the total number of worked-out examples in ERS by the total number of worked-out examples that contain LU j. Using TF and IDF, $TFIDF$ weight is computed as

$$w_{ij} = TFIDF(c_j, e_i) = TF_{ij} * log\left(\frac{N}{n_j}\right) \tag{4.3}$$

where N is the total number of examples in the database and n_j is the number of examples that contain LU j. To find similarity between vectors of TFIDF weights, ERS chooses cosine similarity (Markov & Larose, 2007) over Euclidean distance for reasons discussed above. Cosine similarity between a task solution and worked-out example e_i is computed as

$$CS(T, e_i) = \frac{\sum_{j=1}^{n}(w_{ij} * w_{tj})}{\sqrt{\sum_{j=1}^{n} w_{ij}^2} * \sqrt{\sum_{j=1}^{n} w_{tj}^2}} \tag{4.4}$$

The dataset was subsequently changed from non-binary to binary since it was observed that there were very few worked-out examples and tasks in ERS's domain that had more than one occurrences of LUs (Chaturvedi & Ezeife, 2015b). The most common similarity functions used with binary data are Jaccard's coefficient (JC) (Jaccard, 1901) and Hamming distance (Hamming, 1950). JC works best with binary asymmetric attributes (Pang-Ning et al., 2005) and therefore are more applicable to ERS's domain data. Jaccard's coefficient between two binary vectors x and y is measured as

$$JC(x, y) = \frac{f_{11}}{f_{11} + f_{01+}f_{01}} \tag{4.5}$$

where f_{11} is the frequency of occurrence of 1 and 1 in the corresponding bits of x and y, f_{01} is the frequency of occurrence of 0 and 1 in the corresponding bits of x and y and f_{10} is the frequency of occurrence of 1 and 0 in the corresponding bits of x and y. Here, f_{01} and f_{10} represent the non-matching attribute pairs. For example, if x = [1, 0, 0, 1] and y = [1, 0, 1, 0], then $JC(x, y) = 1/3$. Hamming distance HD is also used with binary data and is defined as the number of bits that are different in two binary vectors x and y measured as

$$HD(x,y) = \frac{f_{01} + f_{10}}{f_{11} + f_{01+}f_{01} + f_{00}} \tag{4.6}$$

For example, $HD(x, y) = 1/2$. In section 5.3, we compare the performance of our proposed algorithm that finds the nearest neighbors of each task with all our worked-out examples using both JC and HD values to further justify our choice of using JC over HD and other similarity functions.

4.3.2 GREPD: Generate Relevant Examples and Predict Difficulty of a task

Knowledge customization of ERS started its journey by building and implementing an algorithm called GREPD 3 (Generate Relevant Examples and Predict Difficulty of a task) (Chaturvedi & Ezeife, 2014) that takes as input the (transformed) vector representation of all worked-out examples and tasks solutions in ERS's domain (this transformation into vectors is done by KERE algorithm of KE module), and then uses k-nearest neighbor algorithm (defined in section 2.1.2.1) to output the list of worked-out examples closest to the current task and predict the difficulty level of the task. Each worked-out example e_i and task solution transformed by KERE is represented as a vector of LUs LU_1 to LU_n (where n = total number of LUs in ERS). For example, if n = 10 (the first 10 LUs in table 3.1 3.1) and worked-out example e_t is given as {int a; float b; scanf("%d", &a); scanf("%f", &b);}, then e_t 's vector representation is [1, 1, 0, 0, 0, 0, 0, 0, 0, 2]. GREPD then transforms these into vectors of TFIDF weights 4.3, which takes both local (TF calculates weights locally, taking into account a specific LU and worked-out example)

	worked-out example / task solution
E1	int a;
	scanf("%d", &a);
E16	int a,b,c;
	c = a*b;
	printf("c = %d \n", c);
T8	int a, b, c;
	int sum, product;
	scanf("%d%d%d", &a, &b, &c);
	sum = a+b+c;
	product = a*b*c;
	printf("Sum = %d and product = %d ", sum, product);

(a) Task solution T8 and worked-out examples $E1$ and $E16$

	LU1	LU2	LU3	LU4	LU5	LU6	LU7	LU8	LU9	LU10	LU11
T8	0.23	0.23	0.76	0.76	3.04	0	1.52	1.01	0.34	0.61	0.76
E1	0.23	0.23	0	0	0	0	0	0	0.34	0.61	0
E16	0.23	0.23	0.76	0.76	0	0	0	1.01	0.34	0	0

(b) TFIDF weights for task T8 and examples E1 and E16

Table 4.3: Task solution T8 and worked-out examples $E1$ and $E16$ used to demonstrate algorithm GREPD (Chaturvedi & Ezeife, 2014)

and global information on worked-out examples (IDF computes the weights globally). Table 4.4a shows a task solution (T8) and two worked-out examples (E1 and E16) and table 4.4b shows their vector representation using TFIDF weights.

To break any ties between same or similar ($<= 0.05$) cosine similarity values, GREPD uses a new formula called modified cosine method (MCS) that multiplies the cosine value (equation 4.4) to the number of matching LUs between the worked-out example and task solution (we call it TR for tie resolution) and is computed as equation 4.7.

$$MCS(T, e_i) = CS(T, e_i) * TR \tag{4.7}$$

It sorts the MCS_T values in ascending order and stores the top k of them in a list L and then uses a voting mechanism to classify tasks as difficult or easy, as shown in algorithm 3.

Algorithm 3 GREPD : Generate Relevant Examples and Predict Difficulty of a task

```
Input:
1. $LU\_EX[m, n]$, where each m = vector representing a worked-out example, n = total number of LUs in the domain
2. $LU\_T$ : vector of size n representing a task solution T
4. $k$ : integer specifying the number of neighbors used in k-nn
5. $DL$ : vector of size m of actual class labels for each example in ERS (E for easy / D for difficult) as given by experts
Other variables
$MCS_T$: One-dimensional array of size 25, to store the modified cosine similarity (MCS) values between CM_T and every example in CM_EX.
Output
1. Compute TFIDF weights for each 1 .. m worked-out example given in $LU\_EX$ and for task T given as $LU_T$
2. Compute modified cosine similarity ($MCS_T$) between task T and each 1..m worked-out example
3. Sort the $MCS_T$ values computed in step 2 in ascending order and store top k of them in L1.
4. If the number of examples in L1 with difficulty level of 'E' is greater than number of examples with difficulty level of 'D', then classify task T represented by CM_T as 'E'; otherwise classify it as 'D'.
```

4.3.2.1 Limitations of GREPD

GREPD suffers from several limitations, which motivated us to improve upon it and propose a modified GREPD (MGREPD) that overcomes these limitations. Limitations of GREPD are:

1. Type of data used: Each worked-out example or task solution in GREPD is represented as a vector of continuous values - one for each LU, where each value represents the total number of times that LU occurs in it. ERS is an ITS based on the pedagogy of EBL (Example-based learning), in which students study worked-out examples to learn the domain and apply it to succeed in the assigned tasks. Therefore, the worked-out examples used in the domain of ERS are designed in such a way that a LU does not occur multiple times in them. This motivated us to change the type of domain data from continuous to binary and asymmetric, and change the similarity function from cosine similarity to Jaccard's coefficient (equation 4.5).

2. The actual class labels used in the k-nn algorithm of GREPD were manually given by experts for each worked-out example. MGREPD defines an algorithm that finds the difficulty level of each worked-out example.

4.3.3 MGREPD (Modified method to Generate Relevant Examples and Predict Difficulty of a task)

MGREPD (Algorithm 8) is designed to overcome the limitations of GREPD that was proposed in an earlier work done towards this thesis (Chaturvedi & Ezeife, 2014). Similar to GREPD, it uses k-nn classification algorithm to find the neighboring worked-out examples of each task. It takes as input m binary vectors (each of size n) representing m worked-out examples in ERS (LU_EX) , a vector LU_T of size n that stores the LUs of a domain task solution, an integer value k (for the number of neighbors) and a matrix DL that stores the actual difficulty level (DL) of each worked-out example (DL is used as the class label attribute). MGREPD uses a simple algorithm called findDL (algorithm 4) to assign a difficulty level DL to each worked-out example as 'E' for easy and 'D' for difficult. Algorithm findDL uses the expert knowledge stored in ERS's domain model to derive the value for DL. ERS's domain experts provide a list of LUs in ascending order of difficulty such that LU1 represents the simplest LU while LU50 has a higher level of difficulty and LU100 has even higher level of difficulty than LU50 4. These LUs are then partitioned into 2 disjoint sets of simple (S) and complex (C) LUs based on material difficulty. Algorithm findDL determines if a worked-out example is of difficulty level E or D using 2 factors:

1. *highestId*: Id of the highest LU covered by an example. Each worked-out example consists of one or more LUs. Since the LUs are arranged in ascending order of difficulty, *highestId* of an example can be used to assess its difficulty level. For example, worked-out example find_area in figure 1.1 has $highestId = L11$. An example's *highestId* can belong to either set S or C. A worked-out example is assigned to be difficult (D) if *highestId* belongs to set C. Otherwise, if *highestId* belongs to set S, then a second factor (number of LUs contained in it) is assessed.

Algorithm 4 findDL: compute actual class label (difficulty level E/D of examples)

Input:

LU_EX_i: binary vector of size n for LUs present (1) or not (0) in worked-out example i

Global variables (domain model accessible to findDL):

1.LUs partitioned by expert as simple S / complex C, based on material difficulty

2.Threshold value *thresh* (*thresh* is assumed to be 1/3 of the total number of LUs in ERS)

Output: DL_i: difficulty level of the example: 'E' for easy and 'D' for difficult

Method:

```
***begin of findDL
//This algorithm computes the actual class label (difficulty level of an example)
1. $highestId_i$ =id of the highest LU in $LU\_EX_i$
2. $numLU_i$ =total number of LUs present in $LU\_EX_i$
3. $DL_i$ is computed as :
// if $highestId_i$ is simple, implying that example i covers only
// simple LUs but the number of LUs needed to understand i
// (indicated by $thresh$) is high,
// then example i is assigned a 'D'
// (even though it has only simple LUs).
3.1. if $highestId_i \in \{S\}$
    3.1.1. if $numLU_i < thresh$
        $DL_i$ = 'E'
    3.1.2.else
        $DL_i$ = 'D'
// if $highestId_i$ is complex, then it is
// assigned a 'D'.
3.2. if $highestId_i \in \{C\}$
    $DL_i$ = 'D'
***end of findDL
```

2. *numLU*: The number of LUs that the worked-out example consists of. For example, find_area in figure 1.1 has $numLU = 7$. Domain experts of ERS determine that worked-out examples who have all LUs belonging to set S can also be assigned a DL of D, if the worked-out example has 'too many' LUs in it. For such scenarios, experts are required to provide a threshold value or formula to 'too many'. This threshold for marking the difficulty level of the material as provided by experts is 1/3 of the total number of LUs (e.g. if ERS has 24 LUs in its scope, a worked-out example *we* that has *highestId* as LU12 (LU12 belongs to set S) and $numLU > 8$, then *we* is assigned a DL = D (even though all LUs it contains are simple).

Table 4.4 shows three worked-out examples, their *highestId*, *numLU* and difficulty level DL as assigned by findDL, assuming that the total number of LUs is 26. EDL1 is assigned E, since all its LUs belong to set S (Simple). EDL2 is assigned D, since the LU with *highestId* is LU26 and it belongs to set C (Complex). EDL3 has all LUs as simple (since its *highestId* is LU14 and it belongs to S) but it has 'too many' LUs in it that exceed the threshold given by experts. Therefore EDL3 is assigned D.

MGREPD (algorithm 8) computes the similarity between a task solution and each worked-out example using Jaccard's coefficient (equation 4.5), sorts these JC values and generates a list l of k examples closest to the task using k_nn (defined in section 2.1.2.1). It then uses a voting mechanism on l to predict the task's difficulty level. If the total number of easy (E) examples in l is greater than the number of difficult (D) ones, then this task is predicted as easy; otherwise it is predicted as difficult.

4.4 Knowledge Organization in ERS

This module of the proposed ERS system focuses on organizing the list of worked-out examples in its domain based on the learning units (LU) they contain. The main motivation for proposing and implementing a learning unit based organization of worked-out examples is to enable ERS to assist students to prepare for final examination, when they must have completed all tasks and must have already learnt all the learning units. Typically, ERS suggests worked-out examples for the current task that a student is assigned. If students are looking to study worked-out examples at the end (meaning that these examples are not sought for any particular task), then the KO module will group all those worked-out examples that have related LUs into a single cluster (as opposed to traditional lesson-wise organization). For example, ERS's cluster for domain1 (C programming) that has arrays in it, will also have for-loops in it (ERS experts assume that most C codes that use one-dimensional arrays tend to use simple for loops and those with two-dimensional arrays tend to use nested-for loops).

ERS uses k-means clustering algorithm (algorithm 1) to cluster its resources but modifies the important steps of this algorithm to suit its functionality. It treats the problem

	Worked-out example	*highestId*	*numLU*	DL
EDL1: Find the area of a triangle, given its base and height.	#include <stdio.h> int main(){ float base, height; float area_of_triangle; printf("Enter the values for base and height:"); scanf("%f%f", &base, &height); area_of_triangle = 0.5 * base * height; printf("Area = %.2f\n", area_of_triangle); }	LU11 (S)	7	E
EDL2: Write function definition for a function fraction that takes 2 integers as input, and returns their fractional value numerator/denominator.	float fraction(int numerator, int denominator){ return (float) numerator / denominator; }	LU26 (C)	2	D
EDL3: Write C code to input integers a and b and print the following results: a+b, a+b-(a+b), a<b, a<b && a+b-(a+b)	#include <stdio.h> int main(){ int a, b, result1, result2, result3, result4; printf("Enter 2 integers:"); scanf("%d%d", &a, &b); result1 = a + b; result2 = a + b - (a + b); result3 = a < b; result4 = result3 && result2; printf ("Result1 = %d \n", result1); printf ("Result2 = %d \n", result2); printf ("Result3 = %d \n", result3); printf ("Result4 = %d \n", result4); }	LU14 (S)	9	D

Table 4.4: Difficulty level DL assigned to three worked-out examples by findDL

Algorithm 5 MGREPD : Modified algorithm to Generate Relevant Examples and Predict Difficulty of a task

```
Input:
1. $LU\_EX$: size m * n, each row m is a binary vector representing a worked-out example of ERS, n = total number of LUs
2. $LU\_T$ : binary vector of size n LUs (present (1) or not (0)) for task T
3. $k$ : integer specifying the number of neighbors used in k-nn
4. $DL$ : vector of size m of actual class labels for each example in ERS;
each class label is of type char (E for easy / D for difficult) computed using algorithm 7 (findDL)
Other Variables:
$JC_T$: One-dimensional array of size n, to store the Jaccard's coefficient $JC_T$ between $LU\_T$ and each example in $LU\_EX$
Output:
1. $List_{relevant}$: List of k examples most relevant to task T
2. Predicted difficulty level of task T as E or D
Method:
***begin of MGREPD
1. Compute Jaccard's coefficient $JC_T$ between $LU\_T$ and each row of $LU\_EX$ using equation 4.2.
2. Sort the $JC_T$ values computed in step 1 in ascending order and store the top k of them in $List_{relevant}$
(e.g. $List_{relevant}$=[2, 4, 1])
3. for each worked-out example $e_i$ in $List_{relevant}$
DL($e_i$) = findDL($e_i$)
DL stores the difficulty level of $e_i$ as 'E' or 'D'.
    countE = number of examples in $List_{relevant}$ with DL = E
    countD = number of examples in $List_{relevant}$ with DL = D
4. if countE > countD, then
    Predicted difficulty level of task T = 'E';
else
    Predicted difficulty level of task T = 'D';
***end of MGREPD
```

of organizing worked-out examples into coherent groups or clusters as a problem of clustering binary datasets. There are existing algorithms that cluster binary symmetric data efficiently but do not consider the asymmetric nature of data (as required by ERS's domain data). For example, Tao Li (2005) proposed a general framework for clustering binary data, in which they cluster both data and their features simultaneously using matrix approximation. They initialize k-means exactly as in Algorithm 1 by randomly picking initial cluster centroids, but recompute the centroids (step 2.2 of Algorithm 1) using an alternating optimization method that minimizes the approximation error between original clusters and the recomputed ones. Their algorithm works well with symmetric binary data that is not too sparse, unlike the data used in this research. Similarly, Ordonez (2003) proposed a modified k-means algorithm called Incremental k-means for symmetric sparse binary data. Incremental k-means uses Euclidean distance as the distance measure, and therefore suffers from the flaws described in figure 4.3. The initialization step of Incremental k_means uses estimation maximization (EM) algorithm (Markov & Larose, 2007) to compute the initial set of cluster centroids but uses sum of euclidean distances to recompute centroids (step 2.2 of Algorithm 1). The proposed KO algorithm implemented for ERS uses not only the local information available to clusters (similar to k-means) but also global information on domain data as explained in the following section.

4.4.1 KOM16: Proposed Knowledge Organization algorithm for ERS

The proposed algorithm KOM16 (Knowledge organization using Modified steps 1 and 2.2 of k-means 1) uses k-means as its basis but with 2 important modifications - (1) step 1 of k-means (algorithm 1) is modified to choose a set of initial centroids using knowledge of the local neighborhood. To find each successive initial centroid, ERS chooses the example that is farthest away from any of the already picked centroids. This guarantees that the initial set of centroids is well-separated. Section 4.4.1.1 presents this step as algorithm 7 (create_initial_centroids). (2) step 2.2 of k-means (algorithm 1) is customized to recompute centroids for binary data. Conventional methods such as computing the mean of all cluster members do not work well with binary data (Pang-Ning et al., 2005). Mode, instead of mean is commonly used for categorical attributes including binary attributes

```
Algorithm 6 KOM16 (Modified steps 1 and 2.2 of k-means)
Input: dataset $\partial$ of size m X n (m data points, each with n attributes), k (number of clusters)
Output: k clusters
Method:
**** begin of KOM16
1.  Choose k data points from $\partial$ as initial centroids using Algorithm 7 (create_initial_centroids)
2. repeat until convergence
    2.1. repeat steps until all m data points are exhausted
        2.1.1. assign point x to its closest centroid using Jaccard's similarity (equation 4.5)
    2.2. recompute the centroid of each cluster using Algorithm 8 (recompute_centroids)
**** end of KOM16
```

but is not useful in the example dataset used in our research. Section 4.4.1.2 proposes a novel algorithm called recompute_centroids (algorithm 8) to recompute centroid bit for each LU to either 1 or 0, depending on the global relevance of the LU, in addition to mode, which is a local property of the cluster.

Definition 6: Data point: A data point is a binary vector of size n, representing a worked-out example or a task solution in ERS, where n is the total number of LUs in the domain of ERS.

For example, worked-out example find_area shown in figure 1.1 can be represented as a data point [1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], assuming n = 25.

4.4.1.1 Neighborhood-sensitive choice of initial centroids (modified step 1 of k-means1)

Definition 7: Similarity vector : Similarity vector sv between a data point e and all data points in a set S is defined to be a vector of size q, where q = number of data points in S . Each element of sv stores the Jaccard's similarity coefficient JC (equation 4.5) between e and each data point in S. Both e and data points in S must belong to the same domain and must have the same number of features.

For example, let q = 3, e = 1010, S = { s1: 1001, s2: 1101, s3: 0110}, sv =[0.5, 0.25, 0.33]

The choice of initial centroids impacts the number of iterations required to get the final clusters by k-means algorithms. Keeping in mind the primary goal of organizing worked-out examples into clusters based on the LUs they consist of, ERS chooses as initial centroids those data points (or worked-out examples (section4.4.1)) that consist of LUs that are very different conceptually, implying that they should be placed in different clusters. With this rationale, we choose a farthest-neighbor approach (Pang-Ning et al., 2005) to pick the initial set of centroids. KOM16 chooses the first centroid randomly. Then, for each successive centroid, the worked-out example that is farthest away from any of the already chosen initial centroids is chosen using Jaccard's coefficient (equation 4.5). This guarantees that the initial set of centroid is well-separated. Algorithm 7 (create_initial_centroids) that describes this method picks the first worked-out example as the first initial centroid ($initial\_centroid_1$). Then it uses JC to generate the similarity vector sv (section 4.4.1.1) between $initial\_centroid_1$ and all other worked-out examples in ERS's domain. The worked-out example that has the least similarity value in sv (and therefore the farthest neighbor of $initial\_centroid_1$) is chosen as the next centroid ($initial\_centroid_2$). To find the third centroid (and the remaining ones), create_initial_centroid finds the LUs that are absent in the already chosen centroids ($initial\_centroid_1$ and $initial\_centroid_2$) and sets those bits to 1 in a temporary vector called *temp* - we call each bit of *temp* as μ_{ij} (jth bit of vector i) as defined in equation 4.3. Conceptually, *temp* stores all LUs not already covered by the existing initial centroids. Thus, *temp* is a good representative of the information on LUs covered by all the already chosen initial centroids.

$$\begin{aligned} \mu_{ij} &= 1, if\, \forall x \in \{initial\_centroids\},\, l = \#\{initial\_centroids\},\, \sum_{j=1}^{l} x_j < (l - \sum_{j=1}^{l} x_j) \\ &= 0,\, otherwise \end{aligned} \tag{4.8}$$

For example, if $initial\_centroid_1 = [\ 0\ 1\ 0\ 0\ 0\ 1]$ and $initial\_centroid_2 = [1\ 0\ 0\ 0\ 0\ 1]$, then *temp* vector $= [\ 0\ 0\ 1\ 1\ 1\ 0]$. Next, it computes sv between *temp* and the

Algorithm 7 create_initial_centroids (modified step 1 of k-means)

```
Input : binary matrix ∂ of m rows and n columns (each row of ∂ represents an example;
each column of ∂ represents a LU in the example) ,
desired number of clusters k
Output: matrix $initial\_centroid$ of k rows and n columns
Method:
***begin of create_initial_centroids
1. add first row in ∂ to $initial\_centroid[1]$—- each initial centroid is a vector of n 1s and
0s
2. compute similarity vector $sv$ between $initial\_centroid[1]$ and ∂ using JC (equation
4.2).
3. find the example that has least similarity value in $sv$ and add that as the next centroid
to $initial\_centroid[2]$.
4. initialize z to 3.
5. repeat until desired number of clusters k is achieved (until z=k)
    5.1 $\partial = \partial$ - {rows already stored in $initial\_centroid$}
    5.2 compute $temp$ by assigning each bit $\mu_{ij}$ of temp to 1 or 0 using equation 4.3.
    5.3 compute similarity vector $sv$ between $temp$ and all rows in ∂ using equation 4.2.
    5.4 find the example that has least similarity value in $sv$ and pick that as
$initial\_centroid[z]$; increment z
***end of create_initial_centroids
```

remaining examples in E to choose the example with the least similarity value as the next *initial_centroid*. This process is repeated until the desired number of clusters is reached.

4.4.1.2 Recomputing centroids for binary data (modified step 2.1 of k-means1)

Step 2.2 of k-means (Algorithm 2 1) recomputes the mean of all clusters to update the set of centroids, if necessary. For categorical attributes including binary ones, mode (instead of mean) of all clusters is commonly used to update centroids (Pang-Ning et al., 2005). Mode of an attribute refers to the value that occurs most often. For example, let cluster c created from an iteration i of KOM16 (Algorithm 6) has 7 data points in it, where each data point has 26 LUs as its features. If the first feature LU_1 for cluster c has values {1, 1, 0, 0, 1, 0, 1} for the 7 data points, then its mode is 1, since the number of 1s is more than the number of 0s. Therefore, the updated centroid's bit for LU_1 is assigned a value of 1. Conceptually, this implies that LU_1 is a good representative for this cluster. As this example illustrates, LUs that are contained in very few worked-out examples in the entire domain will suffer a bias, if mode is used to recompute centroid bits. For instance, in the above example, if feature LU_{26} for cluster c has values {0,0,0,1,1,0,1}, then using the mode method, centroid bit for LU_{26}= 0. But, if the total number of worked-out examples in ERS's domain with learning unit LU_{26} is just 4, then cluster c has 3 out of the 4 worked-out examples that ERS has with LU_{26}, and therefore, LU_{26} should be marked as a good representative of cluster c and its centroid bit should be assigned a value of 1 (instead of 0). We propose a novel algorithm recompute_centroids (algorithm 8) that uses a smart technique to assign a value of 1 to the new centroid bit in 2 situations: (1) when the mode is 1 (2) when the mode is 0 but the presence of that LU is more relevant globally. Algorithm 5 recomputes centroids by first calculating the mode of feature LU_i, for all worked-out examples in its cluster. If the mode is 1 for an attribute LU_i, then it is considered a good representative for the cluster and its bit is set to 1. If mode is 0, then the frequency of occurrence of attribute LU_i, in the entire database (α_i) is determined and used to find the remaining number of worked examples with LU_i in them (*residual*). If the number of times LU_i appears in cluster c is at least a certain threshold number of the remaining examples (*residual*), then the new centroid's bit for LU_i is assigned a value of 1. After experimenting with different threshold values, we chose a value of 30%.

4.5 ERS : algorithm and an example application

This section presents the main ERS algorithm and how it integrates the modules of knowledge extraction, customization and organization. It also includes an example that runs through each of these modules.

4.5.1 The ERS algorithm

The ERS_main algorithm calls the core algorithms for each of its 3 components KE, KO and KC as shown in Algorithm 9. Section 4.5.2 demonstrates an example application of ERS_main.

Algorithm 8 recompute_centroids (modified step 2.2 of k_means)

Input: initial_centroids of k rows and n columns, example dataset of m rows and n columns E, k clusters, *threshold*
Output: k new_centroids
Method:

```
***begin of recompute_centroids
1. repeat for each cluster k
    1.1. repeat for each attribute i of k
        1.1.1. calculate its mode of all members of cluster k
        Let ones = number of 1s; zeros = number of 0s
```

$$\begin{aligned} mode &= 1, \; if \; ones > zeros \\ &= -1, \; otherwise \end{aligned}$$

```
        1.1.2. if mode is equal to 1,
            1.1.2.1. assign new_centroid's bit i to 1
        1.1.3. else
            // $\alpha_i$ is the total number of examples in E that have i=1 in them
            1.1.3.1.$residual = (\alpha_i - ones)$,
            // cluster has at least threshold number of examples in E with LU i
        1.1.4. if $ones >= residual * threshold$,
            1.1.4.1. assign new_centroid's bit i to 1
        1.1.5. else
            1.1.5.1. assign new_centroid's bit i to 0
***end of recompute_centroids
```

Algorithm 9 ERS_main

Input: Worked-out Examples E, Task Solutions T, Learning Units (LU) and their regular expressions RE, Task Tq
Output:
1. All worked-out examples in E and task solutions in T transformed into binary vectors of size n, where n is the total number of LUs in ERS's domain
2. List of most relevant examples for Tq
3. Predicted difficulty level of Tq
4. Worked-out examples organized into different clusters based on the LUs they consist of

Method:

```
***begin of ERS_main
1. Call algorithm KERE of knowledge extraction (KE) module to create the learning unit binary matrix LU_TE (also referred to as $\partial$) from regular expressions and examples and task solutions.
2. Call algorithm KOM16 of Knowledge Organization (KO) module to define clusters of examples and tasks in the database.
3. Call algorithm MGREPD of Knowledge Customization (KC) module to select the most relevant list of examples for task $Tq$.
***end of ERS_main
```

L1	L2	L3	L4	L5
datatypes	variable	assignment	arithmetic expression - simple	arithmetic expression - compound

L6	L7	L8	L9	L10	L11
printf - messages only	printf - format specifiers only	printf - mixed	scanf - single input	scanf - multiple inputs	relational expression

L12	L13	L14	L15	L16	L17
logical expression	relational logical	arithmetic relational logical expression	simple while	simple for	if/else

Table 4.5: Learning units L1 - L17 defined for example 4.5.2

4.5.2 An example application of the proposed ERS system

Example 4.5.2 : Given the domain model, show how an ERS system extracts knowledge using KE, organizes all the existing examples into groups using KO, given k = 2, and mines the examples to generate a customized list for students attempting task T1E using KC.

Domain model (subset of ERS's domain D1) :

1. Learning Units : Let number of LUs in domain n = 17 as shown table 4.5.

2. Worked-out examples: E1, E2, E3, E4, E5 as shown in table 4.6.

3. Tasks and their solution: T1E and its solution shown in table 4.6. For simplicity, there is only one task in the scope of this domain.

Solution of example 4.5.2 : Steps 1 - 4 of algorithm ERS_main (algorithm 9) are executed as follows:

- **Step 1: KE: knowledge extraction:** The core algorithm of this component is KERE (algorithm 22).

  – Input

Example Id	Problem definition	Solution
E1	Write a statement to calculate temperature in Fahrenheit, given temperature in Celcius	fahrenheit = 9 / 5 * celcius + 32;

E2	Write statements to multiply and print 2 integers	c = a*b;
		printf("c = %d \n", c);

E3	Write a for loop that prints all alternate integers from 1 to n, where n = positive integer given as user input. So it prints 1, 3, 5, 7 and so on.	for(i = 1; i < n; i= i + 2)
		{
		printf("%d\n", i);
		}

E4	Write an expression to test the following : age is between 18 and 21 (inclusive).	(age>=18 && age <=21)

E5	Write an if statement to test the following : age is between 18 and 21 (inclusive). If condition is true, print "Eligible", otherwise print "Not eligible".	if (age>=18 && age <=21) printf("Eligible");
		else
		printf("Not eligible");

Task T1E	Write statements to find and print the last digit of an integer x.	x = 123;
		last_digit = x % 10;
		printf("%d", last_digit);

Table 4.6: Worked-out examples and Task solution used in example 4.5.2

	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15	L16	L17
E1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
E2	0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0
E3	0	0	1	1	0	0	0	1	0	0	0	1	0	0	0	1	0
E4	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0
E5	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	1

Table 4.7: dataset ∂ : 5 * 17 matrix to represent table 2's examples and their LUs (defined in table 4.5)

  * set of all LUs, let n=#LUs (n = 17 for example 4.5.2)
  * REs for each LU (REs for all LUs are not shown here. Some RE are listed in table 4.24.2)
  * Worked-out examples. For example, TE_{soln}(E1) = "fahrenheit = 9 / 5 * celcius + 32;"

- Method
  * KERE is applied to the worked-out examples in table 4.6 to obtain 5 binary vectors of size 17 (one for each worked-out example). For convenience, these 5 vectors are converted to a 5 * 17 binary matrix. KERE iterates through each of the 17 LUs to find a matching pattern in TE_{soln}(E1). If a matching pattern is found for an LU l, then a 1 is assigned to the row corresponding to $E1$ and column l of LU_TE ($LU\_TE(E1, l)$= 1), otherwise $LU\_TE(E1, l)$= 0. Row 1 of table 4.7 is assigned a 1 for columns 3 and 5, since example E1 matches RE for LUs assignment and arithmetic expresion-compound. Similarly example E2 of table 4.6 is translated to row 2 of table 4.7. It assigns 1 to columns 3, 4 and 8 indicating that E2 matches REs for LUs assignment, arithmetic expression - simple and printf - mixed. We call the resulting Boolean matrix as ∂.

- Output
  * 5 * 17 Boolean matrix ∂ shown in table 4.7.

• **Step 2: KO: Knowledge Organization:** The core algorithm of this component is KOM16 (algorithm 3).

- Inputs :
    - dataset ∂ (table 4.7)
    - desired number of clusters k = 2 and
- Method
    - Algorithm create_initial_centroids of KOM16 finds the initial set of cluster centroids for dataset ∂ in table 4.7 as {E1, E4}. It selects E1 as the first initial centroid. It then uses JC (equation 4.5) to find similarity with examples E2, E3, E4 and E5 as 0.25, 0.1667, 0 and 0. Therefore, example E4 that is the first most dissimilar (or farthest) example from E1 is selected as the next initial cluster centroid. Since the desired number of clusters is 2, create_initial_centroids stops here.
    - Following the rest of the steps of KOM16, examples E1, E2 and E3 are assigned to one group with E1 as the cluster centroid and E4 and E5 are assigned to group 2 with E4 as the cluster centroid. The algorithm converges in 2 steps but the cluster assignment remains the same in both the steps. The final cluster1 represents LUs {assignment, arithmetic expression - simple, arithmetic expression - compound, printf, relational expression, for loop}, whereas cluster 2 represents {relational expression, relational-logical combined, if/else}.
- Output
    - Cluster 1: {E1, E2, E3}
    - Cluster 2: {E4, E5}

- **Step 3: KC: Knowledge Customization :** The core algorithm of this component is MGREPD (algorithm 5).
    - Inputs:
        - dataset ∂ (table 4.7)
        - k = 3,

  - ∗ DL evaluated using algorithm 4 (findDL). DL for the examples in table 4.6 are found to be {'E', 'E', 'D', 'E', 'E'}.
    - ∗ Task T1E is transformed using algorithm KERE into a binary vector called LU_TE = [0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0], which indicates the presence of LUs 3 (assignment), 4 (arithmetic expression - simple) and 6 (printf - format specifiers only) in T1E.
  - Method:
    - ∗ MGREPD finds the JC between LU_TE and each row in ∂ as [0.25, 0.5, 0.33, 0, 0.1667], sorts these values in ascending order and stores the top k of them in $List_{relevant}$. Thus $List_{relevant}$ for task T1E = [E2, E3, E1]. This implies that students working on task T1E will see examples E2, E3 and E1 as the most useful examples to succeed in task T1E.
    - ∗ Since the difficulty level of 2 (out of 3) examples E2 and E1 in T1's neighborhood is 'E', T1E is assigned a difficulty level of 'E' (easy task).
  - Output
    - ∗ students working on task T1E will see examples E2, E3 and E1 as most useful examples to complete task T1E.
    - ∗ T1E is predicted as 'Easy'.

4.6 Chapter 4 Overview

Chapter 4 presents the core architecture of the proposed ITS called Example Recommendation System (ERS). It defines and proposes algorithms for the main components of ERS. The first component called knowledge extraction is considered to be the backbone of ERS. It's main algorithm KERE extracts LUs from given worked-out examples and task solutions in ERS's domain and represents them in an efficient way that enables ERS to be easily extendable to any new domain. Existing systems such as NavEx (Yudelson & Brusilovsky, 2005) generate syntax trees to extract LUs. Their method can validate the input (in addition to identifying an LU in it), but they do so at the cost of speed of operations, memory space demands and user inconvenience. RE are implemented using finite automaton based techniques such as Non-Deterministic Finite Automata (NFA) (Cox, 2007), which takes O(n) time to match an input string of length n. In a CFG, the overhead per input symbol in the input string is high, since it parses through a set of grammar rules to validate the structure of the input string. The memory required to match a input string to a RE is constant, whereas a CFG will require memory proportional to the depth of the syntax tree generated. Clearly, RE are as effective as syntax trees by making a simple assumption that worked-out examples and task solution don't need to be validated.

Knowledge customization module of ERS proposes algorithm MGREPD to find the k closest worked-out examples for each assigned task so that students do not have to search for such examples themselves, while attempting a task. MGREPD also predicts the difficulty level (DL) of each assigned task (DL is used in chapter 7 as a key feature in the prediction of student performance). The third component of ERS is knowledge organization (KO) that forms clusters of existing worked-out examples that are more meaningful in terms of the domain LUs. KO proposes novel ways to modify the two important steps of standard k-means clustering algorithm (choosing initial centroids and computing new centroids when required) such that the modifications cater to binary and asymmetric data. At the end, this chapter presents a trace of a working example that defines the domain model of ERS and runs through each component (KE, KC and KO) of ERS to produce the desired results.

Chapter 5

Evaluation of individual ERS components

This chapter evaluates the data mining algorithms proposed and applied in ERS's components (evaluation method EM1 as given in section 3.5.1). Section 5.1 highlights the datasets created and used by ERS. Section 5.2 presents an experimental analysis of the knowledge extraction component (described in chapter 4 section 4.1), section 5.3 presents the results and analysis of the knowledge customization module of ERS (described in chapter 4 section 4.3) and section 5.4 presents experiments proposed for the knowledge organization module of ERS (described in chapter 4 section 4.4).

5.1 ERS Datasets

There are two domains that ERS uses in its experiments (section 3.7). The datasets used in domain D1 and D2 are given below:

5.1.1 Dataset for Domain D1 (Programming in C)

- Total number of Worked-out examples = 250
  - Each worked-out example has the following format <ExampleId, Problem Definition, Solution>. For example, <E111, 'Write an assignment state-

ment to add 2 numbers a and b and store the result in aplusb', 'aplusb = a + b;'>. Table 5.1a shows 2 sample worked-out examples for domain D1. A full list of all 250 worked-out examples can be found at https://ritu106.cs.uwindsor.ca/example_list/ (Chaturvedi et al., 2015b) by logging in as bob / bobrocks.

- Total number of Tasks = 31
  - Each task has the following format <TaskId, Problem Definition, Solution, Scoring rule>. Scoring rule for a task defines marks assigned to each LU of the task. For example, <TE1, 'Write statements to find and print the last digit of an integer x.', 'x = 123; last_digit = x % 10; printf("%d", last_digit);', {LU3:1, LU4:2, LU7:1}>. A full list of all tasks can be found at https://ritu106.cs.uwindsor.ca/lessons/ (Chaturvedi et al., 2015b) by logging in as bob / bobrocks.
- Total number of LUs = 27
  - Each LU has the following format <LUid, LUname, Description>. For example, <LU3, 'Assignment', 'The assignment operator '=' in C allows the programmer to set a variable's value'>. Table 3.1 in section 3.7.1 has the list of all LUs in D1. A full list of all LUs can be found at https://ritu106.cs.uwindsor.ca/concept_map/ (Chaturvedi et al., 2015b) by logging in as bob / bobrocks (the website refers to LUs as concepts).

5.1.2 Dataset for Domain D2 (Programming in Miranda)

- Total number of Worked-out examples = 101
  - Each worked-out example has the following format <ExampleId, Problem Definition, Solution>. For example, <E22 , 'Write a program called p5 which takes two lists of numbers as input and which outputs the list with the largest value in the first position. If the lists have the same value in

the first position, then output the first list.', 'p5 n m = n, if n!0 >= m!0 = m, if m!0 > n!0'>. Table 5.1b shows 2 other sample worked-out examples for domain D2. A full list of all 101 worked-out examples can be found at http://ritu100.cs.uwindsor.ca/#/admin/manageExamples (Chaturvedi et al., 2015a) by logging in as rituch/rituch.

- Total number of Tasks = 12
  - Each task has the following format <TaskId, Problem Definition, Solution, Scoring rule>. Scoring rule for a task defines marks assigned to each LU of the task. For example, <TE1, 'Write a program called t2 which outputs the string "hello"', 't2 = "hello", {LU55: 2}>. A full list of all tasks can be found at http://ritu100.cs.uwindsor.ca/#/admin/manageTasks (Chaturvedi et al., 2015a) by logging in as rituch/rituch.

- Total number of LUs = 16
  - Each LU in domain D2 has the following format <LIid, LUname>. For example, <LU43, 'Empty List'>. Table 3.2 in section 3.7.2 has the list of all LUs in D2. A full list of all LUs can be found at http://ritu100.cs.uwindsor.ca/#/learningUnits (Chaturvedi et al., 2015a) by logging in as rituch/rituch.

5.2 Experiment 1 on Knowledge Extraction

This section answers research question RQ1 (chapter 3 section 3.3) on extracting knowledge from ERS's worked-out examples and task solutions correctly and with ease of extendibility to other domains. KERE (Knowledge Extraction using Regular Expressions), as described in section 4.1 identifies the presence of one or more LUs in all those task solutions and worked-out examples in an ITS domain. Section 5.2.1 demonstrates how it is easily extendable to a domain that teaches Miranda functional programming language and also to a non-programming Math domain. Section 5.2.2 validates the correctness of

Example Id	Problem definition	Solution
E1	Write a statement to calculate temperature in Fahrenheit, given temperature in Celcius.	fahrenheit = 9 / 5 * celcius + 32;
E2	Write statements to multiply and print 2 integers.	c = a*b; printf("c = %d \n", c);

(a) Sample worked-out examples in domain D1 (Programming in C)

Example Id	Problem definition	Solution
E1	Write a Miranda program that takes an integer n finds the sum of all integers starting from 1 leading up to and including n (e.g. sum2 5 = 1 + 2 + 3 + 4 + 5 = 15)	sum2 0 = 0 sum2 n = n + sum2 n-1
E2	Write a program m4 that divides an integer X by 5 and then squares it.	m5 x = (x div 5) * (x div 5)

(b) Sample worked-out examples in domain D2 (Programming in Miranda)

Table 5.1: Sample worked-out examples in domain D1 and D2

KERE by comparing the LUs generated by KERE with LUs generated manually by using a method called IOC.

5.2.1 Knowledge Extraction extendable to other domains

This section demonstrates the extendibility property of KERE to other domains and advantages of implementing it for any ITS. Every ITS that automatically extracts LUs from its resources requires the experts definition of the domain model, similar to KERE (e.g. LUs, solutions of all tasks and worked-out examples, algorithms for extraction). What makes ERS's KERE algorithm standout from other ITS is its simplicity, efficiency and extendibility to other domains with significantly less efforts by domain experts. In response to research question (section 3.3), this thesis asserts that it is not required by an ITS to verify syntactic relationships between the LUs of task solutions and worked-out examples in order to extract the LUs (as is done by existing systems such as (Yudelson & Brusilovsky, 2005; Mokbel et al., 2013; Hosseini & Brusilovsky, 2014)) - it just needs to identify the existence of a LU in the task solution or worked-out example. KERE is designed using this principle, which makes KERE more easily extendable to any new domain. The following sections 5.2.1.1 and 5.2.1.2 illustrate this property of KERE for a non-programming domain and for a different programming domain.

5.2.1.1 KERE extendable to non-programming domain

KERE's extendibility to a non-programming domain is illustrated using the following example (Example 4.1) that uses a small domain model for Math with a very limited scope of adding and subtracting fractions.

Example 4.1: Given a domain model $D1_{math}$ for Math with a limited scope that consists of adding and subtracting fractions with common denominators, show how KERE transforms each task solution and worked-out examples in the domain to a binary vector of size n, where n is the total number of LUs in $D1_{math}$.

Input Domain model of $D1_{math}$

Learning Units in the scope of $D1_{math}$ (all of type string)

L_{m1}**:** numerator (e.g. 3)

L_{m2}**:** denominator (e.g. /5)

L_{m3}**:** fraction of type string (e.g. $\frac{3}{5}$)

L_{m4} : add (e.g. 1 / 2 + 3 / 2)

L_{m5}**:** sub (e.g. 1 / 2 - 3 / 2)

L_{m6}**:** mixed (e.g. 4 / 2 - 3 / 2 + 1 / 2).

Regular Expressions for LUs in $D1_{math}$

$L_{m1}(numerator)$**:** \d+

$L_{m2}(denominator)$: \/\d+

$L_{m3}(fraction)$: \d+(\/\d+)?

$L_{m4}(add)$: \d+(\/\d+)?(\+\d+(\/\d+)?)

$L_{m5}(sub)$: \d+(\/\d+)?(\-\d+(\/\d+)?)

$L_{m6}(add\_Sub\_many)$: \d+(\/\d+)?[\-|\+]\d+(\/\d+)?([\-|\+]\d+(\/\d+)?)+}

For example, any string with 1 or more digits in it will be identified as a numerator, whereas a denominator is identified by a / followed by any number of digits.

Worked-out examples: Each example here has a problem definition and its solution.

1. E_{m1}: {Problem Definition: Represent '3 out of 4 slices' as a fraction; Solution: 3 / 4}
2. E_{m2}: {Problem Definition: Add 3 / 4 + 2 / 4; Solution: 3 / 4 + 2 / 4 = 5 / 4}
3. E_{m3}: {Problem Definition: Subtract 3 / 4 - 2 / 4; Solution: 3 / 4 - 2 / 4 = 1 / 4}
4. E_{m4}: {Problem Definition: Calculate 3 / 6 + 2 / 6 - 1 / 6; Solution: 3 / 6 + 2 / 6 - 1 / 6 = 4 / 6}

Tasks:

T_{m1}: Calculate 2 / 6 + 1 / 6 + 1 / 6.

Task solutions

$T_{m1}Soln$: 2 / 6 + 1 / 6 + 1 / 6 = 4 / 6

Solution Example 4.1: Each worked-out example in $E_{m1}..E_{m4}$ and task solution $T_{m1}Soln$ is transformed into a binary vector of size 6, where a value of 1 indicates the presence of an LU in it and 0 its absence. . For each example/task *et*, KERE iterates through the regular expressions for each of the 6 LUs to find a matching pattern in the solution of *et*. If a matching pattern is found for an LU l, then a 1 is assigned to the column l of LU_TE (i.e. $LU\_TE(l)$= 1), otherwise $LU\_TE(l)$= 0. For example, for E_{m1}, KERE finds a matching pattern for $L_{m1}(numerator)$, $L_{m2}(denominator)$ and $L_{m3}(fraction)$. Similarly, the LUs for examples $E_{m2}..E_{m4}$ and task T_m are extracted as $\{E_{m2} : \{L_{m1}, L_{m2}, L_{m3}\}, E_{m3} : \{L_{m1}, L_{m2}, L_{m4}\}, E_{m4} : \{L_{m1}, L_{m2}, L_{m5}\}, T_m : \{L_{m1}, L_{m2}, L_{m3}\}\}$.

Output of Example 4.1:

E_{m1}: [1,1,1,0,0,0]

E_{m2}: [1,1,1,0,0,0]

E_{m3}: [1,1,0,1,0,0]

E_{m4}: [1,1,0,0,1,0]

$T_{m1}Soln$: [1,1,1,0,0,0]

5.2.1.2 KERE extendable to a programming language from a different paradigm

KERE's extendibility is illustrated in this section using a programming language called Miranda. The motivation for choosing Miranda is 2-fold: (1) Miranda is a functional programming language, as opposed to C, which is a procedural language (Hughes, 1989). In procedural languages such as C, programmers focus on how to perform tasks or implement algorithms, whereas in functional languages such as Miranda, programmers focus on what information is desired (instead of how to achieve it). Such differences make the structure of these 2 languages (C and Miranda) very different structurally and this is what motivated to choose Miranda to validate KERE's process of knowledge extraction. (2) Miranda is taught to Undergraduate Computer Science students as a core course in the University of Windsor and its resources were readily available to create ERS's domain model (Frost, 2015).

KERE is implemented for Miranda with its domain model D2 (section 5.1) consisting of (1) 16 learning units (LU) as shown in table 3.2, (2) RE for each LU (RE for LU48, which identifies recursion in Miranda is given as [a-z]+.*\([a-z]+\:[a-z]+\).*), (3) 101 worked-out examples and (4) 12 tasks (and their solutions). Table 5.2 illustrates four Miranda worked examples, solution of a task T_{r1} and LUs extracted by KERE.

5.2.2 Validation of KERE in extracting the correct LUs

In order to answer RQ1 (section 3.3) and validate the correctness of the LUs extracted, this thesis compares the results of algorithm KERE of knowledge extraction in ERS with a manual extraction method called Item-Objective Congruency (IOC) used commonly in the area of Content Validity (Rovinelli & Hambleton, 1976; Li & Chen, 2009). The IOC method collects and analyses judgments from several experts on the relevance of an item

	Question / Task	Worked-out examples / Task Solution	LU's extracted by KERE
E_{r1}	Write a Miranda program that takes an integer n finds the sum of all integers starting from 1 leading up to and including n (e.g. sum2 5 = 1 + 2 + 3 + 4 + 5 = 15)	sum2 0 = 0 sum2 n = n + sum2 n-1	LU1, LU11, LU12
E_{r2}	Write a program m4 that divides an integer X by 5 and then squares it.	m5 x = (x div 5) * (x div 5)	LU1, LU11
E_{r3}	Write Miranda program called p1 which outputs the list [(42,"Ford Prefect")]	p1 = [(42,"Ford Prefect")]	LU11, LU14, LU15
E_{r4}	Write a program called m7 that uses list comprehension to double every number in the input list l.	m7 l = [x*2 | x<-l]	LU1, LU7, LU8, LU11, LU13
$T_{r1}Soln$	Write a recursive program called p8 which takes a list of lists of numbers as input and which returns a list containing the first number from each of the lists on the input; e.g. p8 [[12,7,3],[4,5,2],[6,8,23]] => [12,4,6]	p8 [] = [] p8 (x:xs) = x!0 : p8 xs	LU4, LU8, LU10, LU12

Table 5.2: Miranda worked-out examples and Task solution and their LUs extracted by KERE

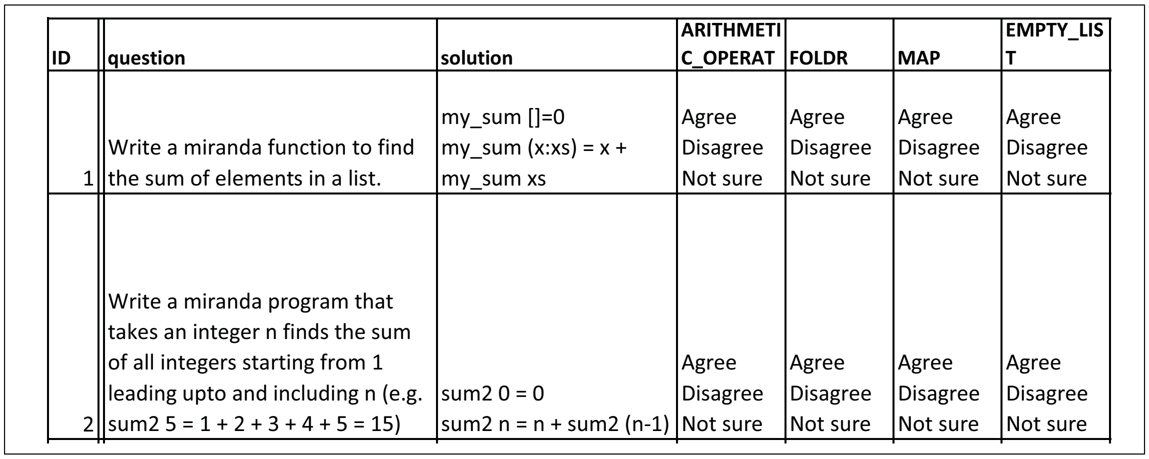
for an instructional resource (e.g. relevance of a LU in a worked-out example). This study uses the term LU, in place of item in IOC and worked-out examples and task solutions in place of instructional resources. Typically, experts give their judgment for each LU i in worked-out example or task solution k as a rating of 1 (if they strongly believe that k must consist of LU i), -1 (if they strongly believe that k must not consist of LU i) and 0 (if they are not sure). It then compiles all the expert ratings to compute a validity index value for each LU i in worked-out example k as shown in equation 2.1 and re-written here for convenience as

$$I_{ik} = \frac{(n-1)\sum_{j=1}^{q} X_{ijk} + n\sum_{j=1}^{q} X_{ijk} - \sum_{j=1}^{q} X_{ijk}}{2(n-1)q}$$

where n = total number of LUs, q = total number of experts and X_{ijk}= the rating (1, 0, -1) of LU i on worked-out example k by expert j. Steps used in this thesis for the process of collection and analysis of the LUs extracted by IOC and comparing them with KERE's extraction of LUs for worked-out examples and task solutions in ERS's domain are:

Step 1: Collect experts ratings on LUs of worked-out examples and task solutions in ERS. ERS uses five experts for each of its domains (Domain D1: C Programming and Domain D2: Miranda Programming). Experts for both the domain are Undergraduate Computer Science students. Experts are requested to give their judgment or rating on whether a LU i meets the objectives of worked-out example k. Figure 5.1 shows a sample collection sheet with 2 worked-out examples (IDs 1 and 2) given to domain experts of Miranda. Learning units of ERS's domain D2 (Miranda Programming) can be found in table 3.2.

Step 2: Experts ratings from step 1 are compiled to compute the validity index I_{ik} of LU i for worked-out example k using equation 2.1. For example, if the five experts give their ratings for LU51 (PRIMITIVE_TYPE) of worked-out example E_{r3} (figure 5.1) as [1, -1, 1, -1, 1] , then $I_{ik} = 0.2$, where $k = 1$ and $i=$ LU51.

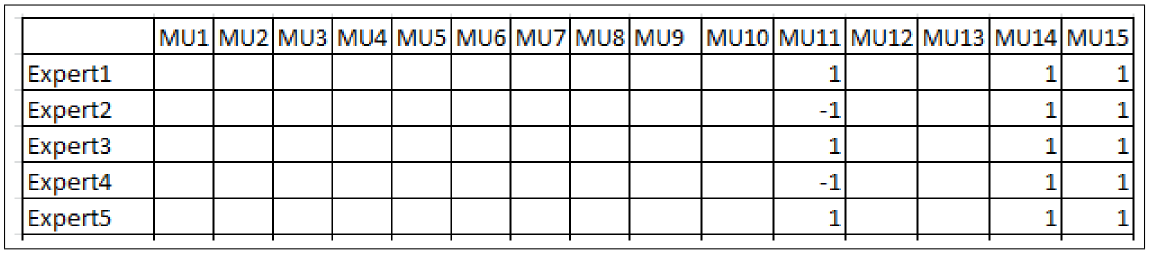


ID	question	solution	ARITHMETIC_OPERAT	FOLDR	MAP	EMPTY_LIST
1	Write a miranda function to find the sum of elements in a list.	my_sum []=0 my_sum (x:xs) = x + my_sum xs	Agree Disagree Not sure	Agree Disagree Not sure	Agree Disagree Not sure	Agree Disagree Not sure
2	Write a miranda program that takes an integer n finds the sum of all integers starting from 1 leading upto and including n (e.g. sum2 5 = 1 + 2 + 3 + 4 + 5 = 15)	sum2 0 = 0 sum2 n = n + sum2 (n-1)	Agree Disagree Not sure	Agree Disagree Not sure	Agree Disagree Not sure	Agree Disagree Not sure

Figure 5.1: Sample sheet given to experts to collect their ratings on LUs for worked-out examples and task solutions - only 2 examples and 4 (out of 15) LUs are shown

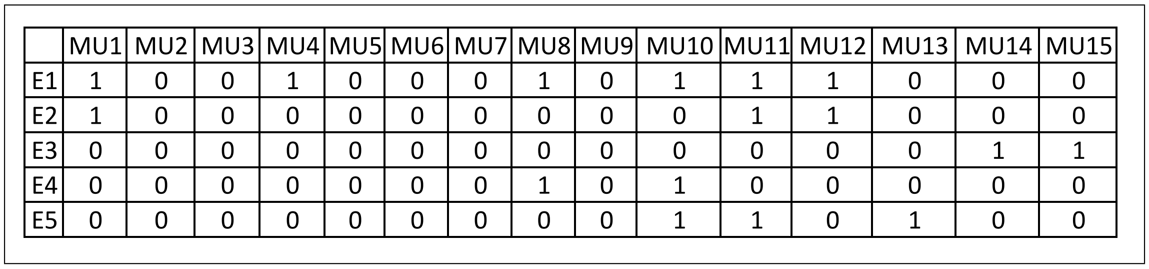
Step 3: IOC method requires domain experts of ERS to provide a threshold to agree with experts ratings. ERS uses a threshold of 0.60 for Domain D2 to indicate that if the validity index for a LU i of a worked-out example k is greater than 0.6, then it is assumed that experts agree that worked-out example k definitely consists of LU i. Using this threshold value, a Boolean matrix LU_by_experts of size m * n (where m = total number of worked-out examples and n = total number of LUs in ERS's domain) is constructed, that stores 1 at location LU_by_experts[k, i], if $I_{ik} > 0.6$ and 0 otherwise. For example, figure 5.2 shows experts ratings for worked-out example E_{r3} from ERS's domain D2 with 3 LUs (LU51, LU54 and LU55). All five experts agree on LU54 and LU55 but two of the five experts strongly believe that LU51 does not meet the objectives of worked-out example E3 (and therefore they give a rating of -1). The validity index for LU54 and LU55 is 1, but the index for LU51 is 0.2. Therefore, worked-out example E_{r3} in figure 5.3 has a 0 for LU51 but 1 for LUs LU54 and LU55.

Step 4: This step uses a simple matching coefficient (SMC) to match the LUs extracted by KERE with the LUs extracted by experts using the IOC method explained in step 3. Algorithm KERE extracts the LUs for each worked-out example (or task solution) and and stores it in a binary vector of size n, where n is the total number of LUs in ERS's domain 2.We access, for convenience, each row of LU_by_experts



	MU1	MU2	MU3	MU4	MU5	MU6	MU7	MU8	MU9	MU10	MU11	MU12	MU13	MU14	MU15
Expert1											1			1	1
Expert2											-1			1	1
Expert3											1			1	1
Expert4											-1			1	1
Expert5											1			1	1

Figure 5.2: Experts ratings for worked-out example E_{r3} from ERS's domain D2



	MU1	MU2	MU3	MU4	MU5	MU6	MU7	MU8	MU9	MU10	MU11	MU12	MU13	MU14	MU15
E1	1	0	0	1	0	0	0	1	0	1	1	1	0	0	0
E2	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
E3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
E4	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
E5	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0

Figure 5.3: Boolean Matrix LU_by_experts created for worked-out example E_{r3} and others

representing a worked-out example as a binary vector of size n. The equation to determine the simple matching coefficient (SMC) (Pang-Ning et al., 2005) between two binary vectors of the same size is given in equation 5.1.

$$SMC(x,y) = \frac{f_{11} + f_{00}}{f_{11} + f_{00} + f_{01+}f_{01}} \tag{5.1}$$

where $f_{11}$is the frequency of occurrence of 1 and 1 in the corresponding bits of x and y. Similarly, $f_{01}$is the frequency of occurrence of 0 and 1, f_{10} is the frequency of occurrence of 1 and 0 and f_{00} is the frequency of occurrence of 0 and 0 matches in the corresponding bits of x and y. In SMC, both f_{11} and f_{00} are considered as matching attribute pairs, whereas f_{01} and f_{10} represent the non-matching attribute pairs. For example, $SMC([10011],[10101]) = 3/5 = 0.6$. Results achieved using SMC between LUs extracted by KERE and IOC for domain D1 are shown in figure 5.4. Similarly, results achieved using SMC between LUs extracted by KERE and IOC for domain D2 are shown in figure 5.5. KERE extracts LUs for domain D1 with an 81% accuracy, indicating that consistency of agreement among experts is 81%. For domain D2, KERE extracts LUs for domain D1 with a 95% accuracy, indicating that experts agree very strongly with the LUs extracted by KERE. A low value of accuracy for the similarity between IOC experts and

KERE's LUs extracted for domain D1 can be attributed to the nature of conventional programming languages such as C. Sometimes, C code written for worked-out examples may require LUs that do not directly meet the learning objectives of the example, but are required to complete the C program. For example, LUs extracted by KERE for worked-out example EDL2 (table 4.4) are LU4 (arithmetic expression) and LU26 (function definition). EDL2 asks to write the function definition for a function called fraction that takes 2 integers as input, and returns their fractional value numerator/denominator is written as

```
float fraction(int numerator, int denominator)
{
    return (float) numerator / denominator;
}
```

Experts may not agree with LU4, since the objective of EDL2 is to write function definition and not to teach arithmetic expressions. Similarly, many worked-out examples in ERS's domain D1 have LU2 (variables) and LU6 (print constant messages) in them, only because they support the other LUs in meeting the objectives of worked-out examples or task solutions. KERE identifies them in the given worked-out example and extracts them as LUs but experts don't. The only constraint with domain D2 (Miranda Programming) is the presence of symbols such as < that can be interpreted in more than one ways. For example, worked-out example E_{r4} in table 5.2 uses list comprehension to double every number in the input list (m7 l = [x*2 | x <- l]). KERE identifies the symbol < as a relational operator and therefore extracts LU53 incorrectly, although <- in list comprehension means membership.

5.2.3 Complexity of KERE

Core of the knowledge extraction algorithm KERE is regular expression analysis. Identifying a LU in a given string of length w using regular expressions can be done in linear time, linear in the length of the string w (Cox, 2007; Dubé & Feeley, 2000). This is a

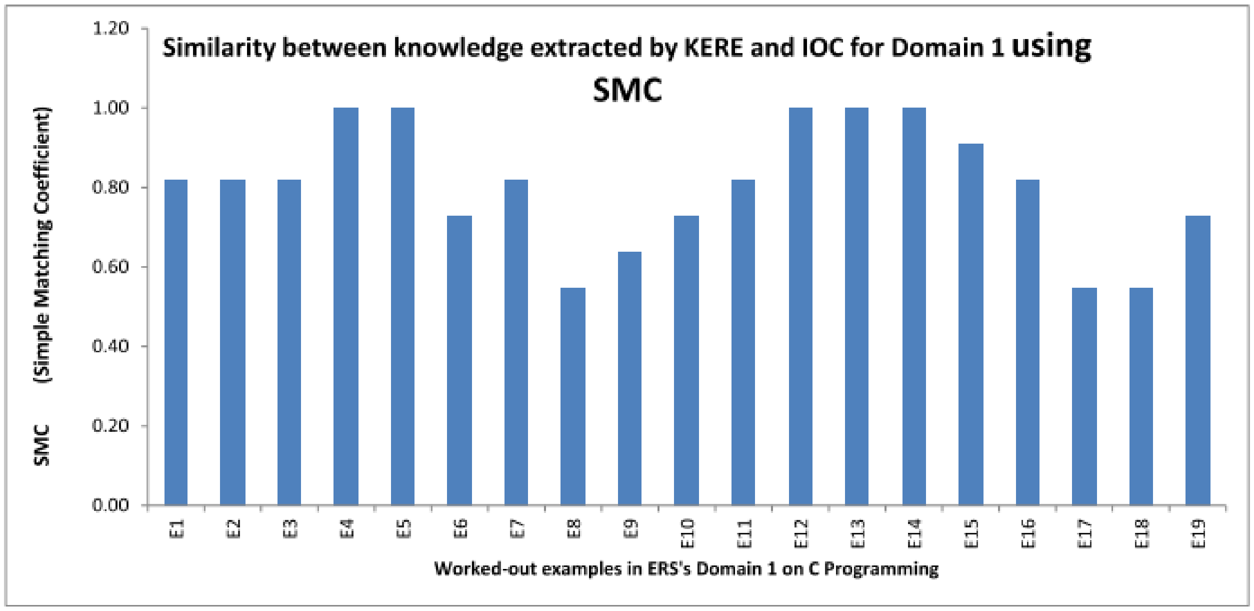


Figure 5.4: Comparing domain D1's LUs extracted by KERE and those extracted manually by IOC method for correctness

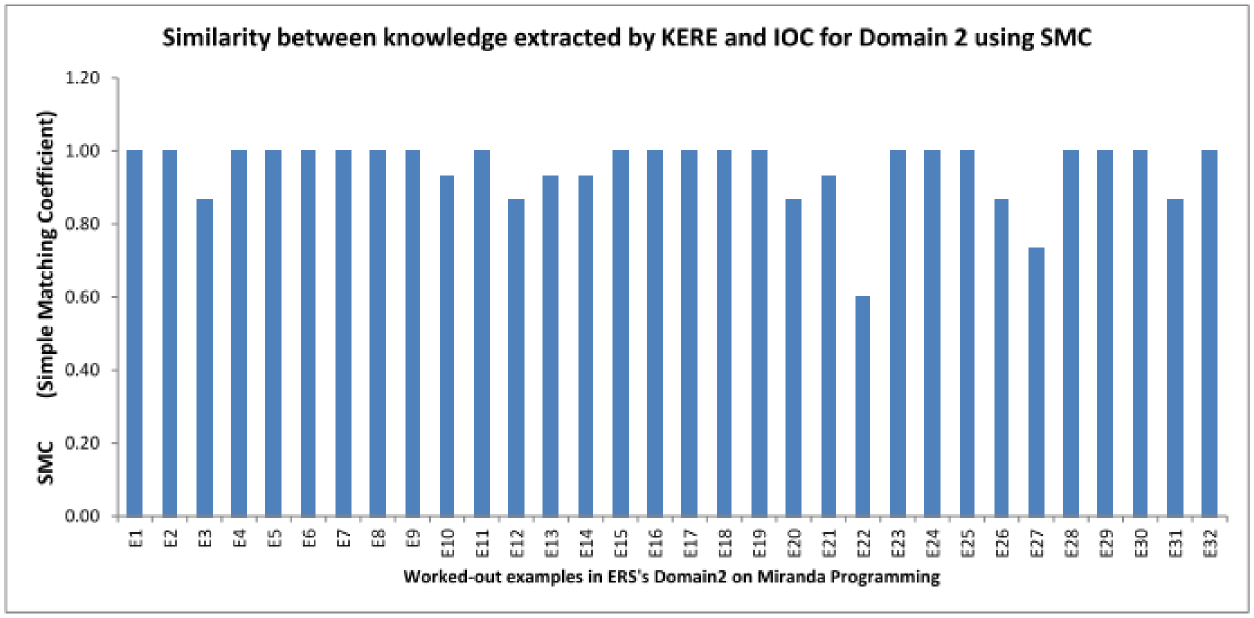


Figure 5.5: Comparing domain D2's LUs extracted by KERE and those extracted manually by IOC method for correctness

clear improvement over methods such as syntax trees used by existing systems, in which identifying an LU takes cubic complexity.

5.3 Experiment 2 on Knowledge Customization

Experiment 2 is an attempt to partially answer research question RQ2 (section 3.3) by validating customization algorithms GREPD and MGREPD described in section 4.3. Both these algorithms build a k-nearest neighbor prediction model using different similarity functions that are observed to be applicable to ERS datasets. This section describes the steps required to build a prediction model and the methods used to validate it.

Given a set of data records represented as (x,y) where x is a set of features, prediction is the task of mapping the feature set x into a special feature y, where y is termed as the class label. In general, steps to build a prediction model are:

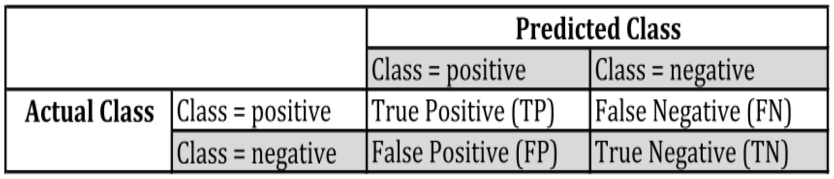
Step 1: Divide data into 2 subsets: training and test. Training dataset is used to build the model. Test dataset is used to test the model. Records in both the subsets have the x values, and the y values (these y values are referred to as actual class labels in step 3).

Step 2: Build the model by applying a prediction algorithm on the training dataset.

Step 3: Apply the model to test dataset - their actual class labels are compared to the predicted ones to evaluate the model.

Step 4: Use the model to classify unseen records and predict their y values.

Each time a record is tested for the class that it predicts, it is compared with its actual class label and a confusion matrix is generated. A confusion matrix is a table that allows visualization of the performance of a classification algorithm. It gives a count of data records or instances that are correctly and incorrectly predicted by the algorithm. In general, for a 2-class problem (such as the one used in this study), labels are termed as positive / negative; the original class labels are referred to as actual and those determined by the classification algorithm are termed as predicted. Figure 5.6 shows a generic



		Predicted Class	
		Class = positive	Class = negative
Actual Class	Class = positive	True Positive (TP)	False Negative (FN)
	Class = negative	False Positive (FP)	True Negative (TN)

Figure 5.6: generic confusion matrix for a binary class label (positive / negative)

$$Accuracy = \frac{Number\ of\ correct\ predictions}{total\ number\ of\ predictions} = \frac{(TP+TN)}{(TP+TN+FP+FN)}$$

$$precision = \frac{Number\ of\ actual\ positives\ correctly\ predicted}{number\ of\ predicted\ positives} = \frac{TP}{TP+FP}$$

$$recall = \frac{Number\ of\ actual\ positives\ correctly\ predicted}{number\ of\ actual\ positives} = \frac{TP}{TP+FN}$$

$$f_{score} = \frac{2}{\frac{1}{r}+\frac{1}{p}} = \frac{2 * r * p}{r+p}$$

Figure 5.7: Performance measures for classification algorithms used to predict student performance

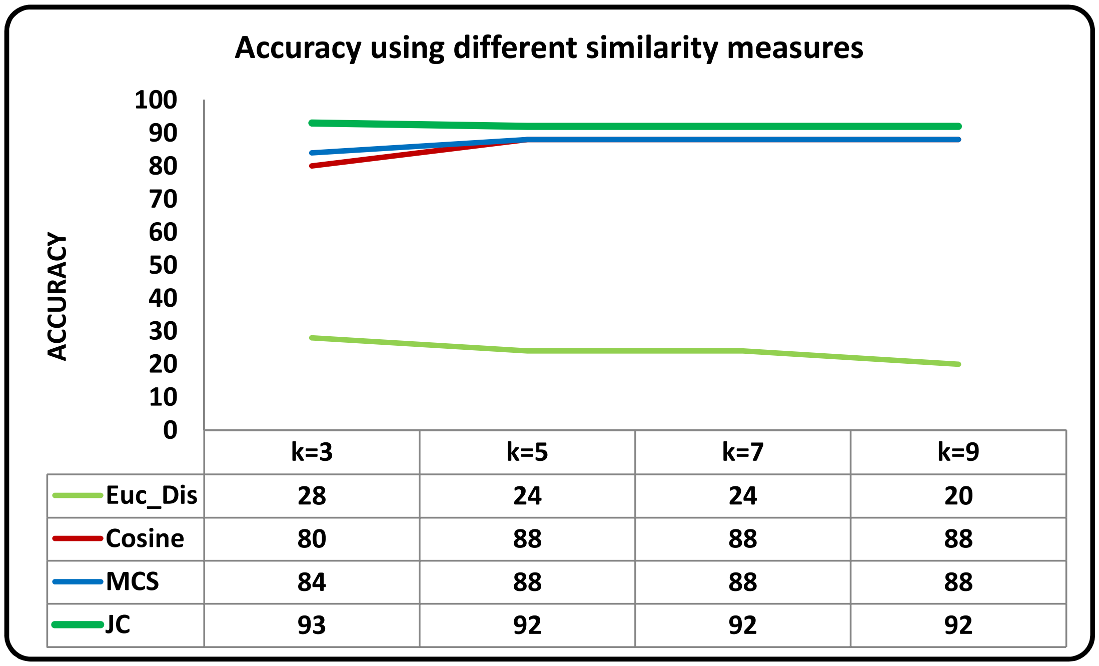
confusion matrix in which each column represents the total number of records in a predicted class, and each row represents the total number of records in an actual class. The classification algorithm then, assigns to each record or instance one of the following :

- True Positive (TP): actually positive - also predicted as positive.
- True Negative (TN): actually negative - also predicted as negative.
- False Positive (FP): actually negative - but predicted incorrectly as positive.
- False Negative (FN): actually positive - but predicted incorrectly as negative.

Performance measures such as accuracy, recall and f_measure (Markov & Larose, 2007) that typically evaluate prediction algorithms such as k-nn can be computed using confusion matrix as shown in figure 5.7.

There are different methods of evaluating prediction models such as cross validation and holdout methods (Payam et al., 2009). MGREPD uses leave-one-out cross validation

(LOOCV) to evaluate its model. In each iteration of LOOCV, one sample (e.g. ith worked-out example) from the complete dataset (of size N) is considered to be the test data (*test*) and the rest of the (N-1) samples are taken as training data. MGREPD predicts the difficulty level of test data using the class labels of training data. The actual class labels (difficulty level) of all N examples are computed using findDL (algorithm 4). At the end, each example's actual class label is compared against its predicted one to find the total number of correct predictions. To evaluate MGREPD, leave-one-out method of cross-validation (LOOCV) (Payam et al., 2009) and measures such as accuracy and f-score are used (Markov & Larose, 2007). Accuracy A measures the ability of the model to match the actual value of the class label with its predicted one (e.g. "Easy" predicted as "Easy" and "Difficult" predicted as "Difficult") as shown in figure 5.7. *Accuracy* is not a meaningful measure when dealing with class labels that are imbalanced or when one of the class labels is uncommon. For example, let us assume that 10 out of 100 examples are actually labeled as true or "Easy" and 90 are labeled as false or "Difficult". In a worse-case scenario, even if the classifier predicts only 1 (out of 10) "Easy" examples as "Easy", the accuracy computed as 91/100 = 91% is very high. Other measures that are used to evaluate classifiers are *precision* (defined as #actual true values predicted as true / total number of values predicted as true) and *recall* (defined as #actual true values predicted as true / total number of true values). Assuming that all easy examples are true and difficult ones are false, in this example, *precision*=1/10=10% and *recall* = 1 / 1=100%. Most classifiers achieve a trade-off between *precision* and *recall*, since it is very challenging to keep both the measures high. F-score is a combined measure that assesses this trade-off between *precision* and *recall* (figure 5.7). A comparative performance evaluation of MGREPD (that uses JC) against GREPD (that uses cosine and MCS) and other distance measures is shown in figures 5.8 and 5.9 for different values of k (where k = total number of neighbors). An analysis of these performance values validates our claim that Euclidean distance is certainly not a good measure for finding the closeness of worked-out examples with assigned tasks of any ITS. As the graph in figure 5.8 shows, MGREPD using JC performs the best with 93% when the total number of neighbors is 3. This implies that MGREPD's model correctly predicts the class labels for 93% of its

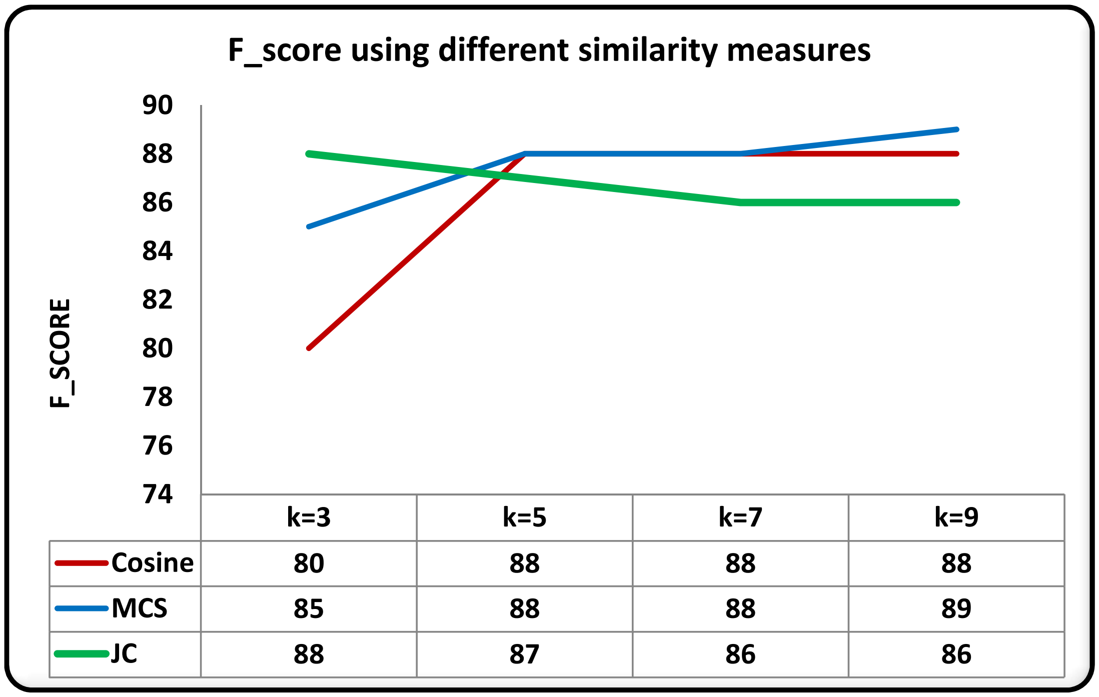


	k=3	k=5	k=7	k=9
Euc_Dis	28	24	24	20
Cosine	80	88	88	88
MCS	84	88	88	88
JC	93	92	92	92

Figure 5.8: Comparison of Accuracy of MGREPD using JC and other similarity measures such as Cosine similarity

test data records. MGREPD also performs with better accuracy than GREPD for other values of k (Chaturvedi & Ezeife, 2014). Measure f-score for MGREPD is found to be as high as 91% for k=1. For k between 2 and 9, it was found to be in the range of 86 - 88%. The algorithm did not perform as well as GREPD for higher values of k (k>=7), as shown in figure 5.9.

Figures 5.8 and 5.9 show results we published in our earlier work on finding a task's k closest neighbors using a dataset ℓ of 70 worked-out examples, 5 task solutions and 11 learning units from domain D1 on programming in C (Chaturvedi & Ezeife, 2014). We increased this dataset to a larger dataset $\mathcal{L}$ from two domains D1 and D2 with a total of 351 worked-out examples, 43 tasks and 43 learning units (section 5.1). Experiments with ERS show that accuracy and f_score values on applying MGREPD to larger dataset $\mathcal{L}$ are as high as 96% for k = 5 neighbors (compared to 93% for k = 3 neighbors when applied to the smaller dataset ℓ). These results are shown in figures 5.10, 5.12, 5.11 and 5.13.



	k=3	k=5	k=7	k=9
Cosine	80	88	88	88
MCS	85	88	88	89
JC	88	87	86	86

Figure 5.9: Comparison of f_score of MGREPD using JC and other similarity measures such as Cosine similarity

Using the larger dataset $\mathcal{L}$, we also compare results of MGREPD with yet another distance function called hamming distance, applicable to binary data, similar to Jaccard's coefficient. Hamming distance between 2 binary vectors is defined as the total number of bits in which they differ (equation 4.6). A comparative evaluation of performance of MGREPD using JC and hamming distance as similarity functions is shown in figures 5.10, 5.12, 5.11 and 5.13. These graphs indicate that MGREPD that uses JC performs better than hamming distance for both domains D1 and D2. This can be attributed to the fact that JC ignores the 0-0 matches and therefore works best with asymmetric binary data, similar to that of ERS, whereas hamming distance gives equal importance to both 1-1 and 0-0 matches by not counting any of them.

These graphs also indicate that MGREPD performs better with domain D2 (Programming in Miranda), as opposed to D1 (Programming in C), which can be attributed to the nature of functional languages such as Miranda compared to imperative languages such as C. Programs in C are written with a focus on the order of execution of the state-

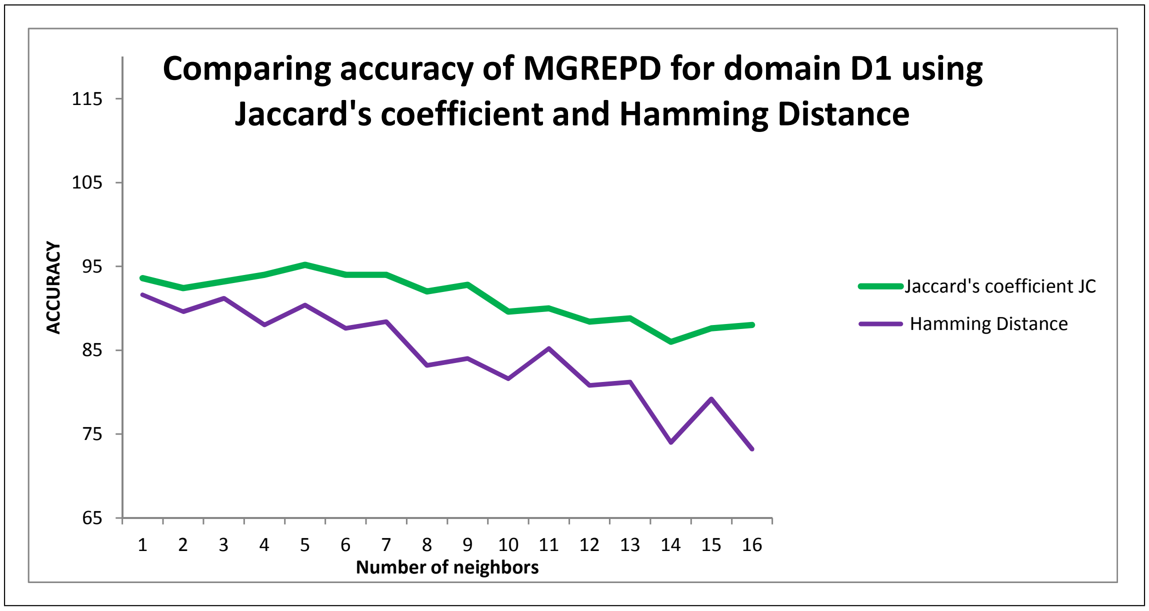


Figure 5.10: Comparison of accuracy of Jaccard's coefficient verses Hamming Distance for domain D1

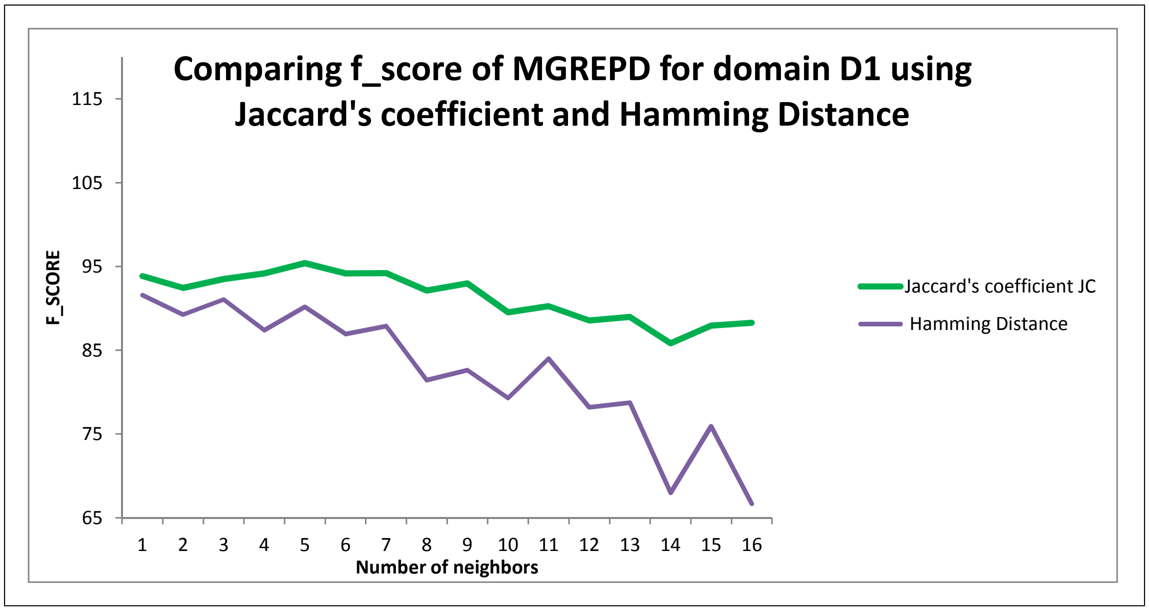


Figure 5.11: Comparison of f_score of Jaccard's coefficient verses Hamming Distance for domain D1

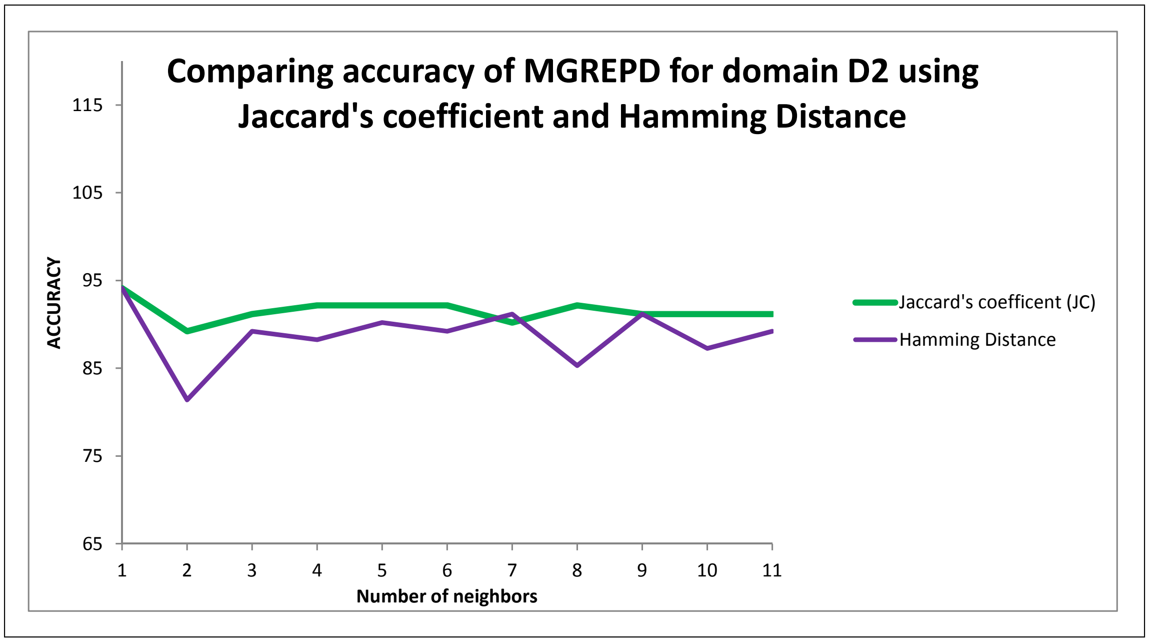


Figure 5.12: Comparison of accuracy of Jaccard's coefficient verses Hamming Distance for domain D2

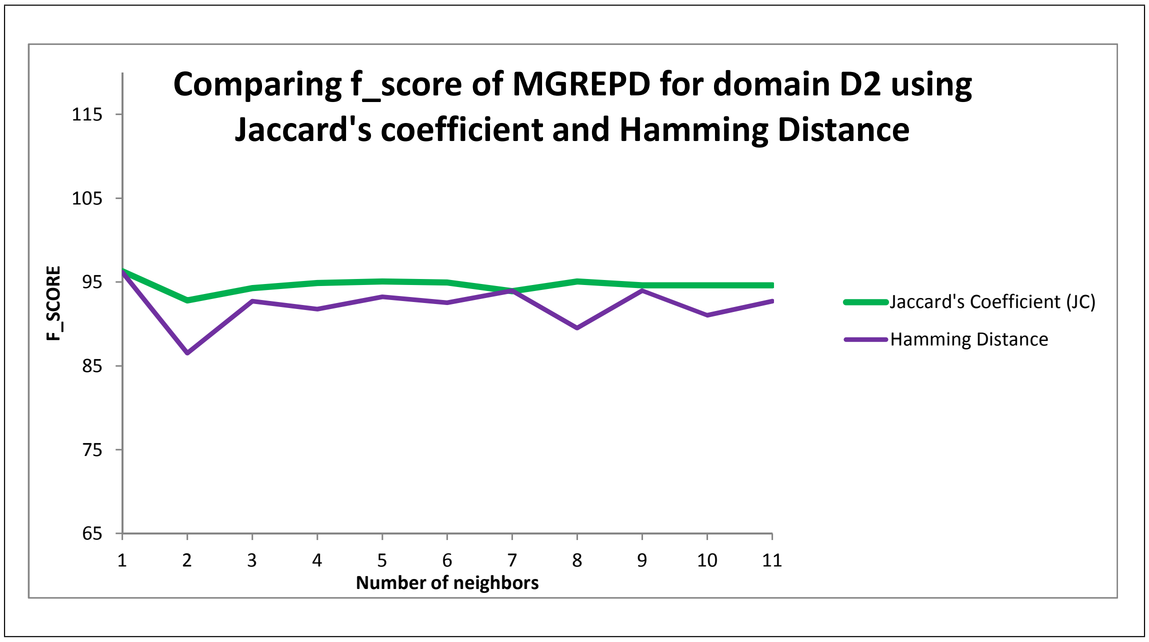


Figure 5.13: Comparison of f_score of Jaccard's coefficient verses Hamming Distance for domain D2

ments. Therefore, worked-out examples written in C tend to have LUs that may not be direct prerequisties or outcomes of other LUs in them. Miranda programs, on the other hand, are written with a focus on what needs to be done to obtain results (instead of how). They do not rely on states of other statements. Therefore, LUs extracted from worked-out examples written in Miranda are strongly related to each other. This results in more accurate JC values when comparing D2's worked-out examples and tasks.

5.3.1 Knowledge Customization extendable to other domains

As section 4.3 explains, knowledge customization module of ERS requires two algorithms: MGREPD and findDL. MGREPD, excluding findDL, is domain-independent, as long as all the worked-out examples and task solutions in its domain are represented as vectors of binary asymmetric values, where 1 indicates the presence of an LU in them and 0 the absence. Algorithm findDL, on the other hand, requires domain information from experts such as LUs partitioned by expert as simple S / complex $C4$.

The core idea of MGERPD is also implemented in a completely different research area in a prior work that we did (Chaturvedi et al., 2015c), in which we propose an algorithm called PEP (Predicting Emotions in Players) that predicts emotions in players based solely on game design features that are represented as asymmetric binary data. We apply k-nn classification algorithm on a reduced set of binary features to find a game g's nearest neighbors and their expected emotions. The expected emotions of these neighbors are then used to predict emotions triggered by g.

5.3.2 Complexity of GREPD and MGREPD

Both GREPD and MGREPD algorithms for knowledge customization use k nearest neighbor (k-nn) algorithm and therefore their complexity is defined to be the complexity of k-nn. Complexity to find k-nearest neighbors of a task t from a set of m number of worked-out examples, each of size n (where n is the total number of LUs) is $O(m * n) + O(m * k)$ (Markov & Larose, 2007), where complexity to compute distance from task t to all m examples is $O(m * n)$ and complexity to find the k closest examples is $O(m * k)$.

5.4 Experiment 3 on Knowledge Organization

Knowledge Organization (KO) (section 4.4) forms clusters of worked-out examples so that all examples belonging to a single cluster have related LUs. Algorithm KOM16 of KO modifies standard k-means clustering algorithm to accommodate the nature of ERS's data derived by its first component KE. Clustering can be evaluated using both internal and external validation measures (also known as indices). Internal validity index (such as Dunn's index) is based on the information intrinsic to the data alone, whereas external validity index (such as f_score) is based on previous information about data. Internal validity index are preferred when the class of data used in clustering is not known in advance (Maulik & Bandyopadhyay, 2002). Such indices are useful in validating the algorithms written for clustering, comparing them with others and in choosing the number of clusters. This thesis chooses to evaluate KOM16 using internal validity indices such as Dunn's index because (1) the objective of this experiment is to compare the performance of KOM16 with standard k-means for binary data (2) ERS does not have any pre-defined classes or clusters that can be used to compare the results of KOM16 for external validation. Dunn's index (Dunn, 1974) for each cluster partition, given the distance $d(x, y)$ between two data points x and y is defined as

$$Dunn = min_{1 \leq i \leq k}\{min_{1 \leq j \leq k, i \neq j}\{\frac{\delta(D_i, D_j)}{max_{1 \leq r \leq k}\{\triangle(D_r)\}}\}\} \tag{5.2}$$

$$\delta(D_i, D_j) = min_{x \in D_i, y \in D_j}\{d(x, y)\}$$

$$\triangle(D_r) = max_{x, y \in D_r}\{d(x, y)\}$$

In equation 5.2, δ represents the inter-cluster distance between 2 clusters D_i and D_j. $\triangle D_r$ measures the intra-cluster distances in cluster D_r. Larger values of Dunn's index indicate good clusters, implying high intra-cluster similarity and low inter-cluster similarity. The value of k that maximizes Dunn is then chosen as the optimal number of clusters.

This section experiments with algorithm KOM16 proposed in section 4.4 on ERS's dataset of worked-out examples (WOE dataset) and compares its cluster formation with clusters formed by standard k-means algorithm (algorithm 1 in chapter 2) using different parameters such as Dunn's index, total number of iterations to converge and interpretability. Table 5.3 shows results of 4 different experiments done with ERS dataset. This table indicates that Dunn's index is as high as 0.81 when KOM16 is used for cluster formation of ERS dataset as compared to standard k-means that gives a Dunn's index of 0.51. The optimal number of clusters formed by KOM16 for ERS is found to be 6, when Dunn's index is the highest (Dunn's index = 0.81, when k = 6). We compute the Dunn's index index for all values of k from 1 to 15, and find that k=6 has the highest value.

In order to validate the proposed clustering algorithm KOM16 further, we implement it on a benchmark zoo dataset (Bache & Lichman, 2013) and compare its results with standard k-means. Zoo dataset (Bache & Lichman, 2013), available at the UX Irvine Machine Learning Repository (uci.kdd) consists of 17 binary attributes and 1 categorical attribute. We transformed the non-binary attribute into 4 different binary attributes. Therefore, zoo dataset in our experiments consists of 101 instances (or records) of 21 binary attributes each (attributes such as hair, feathers, eggs and tail). There are 7 pre-defined classes of animals in the zoo dataset and the total number of animals in each class are 41, 20, 5, 13, 3, 8 and 10. For example, animals (frog, newt, toad) classify as class 5, whereas pitviper, seasnake, slowworm, tortoise, tuatara classify as class 3. KOM16, when applied to this zoo dataset, results in Dunn's index = 1.41 for 5 clusters (instead of 7). This can be attributed to the fact that classes 3 and 5 have very few instances in them (class 3 has just 5 animals in it and class 5 has 3 animals from a total of 101) and therefore KOM16 is not able to identify them as separate clusters. Figure 5.14 shows the Dunn's index computed for all values of k from 1 to 15 for both datasets (ERS and Zoo).

We also validate the cluster formation in zoo dataset by KOM16 against its actual pre-defined classes (Bache & Lichman, 2013) using external validation measures such as accuracy and f_score and use the results as a benchmark for ERS (since both ERS and zoo

Case	Algorithm	Number of clusters	Number of iterations to converge	Dunn's Index
1	Modified Step1 Modified Step6 (6 KOM16)	6	5	**0.802**
2	Modified Step1 Standard Step6	6	3	0.802
3	Standard Step1 Modified Step6	6	5	0.601
4	Standard Step1 Standard Step6 (Algorithm 1)	6	6	0.512

Table 5.3: Comparative Analysis of proposed modified k-means with standard k-means - ERS dataset

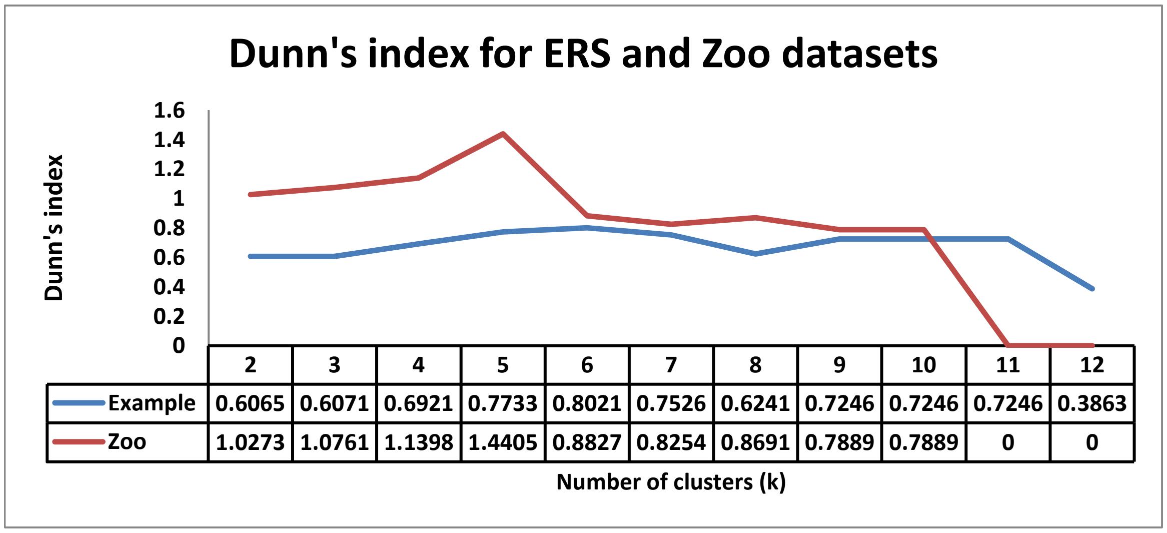
data are binary). A confusion matrix is generally created to compute such measures. A confusion matrix is a table that allows visualization of the performance of a classification algorithm 5.6. Each column of the matrix represents the instances in a predicted class, and each row represents the instances in an actual class. The confusion matrix in table 5.4 shows the total number of zoo classes predicted by KOM16 that match / mismatch the actual pre-defined ones. For example, row 1 in table 5.4 indicates that there are 36 (out of 41 animals in zoo dataset) that are predicted correctly as class 1, whereas there are 2 animals that are actually class 1 but are predicted incorrectly as class 2. Similarly, there are 3 animals that actually class 1 but are predicted incorrectly as class 5. Accuracy (given as total number of correct predictions / total number of predictions) for KOM16 is found to be 77% and f_score is computed as 79%, whereas f_score using standard k-means is computed to be 74%. Details on computing accuracy and f-score are given in section 5.3. This comparison shows that the proposed algorithm KOM16 is a viable algorithm to cluster data that is asymmetric and binary.

5.4.1 Complexity of KOM16

Complexity of KOM16 is the same as that of standard k-means clustering. Complexity of KOM16 to organize m number of worked-out examples with n number of LUs in each

		P	R	E	D	I	C	T
	class	1	2	3	4	5	6	7
A	1	36	2	0	0	3	0	0
C	2	0	11	9	0	0	0	0
T	3	2	0	0	3	0	0	0
U	4	0	0	0	13	0	0	0
A	5	0	0	0	1	3	0	0
L	6	0	0	0	0	0	8	0
	7	0	0	0	1	0	2	7

Table 5.4: Confusion matrix for zoo dataset to compare the actual animal classes with those predicted by KOM16



	2	3	4	5	6	7	8	9	10	11	12
Example	0.6065	0.6071	0.6921	0.7733	0.8021	0.7526	0.6241	0.7246	0.7246	0.7246	0.3863
Zoo	1.0273	1.0761	1.1398	1.4405	0.8827	0.8254	0.8691	0.7889	0.7889	0	0

Figure 5.14: Validity Indexes for k=1..12 for Zoo and ERS dataset

is $O(I*k*m*n)$ (Pang-Ning et al., 2005), where I is the number of iterations required to converge and k is the number of neighbors. Typically I is a small value and k is a value much smaller than m, therefore complexity of KOM16 can be considered to be $O(m*n)$.

5.5 Chapter 5 Overview

In this chapter, we present experimental analysis and results of each ERS component, i.e. KE, KC and KO. Algorithm KERE of KE is validated using 2 programming domains that are from different paradigms (domain D1: C Programming and domain D2: Miranda programming) on its correctness of extracting LUs from given worked-out examples and task solutions. It extracts LUs for domain D1 with an 81% accuracy, whereas domain D2 with a more promising 95% accuracy. KC uses k-nearest neighbor classification method to find a task's nearest worked-out examples and to predict the difficulty level of the task. It does so with an accuracy of as high as 93% and f_score of 88%. MGREPD, the core algorithm of KC, can be applied to any dataset that is binary and asymmetric. We applied MGREPD to a dataset from a completely different application area that represents emotions in players when playing a game. MGREPD predicts the emotion in players with an accuracy of 67% and f_score of 76% (Chaturvedi et al., 2015c). Algorithm KOM16 of ERS's knowledge organization module is validated using Dunn's index as the internal validity index. High values of Dunn's index for D1 indicates the formation of compact and well-separated clusters by KOM16. KOM16 is also implemented on a benchmark binary dataset Zoo (Bache & Lichman, 2013) to evaluate if the number of clusters formed by KOM16 are optimal. This is done by comparing these clusters formed by KOM16 with predefined classes for Zoo as given in UX Irvine Machine Learning Repository (Bache & Lichman, 2013) to compute an f_score of 79%.

Chapter 6

Evaluation of ERS as a tutor

This chapter describes and analyses the methods adapted to evaluate ERS as a tutor (evaluation method EM2 (section 3.5.1)), in order to validate its prime goal of improving student learning. Assuming that learning directly corresponds to high marks, the main goal, therefore, is to evaluate if ERS improves the likelihood of students scoring higher marks in the assigned tasks (ERS's students are considered successful in a task if they score a mark of 75 or higher in it). To accomplish this, ERS uses student data from Winter 2015 and Fall 2015 semesters of an Undergraduate course on C Programming for beginners, offered as a service course by the School of Computer Science at the University of Windsor. This course requires students to complete 10 individual assignments (worth 5% each) and a written final exam (worth 50%). Each assignment consists of 2 or more tasks, which, by definition 3.1, are gradable questions or instructions assigned to students (e.g. task T1: "There are 2.54 centimeters to 1 inch. Write a C program that asks a user to enter the value of his/her height in inches and then displays the height in centimeters.").

Two different scenarios are used to evaluate ERS using student and assignment data. In scenario 1, the same set of students (e.g. all students registered in Fall 2015) is offered 2 similar groups of assignments. Group I of assignments uses ERS to offer worked-out examples for its tasks, whereas group II of assignments does not use ERS. Student performances in these 2 sets of assignments are then analyzed in section 6.1 to find the group that improves student learning the most. In scenario 2 (section 6.2), performance of two different groups of students that use the same set of assignments is compared. The

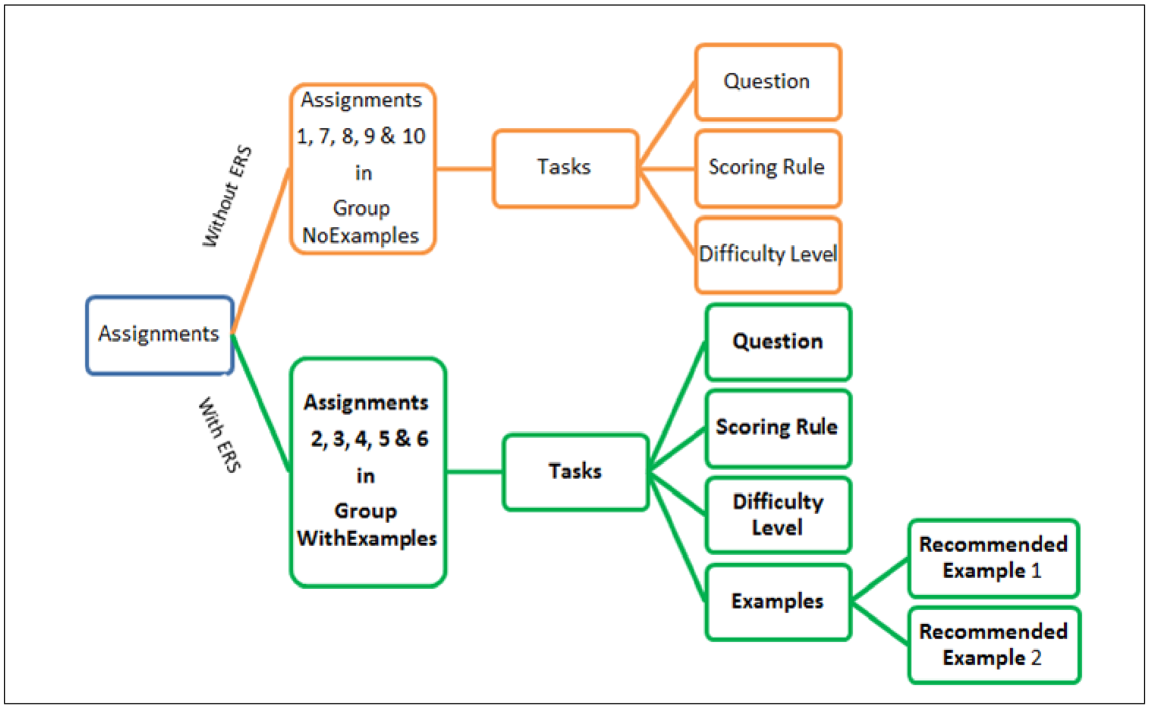


Figure 6.1: Hierarchy of Assignments and tasks used in evaluation of ERS (Scenario 1)

two groups of students are those who registered in Winter 2015 semester (when students do not use ERS for assignments) and those who registered in Fall 2015 semester (when students are required to use ERS).

6.1 Scenario 1: One group of students doing two similar set of assignments using two different approaches (no ERS and with ERS)

In this scenario, the required 10 course assignments are divided into 2 different groups - Group I is named NoExamples and Group II is named WithExamples. Group NoExamples consists of assignments 1, 7, 8, 9 and 10, and as the name suggests, ERS does not recommend any worked-out example for the tasks in these assignments. Group WithExamples consists of assignments 2, 3, 4, 5 and 6 and ERS suggests 2 most relevant worked-out examples for the tasks in these assignments. A complete hierarchy of assignments and tasks used for this scenario is shown in figure 6.1.

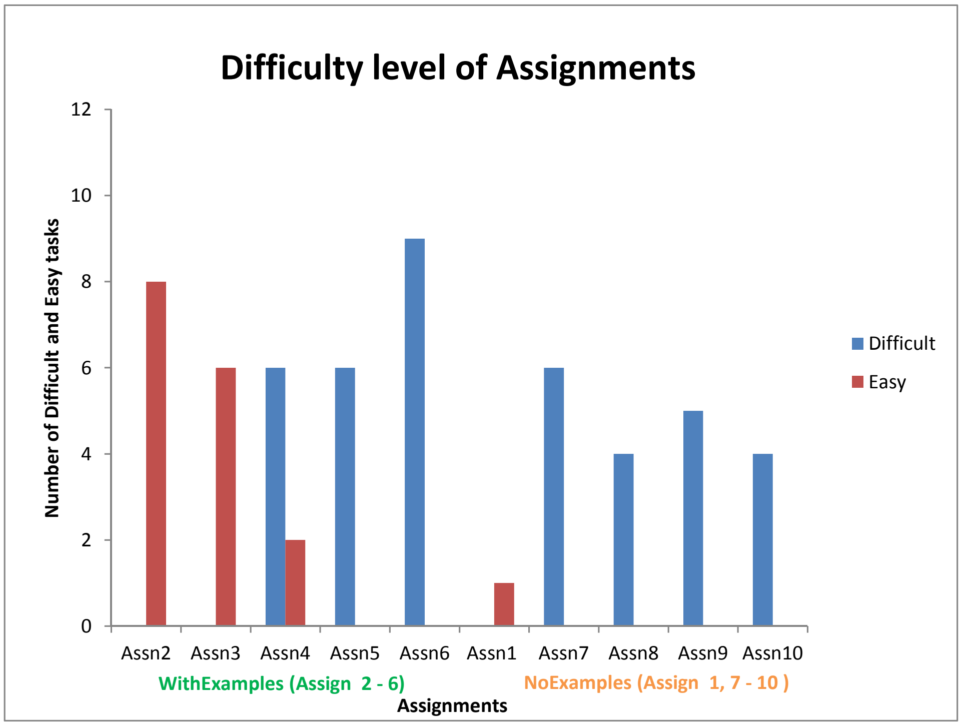


Figure 6.2: Distribution of assignment tasks based on difficulty level

These assignment groups are formed on the basis of 2 criteria (1) *Total number of worked-out examples:* ERS has many more worked-out examples on the LUs that meet learning objectives of tasks in assignments belonging to Group II (Group WithExamples), as opposed to group I (Group NoExamples) (2) *Total number of difficult tasks that the assignments consist of* : a task is deemed "D" (for "Difficult") or "E" (for "Easy") by algorithm MGREPD of the knowledge customization module of ERS described in chapter 4 (Algorithm 5). It may seem that tasks from assignments in Group NoExamples are more difficult than tasks from assignments in group WithExamples because they are assigned in the latter part of the course and consist of LUs that possibly are of higher difficult level. But experiments show that there is no such bias in the 2 groups and the total number of difficult tasks in the two groups is comparable. The graph in figure 6.2 shows that the number of difficult tasks in Group NoExamples is 19, whereas group WithExamples has 21 difficult tasks.

In order to answer research question RQ2 (section 3.3), we claim that students who follow ERS's recommendation of studying the most relevant examples suggested for an assigned task, before attempting the task, will succeed in the task with high marks. In order to validate our claim, we compare and analyze average student marks in each assignment and compute the improvement in student learning when they use the suggested worked-out examples.

We observe through experiments that Group II (WithExamples) outperforms GroupI (NoExamples) by 26% in the average obtained by students. Figure 6.3 illustrates that the class average for assignments in group WithExamples is 89%, whereas the average for assignments in group NoExamples is just a 73%. Figure 6.4 shows that the average student marks for all assignments in group WithExamples is higher than 85% for four out of five assignments in the group. On the other hand, average marks for assignments in Group NoExamples is between 65% and 75%. We also compute the improvement in learning for ten students in the course as 17% by using assignment marks of these individual students for groups NoExamples and WithExamples as shown in equation 6.1 (S = total number of students). A similar formula has been used in an existing research developed for a web application programming course (Goreva et al., 2007). It is worth mentioning here that experiments to compute improvement require individual student marks. This required by REB (Research Ethics Board) of the University of Windsor to seek student consent to participate in this study. Out of 35 students registered in the fall term, only ten consented to participate and therefore we computed the improvement factor for these ten students.

$$improvement = \frac{\sum_{s=1}^{S} Avg\_WithExamples - Avg\_NoExamples}{S} \tag{6.1}$$

6.1.1 Statistical significance

A two-sample t-test is performed to find the statistical significance of the above results for using the average scores for Scenario 1. There are 2 samples for scenario 1: sample 1 is for group WithExamples and sample 2 is for group NoExamples. The following steps are taken to accomplish this:

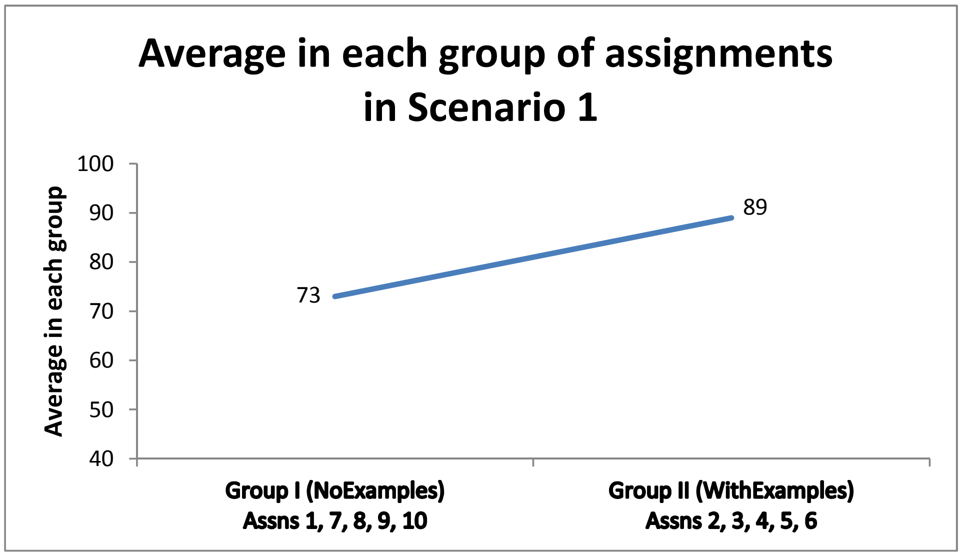


Figure 6.3: Average marks for groups NoExamples and WithExamples in Scenario 1

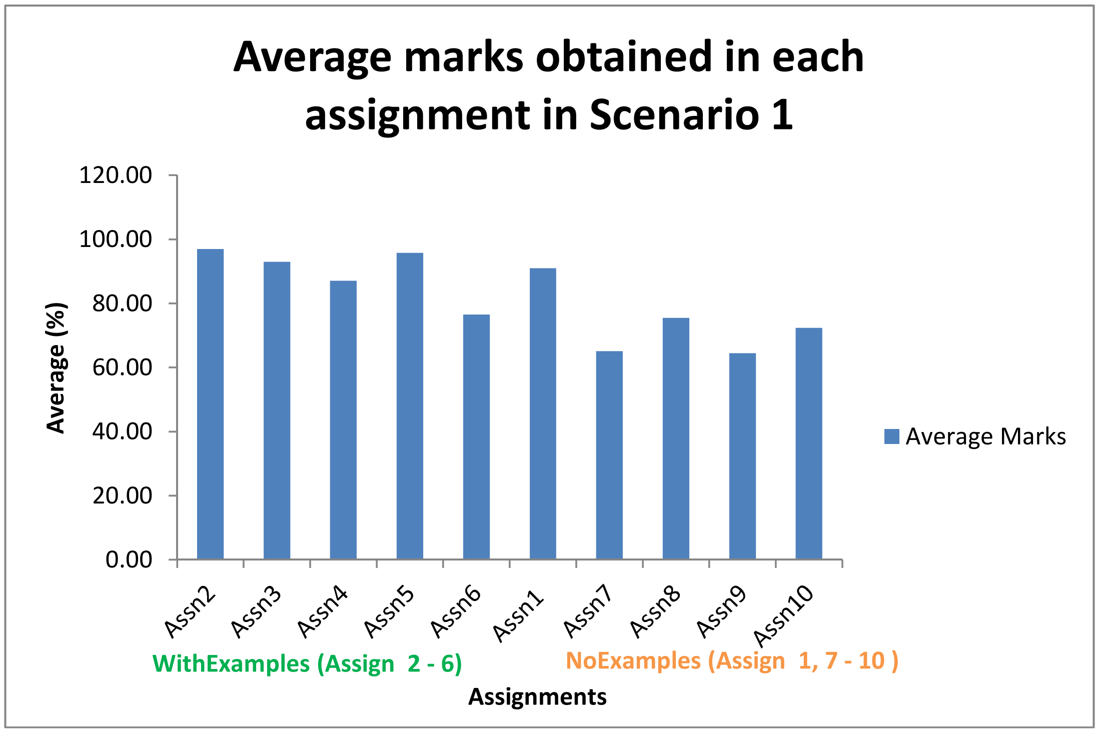


Figure 6.4: Average obtained in each individual assignment

1. Identify the null hypothesis and alternative hypothesis. If $\mu_1$and μ_2 are the averages of the two samples used, then the null hypothesis is that $\mu_1 = \mu_2$. The alternate hypothesis for this scenario is chosen to be $\mu_1 > \mu_2$, meaning that the average of students for tasks in the group WithExamples is higher than tasks in group NoExamples.

2. Establish the level of significance (α value). This indicates the probability that the difference in the average values of the 2 samples is due to chance (or not). We choose α to be 0.01, indicating our belief that there is less than 0.01 probability that $\mu_1 > \mu_2$ by chance.

3. Calculate the mean and standard deviation for both the samples. As shown in section 6.1, $\mu_1 = 89$ and $\mu_2 = 73$.

4. Use paired t-test to caculate the t-value: we choose to use a paired t-test because both samples have the same participants being tested for two different groups of assignments. The t-value calculated for the two samples in scenario1 is found to be 3.41. Comparing this value with t-table values, we find that probability that the samples are different by chance is only 0.001. This proves that the results for this scenario are statistically significant.

6.2 Scenario 2: Two different group of students doing the same set of assignments using two different approaches (no ERS and with ERS)

This scenario, as shown in figure 6.5, uses student data from 2 different semesters (Winter 2015 and Fall 2015) of students registered in Programming C course. Students from both terms were assigned the same tasks from assignments 2, 3, 4 ,5 and 6 from group WithExamples. Students from Winter 2015 were not required to use ERS (and were not suggested worked-out examples for any task by ERS), but they were informed that worked-out examples exist and they can use them if they choose to. Students from

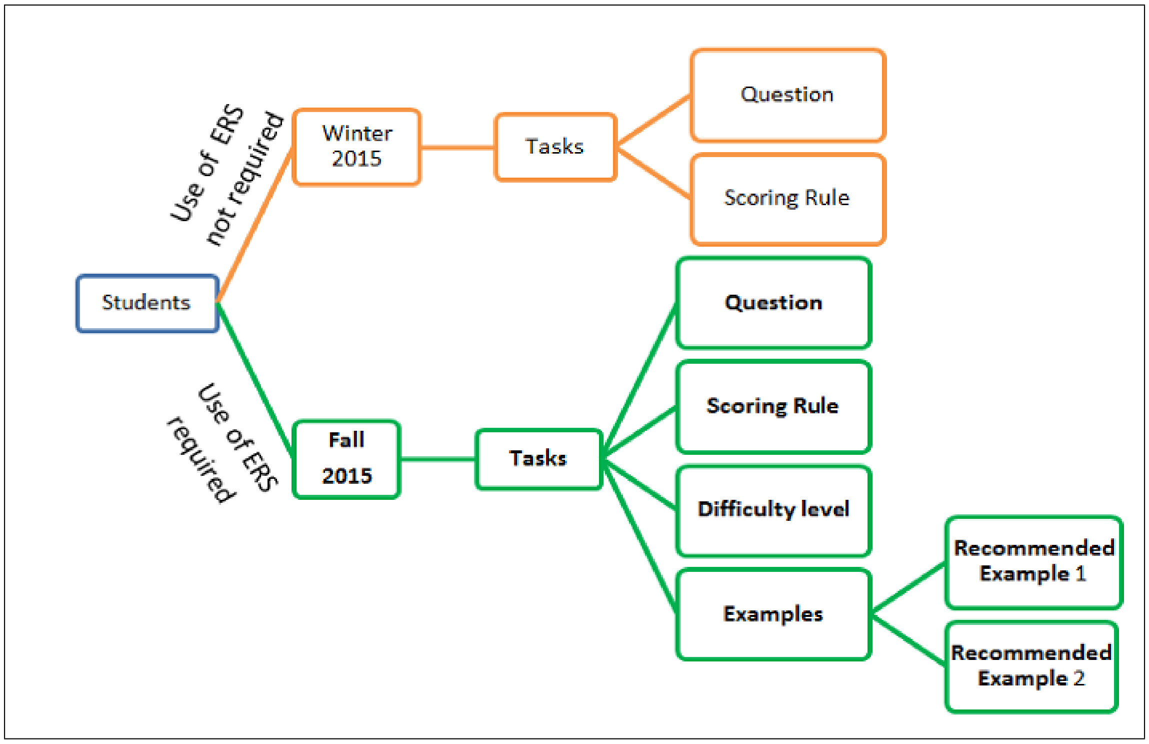


Figure 6.5: Groups of Students and Assignments from group WithExamples used in evaluation of ERS (Scenario 2)

Fall 2015 were required to use ERS for these five assignments (and were suggested two worked-out examples for each task assigned to them).

The average student performance for Fall 2015 for this set of assignments is measured as 89% and is higher than 83% measured for students of Winter 2015, as shown in the graph in figure 6.6. This is an indication that scenario 2, similar to scenario 1, follows the same trend that students have a higher likelihood of getting high marks in tasks if they study the worked-out examples recommended by ERS. This scenario allows us to validate research question RQ2 (section 3.3) further, that worked-out examples recommended by ERS contribute in meeting the overall goal of improving student learning.

6.2.1 Statistical significance

Similar to scenario 1, a two-sample t-test is performed to find the statistical significance of the above results for using the average scores for scenario 2. The results for this scenario

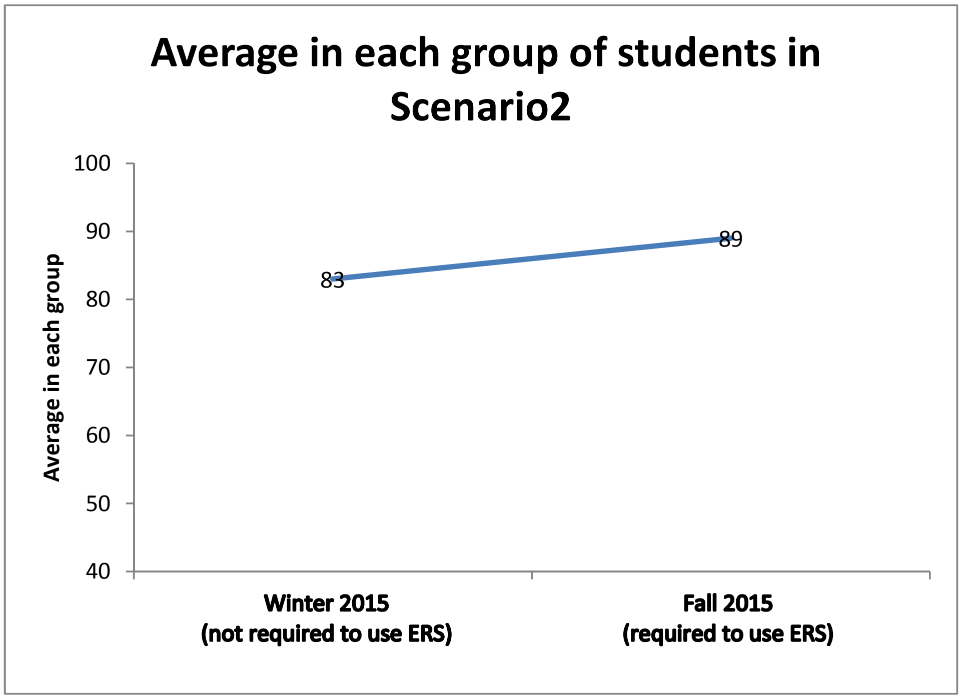


Figure 6.6: Average marks for groups Winter 2015 and Fall 2015 in Scenario 2

are not statistically significant, since the probability that the two averages are different is very high (15%).

6.3 Chapter 6 Overview

This chapter evaluates the educational impact of ERS as a tutor. Experiments with student marks for two controlled scenarios illustrate that students who study worked-out examples suggested by ERS have a higher likelihood of completing the course with high marks and thereby improve learning. Although the dataset used is small, the results are promising and cannot go unnoticed. Increasing the dataset and incorporating external factors such as students prior knowledge on the subject matter in future can ascertain the educational impact with a much higher confidence. Student performance in existing EBL-based ITS (table 2.7) is evaluated based on system usage. For example, NavEx (Yudelson & Brusilovsky, 2005) counts the frequency of usage of examples for each student as a measure of his/her learning, and PADS (Li & Chen, 2009) uses the time spent by students on given tasks as its measure of learning. ERS measures student learning by objectively comparing student marks in assigned tasks in different scenarios that are designed to evaluate ERS in its role of an educator.

Chapter 7

Evaluation of ERS's student and domain features using predictive mining

This chapter evaluates ERS's student and domain model features using method EM3 described in section 3.5.1. A key component of ERS is to design a model that can accurately predict student knowledge on its domain LUs. This chapter uses data mining to predict student performance in assigned tasks as a measure of evaluating student and domain model features of an ITS.

7.1 Introduction

For online learning environments, in general, predicting student performance is a function of two complex and dynamic factors: (1) student learning behavior (e.g. time spent on a given resource) and (2) their current knowledge in the domain (e.g. marks scored in a task). Learning behavior is captured from student interaction with the ITS and is stored in the form of web logs. Student knowledge in any ITS domain is represented by the marks they score in tasks or tests and is stored in a specific component of the ITS called student model. In order to build an accurate prediction model, this raw data from student model and web logs must be engineered carefully and transformed into meaningful features. We propose to predict student performance by using features that measure student knowledge objectively and are better informed about assigned tasks. In order to accomplish this, ERS is designed with a fine-grained domain and student model. As described in section 2.2.1, granularity refers to the level of detail that a domain model component is represented with. ERS's domain model is designed as a fine-grained model, in which its domain consists of worked-out examples and task solutions, which are further subdivided into indivisible learning units (LUs) (e.g. domain D1 consists of worked-out example E1, which consists of LUs {datatype, printf}). ERS's student model stores student knowledge on these learning units (LU) (e.g. student $s1$ scores 4 out of 6 in LU4 ('Simple Arithmetic expressions')) and student learning behavior (e.g. number of visits to a worked-out example). Pardos et al. (2007) have also proven in their study that finer the granularity, more accurate is the prediction of student performance. We define the hypothesis and the rationale behind our prediction model next.

Hypothesis H for predicting student performance :

Success of students is predicted with a high value of accuracy if the features used for prediction are better informed about the assigned tasks (e.g. difficulty level of the task) and are measured objectively (e.g. student's average marks for a suggested worked-out example).

Rationale:

Raw data, usually, is not in a form suitable for prediction algorithms but deriving or constructing features from it enables these algorithms to learn (Domingos, 2012). We claim that if the features used in a prediction algorithm are task-oriented, objective and student-centric, then the algorithm learns better and predicts student success with a high value of accuracy.

7.2 Related Works on Predicting Student Performance

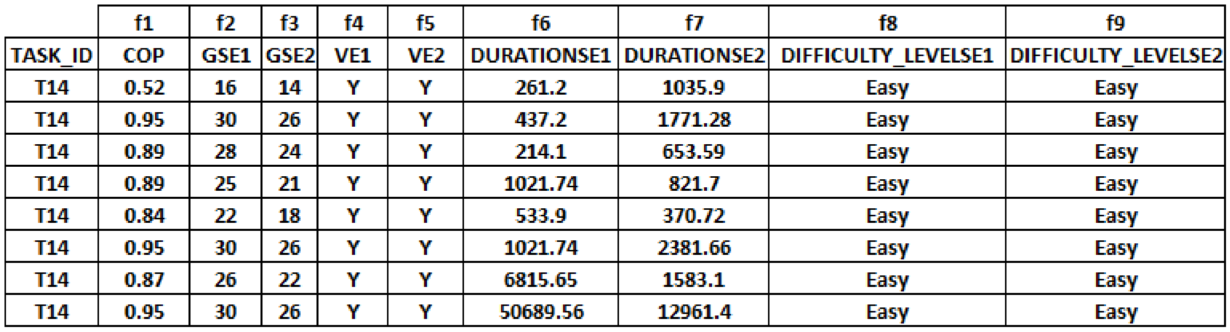
Predicting academic performance of students has been a challenging problem for intelligent tutoring systems. None of the existing EBL-based ITS such as NavEx (Yudelson &

Brusilovsky, 2005), PADS (Li & Chen, 2009) and others (section 2.2.2) attempt to predict student performance. There exist non-EBL-based ITS that predict student performance but they either lack in use of state-of-the-art techniques or in selection of appropriate features for prediction. Minaei-Bidgoli et al., (2003) in their study of an ITS called LON-CAPA use web-log features to predict student performance in the final exam with a prediction accuracy of 87%. Features they use describe students learning behavior such as total number of correct answers by a student, student's success at the first attempt for each task, total number of attempts made in each task to get the correct answer and total time spent on each task until solved. None of these features measure student knowledge on LUs or tasks objectively are not good indicators of student performance. Another drawback of using features for prediction solely based on student interaction with an ITS is that it may mislead the prediction model (e.g. time to complete a task will be recorded by the ITS as very high in situations where student starts working on a task but does not logout after completion. In this scenario, "time to complete the task" will be recorded as high - an (incorrect) indication to the ITS that the student is struggling with the subject). Thai-Nghe et al., (2010) adapt techniques used in recommender systems to predict student performance. A recommender system (RS), in general, is an information filtering method which links users to items (e.g. Netflix recommends movies to its users) (Mabroukeh, 2010). If there are m users and n items, a RS will arrange them as an m * n matrix M, such that $M(i, j) = 1$, if user i likes item j; 0 otherwise. Typically, many entries in M are missing (e.g. when there is a new user who hasn't liked any item yet) and are predicted by RS based on the information of the other existing users using techniques such as clustering and collaborative filtering (Markov & Larose, 2007). The authors (Thai-Nghe et al., 2010) map students to users and tasks to items in order to assign a rating to the student-task pairs. A limitation of mapping the problem of predicting student performance to recommender systems is that prediction in RS is based on web usage patterns of users, whereas student models in ITS are more concerned with web content usage (e.g. student's knowledge on a given task). Shen et al., (2010) propose a system very similar to that of Minaei-Bidgoli et al., (2003) but they use a finer level of granularity to extract features specific to each step of a task. Each task is divided into several steps by experts and student's knowledge on each step is predicted. Their system too, like many others, relies solely on logged features extracted from student's interaction with it, instead of student's current knowledge on the domain. McCuaig and Baldwin(2012) use a different approach to predict student success in a web-based course. They do not provide or use any formal assessments such as quiz or exam grades. They use features such as student's mean self-confidence for the overall semester (by asking students to fill a questionnaire each week), total number of active days spent using their system and the average time spent in doing problem sets (students attempt an ungraded problem set each week) to predict student success using decision trees. The low value of prediction accuracy for their method (70%) can be attributed to the subjective nature of features such as confidence level (entered by the student) and the lack of task-oriented features. There is no direct measurement of student knowledge either, since the problem sets they use are ungraded. As indicated above, all the existing methods that predict student performance suffer from a major limitation - improper feature selection. This thesis overcomes the above limitations to predict student performance by deriving features that are objective and well-informed about the tasks assigned to students and also well-informed about the resources (such as worked-out examples) that ERS recommends for each task. Examples of such features include students performance on a task and its LUs, their knowledge on the worked-out examples that assist in the task, difficulty level of these examples and so on.

7.3 Proposed Methodology to predict student performance (PSP)

In order to validate hypothesis H (defined in section 7.1), we map the problem of PSP to the data mining problem of prediction using task-oriented and objective features. The student and domain datasets used for prediction are described next followed by the algorithms used for feature extraction.

Dataset used for PSP



	f1	f2	f3	f4	f5	f6	f7	f8	f9
TASK_ID	COP	GSE1	GSE2	VE1	VE2	DURATIONSE1	DURATIONSE2	DIFFICULTY_LEVELSE1	DIFFICULTY_LEVELSE2
T14	0.52	16	14	Y	Y	261.2	1035.9	Easy	Easy
T14	0.95	30	26	Y	Y	437.2	1771.28	Easy	Easy
T14	0.89	28	24	Y	Y	214.1	653.59	Easy	Easy
T14	0.89	25	21	Y	Y	1021.74	821.7	Easy	Easy
T14	0.84	22	18	Y	Y	533.9	370.72	Easy	Easy
T14	0.95	30	26	Y	Y	1021.74	2381.66	Easy	Easy
T14	0.87	26	22	Y	Y	6815.65	1583.1	Easy	Easy
T14	0.95	30	26	Y	Y	50689.56	12961.4	Easy	Easy

Figure 7.1: Partial dataset prepared for PSP - 8 rows shown correspond to marks scored by 8 students in task T14

- Original dataset S_r: PSP uses a student dataset of ten students who were registered in Fall 2015 in a course on Programming in C for beginners offered by the School of Computer Science, University of Windsor. Although 35 students registered in Fall 2015, only 10 of them consented to participate in this study. There were 13 tasks that students were assigned throughout the term. This generated a dataset with 130 instances or records (one for each student per task).

- Simulated dataset S_s: Using the original dataset S_r, additional instances were generated to create a simulated dataset with 520 instances.

7.3.1 Features selected for PSP and their Extraction algorithms

Nine features are carefully chosen for each student Sid and each task Tid in ERS, in order to meet the objectives of PSP model. We discard the student Ids since they are not required for prediction. Figure 7.1 shows the first 8 rows of the dataset prepared for PSP. Features include {feature f1: Sid's current overall performance (COP), features f2 and f3: grades in the worked-out examples suggested by ERS (GSE1 and GSE2) (we use the top 2 suggested examples in the current research), features f4 and f5: whether student has visited the suggested examples (VE1 and VE2), features f6 and f7: time spent on the suggested examples (DurationSE1 and DurationSE2), features f8 and f9: difficulty level of the suggested examples (Difficulty_LevelSE1 and Difficulty_LevelSE2) .

Features and algorithms used to derive them are listed next.

1. Feature f1: COP (Current Overall Performance) : Student model of each student in ERS stores the marks they achieve in each learning unit (LU) in ERS's domain. COP is derived by finding the average performance of a student in all the LUs learnt so far.

2. Feature f2 : GSE1 (Grade in the Suggested Example 1): Algorithm 10 (called GSE) explains the steps required to derive this feature. ERS students are graded on assigned tasks and these grades are distributed among the tasks's LUs. Features 2 and 3 compute the grades students score in worked-out examples using the grades of their LUs. GSE takes as input (1) the current task Tid's represented as a binary vector of n LUs, (2) each worked-out example also represented as a binary vector of n LUs (GSE stores these vectors as a binary matrix of size m * n, were m is the total number of worked-out examples) and (3) student Sid's current scores in each LU. Step 1 of GSE applies MGREPD (algorithm 5) to find Tid's k nearest neighbors (called as $List_{relevant}$) using Jaccard's Coefficient (JC) as its similarity function. Step 2 of GSE fetches the LUs of each of the k worked-out examples in $List_{relevant}$ and stores them in E1_LU. For example, if $List_{relevant}$ for a given task TP1 is [EP2, EP4], then E1_LU = [LU1, LU2, LU5], assuming that EP2 consists of these 3 LUs. Step 3 computes the sum of grades student Sid scores in worked-out example 1 stored in $List_{relevant}$. For example, GSE1 for student Sid = grade[LU1] + grade[LU2 + grade[LU5].

3. Feature f3: GSE 2 (Grade in the Suggested Example 2) is extracted using the same algorithm GSE and steps used for feature 2. Using the same $List_{relevant}$ in step 2, E2_LU = [LU3, LU5] and GSE2 in step 3 computed for student Sid = grade[LU3] + grade[LU5].

4. Feature f4: VE1 (Visited Example 1) is extracted from the weblogs generated by student interaction with ERS for worked-out example1 in $List_{relevant}$. It indicates whether a student has visited the suggested example or not.

5. Feature f5: VE1 (Visited Example 2) is extracted from the weblogs generated by student interaction with ERS for worked-out example 2 in $List_{relevant}$. It indicates whether a student has visited the suggested example or not.

6. Feature f6 : DurationSE1 is derived by finding the sum of the time spent (in seconds) on the worked-out example E1 in $List_{relevant}$. Each time a student visits a page, its url, date and time is recorded in the web log and stored as a row in table Page of the relational model. Then the time spent by student *Sid* each time he/she visits E1 is extracted using a PL/SQL function called timespent_on_example_by_student, shown in figure 6.2. This function searches for the name of the worked-out example in table Page's url (e.g. if student *Sid* has browsed example E101 at 2 different times and his data is stored 2 rows in table Page as <7821, *Sid* , 1843, '/example/E101'> and <7829, *Sid* , 43, '/example/E101'>, then function timespent_on_example_by_student will search for rows in Page for *Sid* with 'E101' in their url and add up the time_spent values for those rows.

7. Feature f7 : DurationSE2 is derived in the same way as feature 6 for worked-out example E2 in $List_{relevant}$.

8. Feature f8: Difficulty_LevelSE1 (Difficulty level of suggested example 1) is derived using algorithm findDL (algorithm 4) in section 4.3 that assigns a difficulty level of E (for easy) or D (for difficult) to worked-out example 1 in $List_{relevant}$.

9. Feature f9: Difficulty_LevelSE2 (Difficulty level of suggested example 2) is derived using algorithm findDL (algorithm 4) in section 4.3.4 that assigns a difficulty level of E (for easy) or D (for difficult) to worked-out example 2 in $List_{relevant}$ (similar to feature 8).

10. Class label attribute SUCCESS_IN_TASK: SUCCESS_IN_TASK is assigned a yes, if a students succeeds in the assigned task and no otherwise. It is assumed that a student succeeds in a given task if he/she has achieved a grade of 75 or higher in it.

Algorithm 10 GSE (Grade in the Suggested Example)

```
Input: 1. task $Tid$ as a binary vector of size n, 1/0 indicating
    presence/absence of LU (where n = number of LUs in ERS)
2.$LU\_EX$: binary matrix of size m examples * n LUs:
         each row in LU_EX represents a worked-out example
         1/0 indicates presence/absence of LU in that row
3. $grades_{Sid}$ : vector of size n that holds student $Sid$'s current scores
        in each LU
Output: grade in suggested examples 1 and 2: $GSE1$ and $GSE2$
Other variables: $List_{relevant}$:k examples most relevant to task $Tid$
Method:
***begin of GSE
1. Find $Tid$'s nearest 2 neighbors using MGREPD (algorithm 5)(assuming k = 2)
  1.1. Compute the similarity between task $Tid$ and each row $i$
    in $LU\_EX$ using Jaccard's coefficient $JC$4.5
  1.2. Sort the $JC$ values computed in step 1.1 in ascending order
    and store the corresponding examples of top k (=2) of them in
       $List_{relevant}$(lets call them row_E1 and row_E2)
2. Find row_E1's LUs from ERS's domain model and store as $E1\_LU$
    $E1\_LU$ =LUs of worked-out example row_E1
3. Find $Sid$ 's grade in worked-out example row_E1
    If n = number of LUs,
    $GSE1 = \sum_{i=1}^{n} grade_{Sid}[E1\_LU[i]], \forall E1\_LU[i] = 1$
4. Repeat steps 2 and 3 for worked-out example row_E2 to find $GSE2$
***end of GSE
```

```
CREATE OR REPLACE FUNCTION TIMESPENT_ON_EXAMPLE (EXAMPLE EXAMPLES.EXAMPLE_ID%TYPE,
                                          STUDENT PAGE_VISIT.STUDENT_PID%TYPE)
RETURN NUMBER
IS
VTIME_SPENT NUMBER;


BEGIN

     SELECT SUM(TIME_SPENT)
     INTO VTIME_SPENT
     FROM PAGE_VISIT
     WHERE PAGE_URL LIKE '%'||EXAMPLE||'%'
     AND STUDENT_PID = STUDENT;


    RETURN VTIME_SPENT;
END;
```

Figure 7.2: PL/SQL script to derive proposed features f6 and f7 listed in section 7.3.1

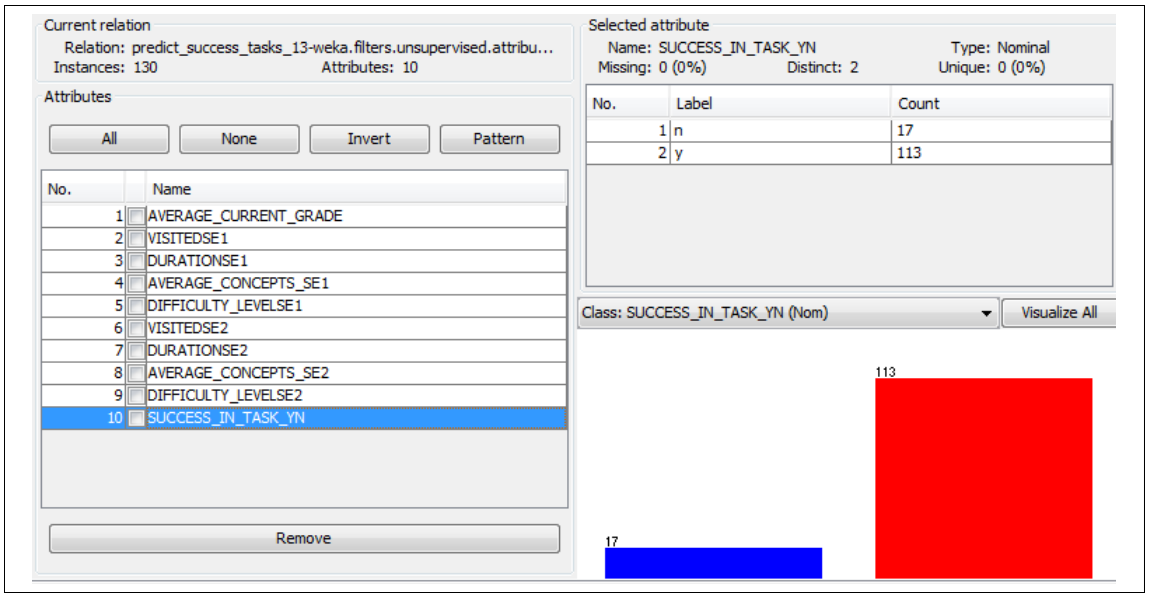
7.4 Results and analysis of predicting student performance

This section presents the results and analysis of the PSP model built using decision tree analysis applied to the novel features described in section 7.3.1. The original student dataset S_r (Section 7.3) generated for the problem of PSP is observed to have two concerns: (1) S_r, obviously is a small dataset with 130 instances (2) the class label attribute (SUCCESS_IN_TASK) is imbalanced because it contains more of class label 'yes' (113 instances) as compared to 'no' (17 instances). Although this is a good indicator that ERS is an effective tutoring system, in which students succeed in assigned tasks with high scores, it still creates an undesirable bias for classification algorithms used to predict student performance. In order to overcome these two concerns (small and imbalanced dataset), we use an existing algorithm called SMOTE (Chawla et al., 2002) to deal with such class imbalance. SMOTE, (Synthetic Minority Oversampling Technique) (Chawla et al., 2002) is an over-sampling technique used to overcome the issues of imbalanced datasets by adding volume to the minority class label, so that the instances of majority and minority class labels are equally distributed. Although the objective of SMOTE is to balance the class labels, it does so by adding more samples or records and therefore allow us to add volume to our otherwise small original dataset. SMOTE adds samples by taking 5 nearest neighbors (NN) of a minority class sample X. It finds the difference between feature vector of X and feature vectors of the NN of X, multiplies this difference by a random number between 0 and 1 and then adds the resulting value to the original value of X. Equation 7.1 represents this idea where X is the original instance/sample; X_{new} is the newly generated sample, X_{NN_i} is one of the 5 NN of sample X; δ represents a random number between 0 and 1. For example, if feature vector of $X = (6, 4)$ and one of its 5 NN is $X_{NN_i} = (4, 3)$, then a new sample generated is $X_{new} = (6, 4) + (2, 1) * 0.5 = (6, 4) + (1, 0.5) = (7, 4.5)$.

$$X_{new} = X + (X_{NN_i} - X) * \delta \tag{7.1}$$

We apply SMOTE to S_r using Weka (Hall et al., 2009), which is an open-source software that contains tools for data pre-processing, classification (including decision tree analysis), and other data mining tasks. Figure 7.3 on the following page shows the class distribution of the original dataset S_r that has 130 rows (or instances as Weka calls them), with 113 of them having a value of SUCCESS_IN_TASK=yes. Figure 7.4 on the next page provides the count distribution of the over-sampled and balanced dataset created by Weka's SMOTE algorithm. SMOTE not only balances the class labels from 11% to 97% but also increases the number of instances from 130 to 520.

Now that our dataset is balanced and has volume, we describe our prediction model and compute its performance. Given a set of data records represented as (x,y) where x is a set of features, prediction is the task of mapping the feature set x into a special feature y, where y is termed as the class label. The steps used to build a decision tree prediction model for PSP (also shown in chapter 5 section 5.3) are (1) divide data into 2 subsets: training and test (2) build the model by applying a prediction algorithm on the training dataset (3) apply the model to test dataset - their actual class labels are compared to the predicted ones to evaluate the model (4) use the model to classify unseen records and predict their y values. The proposed PSP model uses a method called 10-fold cross validation (Payam et al., 2009), in order to divide its datasets S_r and S_s into training and test so that their actual class labels can be compared with the predicted ones. Each iteration takes one of these partitions as test data and the others are used for training


Current relation

Relation: predict_success_tasks_13-weka.filters.unsupervised.attribu...
Instances: 130 Attributes: 10

Selected attribute

Name: SUCCESS_IN_TASK_YN Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

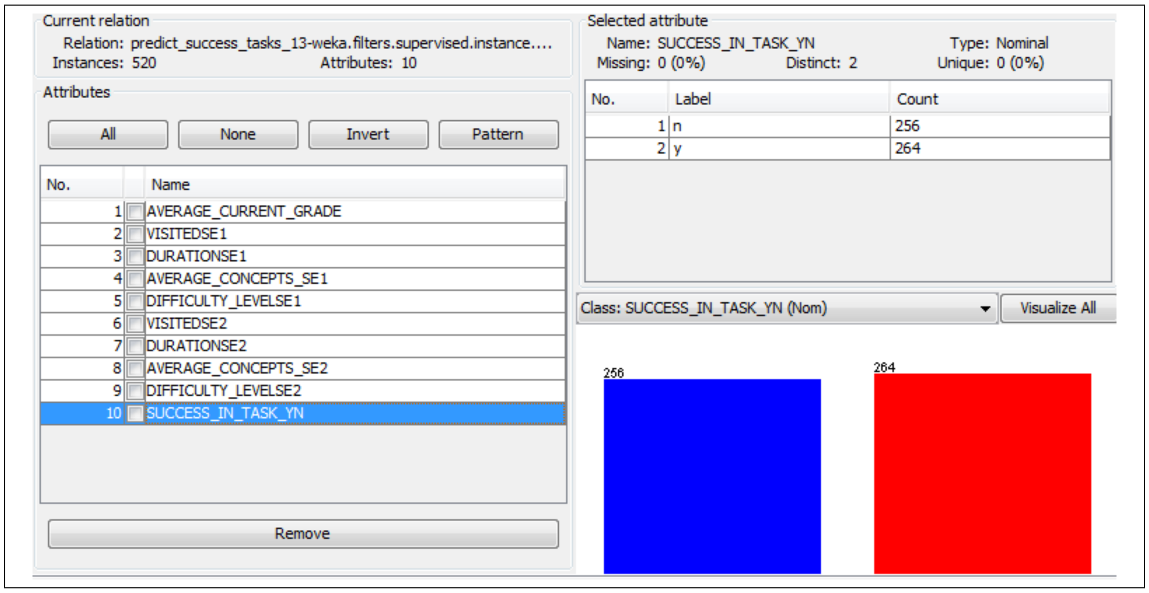
No.	Label	Count
1	n	17
2	y	113

Attributes

No.	Name
1	AVERAGE_CURRENT_GRADE
2	VISITEDSE1
3	DURATIONSE1
4	AVERAGE_CONCEPTS_SE1
5	DIFFICULTY_LEVELSE1
6	VISITEDSE2
7	DURATIONSE2
8	AVERAGE_CONCEPTS_SE2
9	DIFFICULTY_LEVELSE2
10	SUCCESS_IN_TASK_YN

Class: SUCCESS_IN_TASK_YN (Nom)

Figure 7.3: (Imbalanced) Class distribution of the original dataset S_r - class yes has 113 instances; class no has 17 instances


Current relation

Relation: predict_success_tasks_13-weka.filters.supervised.instance....
Instances: 520 Attributes: 10

Selected attribute

Name: SUCCESS_IN_TASK_YN Type: Nominal
Missing: 0 (0%) Distinct: 2 Unique: 0 (0%)

No.	Label	Count
1	n	256
2	y	264

Attributes

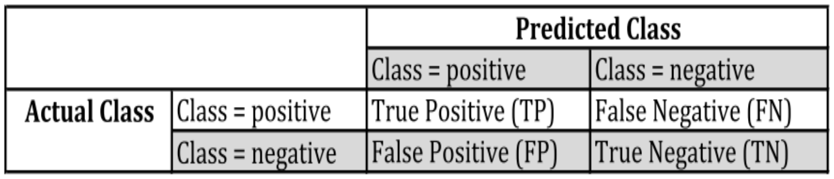
No.	Name
1	AVERAGE_CURRENT_GRADE
2	VISITEDSE1
3	DURATIONSE1
4	AVERAGE_CONCEPTS_SE1
5	DIFFICULTY_LEVELSE1
6	VISITEDSE2
7	DURATIONSE2
8	AVERAGE_CONCEPTS_SE2
9	DIFFICULTY_LEVELSE2
10	SUCCESS_IN_TASK_YN

Class: SUCCESS_IN_TASK_YN (Nom)

Figure 7.4: (Balanced) Class distribution of data after applying SMOTE - class yes has 264 instances; class no has 256 instances

the model. This is repeated 10 times, so that each partition is used for testing only once. Each time the model predicts a class for records in the test dataset, it is compared with its actual class label and a confusion matrix is generated. A confusion matrix is a table that allows visualization of the performance of a classification algorithm. It gives a count of data records or instances that are correctly or incorrectly predicted by the algorithm. In general, for a 2-class problem (such as the one used in this study), labels are termed as positive / negative; the original class labels are referred to as actual and those determined by the classification algorithm are termed as predicted. Figure 5.6 shows a generic confusion matrix and is reproduced below for convenience. In this figure, each column represents the total number of records in a predicted class, and each row represents the total number of records in an actual class. The classification algorithm then, assigns to each record or instance one of the following :

- True Positive (TP): actually positive - also predicted as positive.
- True Negative (TN): actually negative - also predicted as negative.
- False Positive (FP): actually negative - but predicted incorrectly as positive.
- False Negative (FN): actually positive - but predicted incorrectly as negative.



		Predicted Class	
		Class = positive	Class = negative
Actual Class	Class = positive	True Positive (TP)	False Negative (FN)
	Class = negative	False Positive (FP)	True Negative (TN)

Performance measures such as accuracy, recall and f_measure (Markov & Larose, 2007) that typically evaluate prediction algorithms such as decision trees can be computed using confusion matrix as shown in figure 5.7 and shown here again for convenience.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{total\ number\ of\ predictions} = \frac{(TP+TN)}{(TP+TN+FP+FN)}$$

$$precision = \frac{Number\ of\ actual\ positives\ correctly\ predicted}{number\ of\ predicted\ positives} = \frac{TP}{TP+FP}$$

$$recall = \frac{Number\ of\ actual\ positives\ correctly\ predicted}{number\ of\ actual\ positives} = \frac{TP}{TP+FN}$$

$$f_{score} = \frac{2}{\frac{1}{r}+\frac{1}{p}} = \frac{2 * r * p}{r+p}$$

Tables 7.1 and 7.2 show results of decision tree classification model applied to the original dataset S_r and to the over-sampled balanced dataset S_s. The total number of FP (false positives) and FN (false negatives) in a confusion matrix indicate erroneous results. For example, FP in table 7.1 measures the total number of students whose success is wrongly predicted by PSP to be yes (although in reality, those students do not succeed). Similarly, FN measures the total number of students who actually succeed but the model incorrectly predicts them as failures. With decision trees, the number of FP in the original dataset S_r is found to be 6 out 130 instances (4.6%) , whereas it is less than 1% (4 / 520) on the simulated dataset S_s . PSP's decision tree model, when applied to the simulated and larger dataset S_s achieves much higher values of accuracy and f_score, as compared to the original dataset, as shown in table 7.2. Both accuracy and f_score are as high as 96% when class labels are predicted using the simulated dataset S_s with 520 instances as compared to 91% and 89% for dataset S_r with 130 instances. Evidently, the reason for misclassification in S_r for the minority class (SUCCESS_IN_TASK = no) is imbalance in distribution of the two class labels (SUCCESS_IN_TASK = yes and SUCCESS_IN_TASK = no) and too few training instances of the minority class for the model to learn accurately. Such high values of performance measures such as accuracy and f_score validate our hypothesis 7.1 that a prediction model can be built with a high f_score and accuracy by selecting features that have proper knowledge on the assigned tasks and those that measure student knowledge objectively.

	Predicted = Yes	Predicted = no
Actual = Yes	112 (TP)	1 (FN)
Actual = No	6 (FP)	11 (TN)

(a) Confusion matrix generated for dataset S_r

	Predicted = Yes	Predicted = no
Actual = Yes	245 (TP)	19 (FN)
Actual = No	4 (FP)	252 (TN)

(b) Confusion matrix generated for dataset S_s

Table 7.1: Confusion matrix generated by decision tree model with original dataset S_r (130 instances) and (original+simulated = 520 instances) dataset S_s

	Dataset S_r (size=130)	Dataset S_s (size=520)
Accuracy	91 %	96 %
Recall	90 %	96 %
Precision	91 %	96 %
F_Score	89 %	96 %

Table 7.2: Performance measures: Decision tree using datasets S_r and S_s

Figure 7.5 shows a decision tree generated from S_s. As an example, one of the rules generated by this tree is:

Rule 1:

```
if the average current grade of a student is > 86%

    success = yes

else if the average current grade of a student is > 81%


    if average grade in  LUs of suggested example 2(SE2) > 49%

        success = yes

    else if time spent of SE2 < 0.009 (9 minutes)


        if average grade in  LUs of suggested example 1(SE1) > 68%

            success = yes
        else success = no
```

These rules indicate that if a student s has not performed well in the learning units of suggested examples and has not spent enough time on them, then s will not be able to succeed in the task. Even if student s's current performance is average or above average, the rules indicate that the student still has to achieve a certain level of grades in the LUs of the suggested worked-out examples, which yet again asserts the importance of students using and understanding of the worked-out examples suggested for each task by ERS.

7.5 Chapter 7 overview

The main objective of building the PSP model is to accurately predict student success for assigned tasks in a fine-grained ITS system by proposing features that are focused on the task's resources such as similar worked-out examples suggested by the ITS and student's knowledge on these resources. Our proposed method is able to extract meaningful task-based features and implement them to predict student performance using decision tree prediction method with accuracy and f_score values as high as 96%. Existing ITS that predict student performance surveyed in section 7.2 have much lower values of accuracy, which can be attributed to their improper selection of student and domain model features. This validates our hypothesis that student performance is predicted with a high value of accuracy if features used for prediction are well-informed about the assigned tasks and are measured objectively. The rules generated by decision trees allow us to analyze and take informed decisions on ERS's future students.

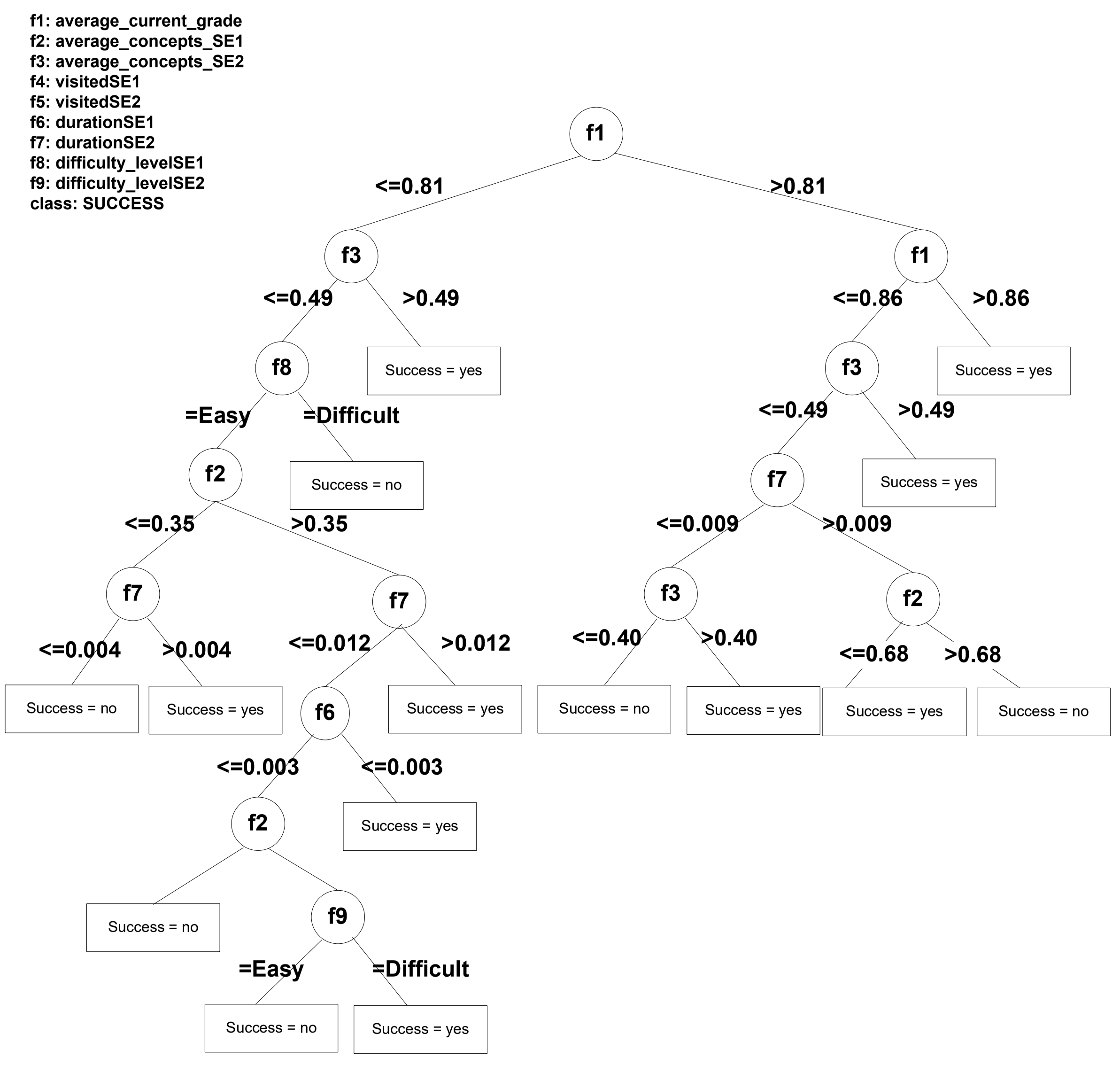


Figure 7.5: Decision tree generated by Weka's J48 for student dataset S_s with 520 instances

Chapter 8

Conclusions, Limitations and Future Work

This chapter presents a summary of this thesis, its limitations and some future directions.

8.1 Thesis Conclusions

It is an accepted fact that it is not feasible for teachers of traditional classroom teaching to offer one-on-one tutoring to their students. Programming courses, especially those taught to beginners, require a lot of resources to enable students learn the subject. An intelligent tutoring system (ITS) is an answer to these problems by providing a domain model that manages all ITS resources and a student model that enables personalized tutoring. ERS is an ITS that uses example-based learning (EBL) to assist students by providing them with selected worked-out examples focused towards the tasks they are assigned in an effort to improve their learning. It also helps them prepare for examinations by providing them with a highly organized repository of worked-out examples. In order to validate the effectiveness of ERS, a working tutoring system that teaches C programming to beginners at the University of Windsor was designed and successfully implemented B.1. An evaluation of ERS as a tutor was conducted on students registered for a course on Programming in C for beginners in Winter 2015 and Fall 2015 semesters. Results of this evaluation (chapter 6) indicate that ERS is effective in meeting its prime goal of improving learning because students scored significantly higher in those tasks for which ERS was used.

This section now presents the significant contributions made by this research in context of the research questions described in Chapter 3 (section 3.3) and repeated below.

Research Question RQ1 Is it possible to define simple and efficient knowledge extraction methods that not only extract the LUs from the domain's worked-out examples and task solutions correctly but are also extendable to new domains without the need of highly trained experts.

The answer to this question is a yes, as demonstrated by this thesis. An ITS that teaches programming using worked-out examples must be capable of representing these examples conveniently so that they can be compared and analyzed with other examples and other related resources such as tasks. This step of extracting knowledge from worked-out examples and representing them conveniently can be mapped to the data pre-processing step of a data mining problem. In general, pre-processing is used to transform raw input data into appropriate formats for subsequent mining (Pang-Ning et al., 2005). The knowledge extraction module of ERS (chapter 4 section 4.1) develops and implements an algorithm KERE that extracts learning units (LU) from each worked-out example and task solution in the domain of ERS using regular expression analysis and transforms them into vectors of asymmetric binary values, where 1 indicates the presence of a LU. Using regular expressions mitigates the need for highly trained experts with complex knowledge on parser and syntax trees (as is required by existing EBL-based ITS). Chapter 5 evaluates KERE for its efficiency, extendibility to other domains and correctness. For example, its extendibility is demonstrated by applying it to another domain D2 (Programming in functional language Miranda). A working website is also developed and implemented to test this property Appendix B.2).

Research Question RQ2 What impact does a focused and concise list of worked-out examples have on students performance and learning?

There are two parts to this question (1) generate relevant worked-out examples for an assigned task (2) validate the impact of using such examples on student learning. One of the challenges that students face in online courses is abundance of information, which includes both relevant and irrelevant information (relevant to the assigned task). The knowledge customization module of ERS in chapter 4 (4.3) defines an algorithm called MGREPD that is built to provide students with a list of worked-out examples that consist of LUs similar to an assigned task. This algorithm applies a data mining algorithm called k-nn (k nearest neighbors) to generate such a list of examples. The core of k-nn lies in the similarity function used for comparing worked-out examples and task solutions, which is dictated by how they are represented. The vector representation of each such example and task solution by the pre-processing module of ERS enables it to experiment with several different but appropriate similarity functions such as cosine similarity, Jaccard's coefficient and hamming distance. Chapter 5 section 5.3 demonstrates that, for both domains D1 and D2, using Jaccard's coefficient generates the closest worked-out examples for a task in terms of the mining model k-nn's accuracy and f_score. Existing EBL-based systems suggest examples independent of the task students are assigned. The onus lies on the students to fetch for the examples that will help them succeed in a task or test.

To answer the second part of this research question, ERS's educational impact as a tutor is validated in chapter 6. Experiments using two different scenarios demonstrate that students who are required to use ERS and its recommended worked-out examples before attempting assigned tasks achieve significantly high marks in those tasks. This answers RQ2 that a focused list of examples has a high impact on student learning.

Research Question RQ3 Can inclusion of global relevance of LUs in a set of worked-out examples impact cluster formation?

This question is very specific to the data mining problem of clustering and the asnwer is yes. ERS realizes that its repository of 250 worked-out examples for domain D1 and 101 such examples for domain D2 must be presented to students in a highly organized way so that students can also use them at times other than attempting tasks (e.g. when preparing for examinations). ERS maps the problem of organizing its repository of examples based on their LUs into a data mining problem of clustering binary asymmetric data. Knowledge organization module of ERS uses standard k-means clustering algorithm with significant contribution to computing centroids of binary asymmetric attributes. K-means typically computes the mode of corresponding binary attributes in a cluster to find its cluster representatives. This is of concern to ERS data since the LUs are not guaranteed to be uniformly distributed among all worked-out examples, and therefore ERS uses global relevance of LUs when computing cluster representatives. For example, a cluster with 5 data records and 4 LUs given as [0100, 1011, 1010, 0011, 0011] has 0011 as its representative (mode of each LU). If the total number of worked-out examples of LU1 in ERS is just 2 and this cluster has both of them, even then LU1 is not chosen to represent this cluster. Including such global relevance of LUs by ERS's modified k-means algorithm called KOM16 results in a much higher value of Dunn's internal validity index, indicating more compact and well-separated clusters. Chapter 5 section 5.4 presents a comparative analysis of KOM16 with standard k-means. This answers RQ3 that cluster formation is impacted by considering LUs in a global context. Very few existing EBL-based ITS organize their examples lesson-wise. KOM16 organizes them based on related LUs.

Research Question RQ4 Is there a way to clearly define and integrate different modules of an ITS and if so, how?

Answering questions RQ1, RQ2 and RQ3 successfully allows us to establish a framework that clearly defines and integrates the basic components (knowledge extraction (KE), organization (KO) and customization (KC)) required by any ITS system. A clear separation of these components enforces modularity since each component can now be treated as an independent unit and enables to create a layer of abstraction between the module that extracts LUs from task solutions and worked-out examples (KE) and other modules (KC and KO), thereby making ERS domain-independent, once KE is executed.

None of the EBL-based ITS in this thesis's literature survey are capable of presenting such a clear and modular framework. This is a significant contribution for the ITS research community.

Research Question RQ5 How does selection of features from domain and student models of an ITS impact the accuracy of predicting student performance?

In this thesis, the proposed ERS is evaluated in many different ways, unlike any of the existing EBL-based ITS. In addition to evaluating ERS's individual modules (chapter 5) and its impact as a tutor (chapter 6), we also evaluate the main features of domain and student model. In an attempt to answering RQ5, we select or derive specific domain and student model features that are significant for ERS. These features are validated by building a decision tree model that predicts student performance using these proposed features and verifying its accuracy and f_score. Predicting student performance is a crucial part of every ITS but none of the EBL-based ITS described in chapter 2 2.2.2 focus on this aspect. Chapter 7 describes the process of carefully engineering the proposed features and applying decision tree analysis to predict student performance with a 96% accuracy.

8.2 Limitations of ERS

Limitations of algorithms used in modules KE, KC and KO of ERS are listed below.

1. Manual construction of regular expressions (RE) by experts: A novel contribution of ERS to the ITS community is its knowledge extraction algorithm (KERE). The domain model for KERE requires ERS experts to provide the regular expressions of each LU in the domain of ERS. Although the use of regular expressions has many advantages as highlighted in all previous chapters, a limitation is that experts have to construct these regular expressions manually.

2. No validation on the correctness of the given input: Regular expressions have a limitation that do not validate the correctness of the input string - they just identify patterns in given input strings. For example, KERE will identify LU9 (given as print type 2 in 3.1) in a given input string s = printf("%d%d", i), even though s is an incorrect statement in C because the number of variables is different than the number of format specifiers in it. Similarly, an expression given as (a+b)) has an unbalanced right parenthesis but KERE will not be able to detect any error in it - it will still identify this expression as LU4 (Arithmetic Expression - Simple). ERS mitigates this limitation by making a simple assumption that an expert is always

right! Therefore, ERS It assumes that the task solutions and worked-out examples provided by experts are always correct and do not need to be validated.

3. ERS's domain model limitations for domain D1 (C Programming): a valid function prototype in ERS must have variables in it, although C allows to define function prototypes without variable names. For example, ERS's KERE does not identify LU25 in "int fun (int);", although it is a valid C syntax.

4. ERS's domain model limitations for domain D2 (Miranda Programming): there are 2 constraints with the RE are constructed for this domain: (1) an IF statement must have an OTHERWISE in it for KERE to identify it as LU46. (2) Miranda has 2 different meanings for symbol <, one is the relational operator <, the other is as a membership operator in list comprehension (<-). KERE extracts LU43 (relational operator) in every instance that it identifies LU47 (List comprehension).

5. Choice of k in KC and KO components of ERS: Both MGREPD and KOM16 are sensitive to choice of k.

6. Sensitivity to imbalanced distribution of worked-out examples: Formation of clusters in KOM16 is sensitive to the number of worked-out examples that contain an LU. For example, ERS currently has 4 worked-out examples with LU26 (function definition), but it has 33 worked-out examples with LU3 (assignment instruction). This could impact the allocation of worked-out examples to clusters (e.g. there may be several clusters representing LU3, whereas LU26 may not be represented by any cluster).

7. Student model data: a challenge in this research has been student data. The journey to get students to consent to participate in this research has been over-overwhelmingly challenging and difficult. Size of the student model is small for evaluation certain aspects of ERS, although it is worth mentioning that many of the ITS systems face this challenge (as shown in table 7.12.7.

8. Controlling external factors: ERS and this thesis did not consider controlling external factors such as a student's prior knowledge in the domain's LUs on his/her performance in the domain.

Despite these limitations, this research has made significant contributions to both the ITS community as well as the Educational Data Mining community.

8.3 Future Work

There are several directions that this research can lead to, presented broadly as three different perspectives.

1. Domain Model

   (a) Creating new examples from existing ones: ERS's experts create worked-out examples and tasks in its repository as and when required. Creating new examples and tasks from existing ones will reduce the time and efforts required by ERS's experts to update their repository.

   (b) Rating the examples in an attempt to find the best worked-out example in ERS: such a rating will assist in recommending relevant and popular examples to future students. It is worth mentioning here that we did a subjective evaluation of the examples by asking students about their opinion on the usefulness of the recommended worked-out examples and the results were 100% positive. Students found the examples extremely useful.

2. Student model

   (a) Creating simulated dataset for students - getting student data to evaluate an ITS is a very challenging task. The entire ITS community will benefit if methods to create new data from existing student data can be proposed and standardized for the ITS researchers.

   (b) Creating a benchmark student dataset for ITS built on the teaching methodology of Example-based Learning.

3. Model Improvements

   (a) Automating the process of constructing regular expression (RE) for any new domain. The domain model for KERE, currently requires the experts to provide the regular expressions of each LU in the domain of ERS.

   (b) Evaluating KOM16 using external validity index such as f_score. This can be done if all worked-out examples are assigned pre-defined classes. For example, the optimal number of clusters for domain D1 is found to be 6. If existing worked-out examples in each cluster have pre-defined class labels in advance, they can be compared with the clusters formed by KOM16 to compute an f_score value.

   (c) Implementing KOM16 in a course setting and compare its benefits in terms of usability and usefulness to students against lesson-wise organization.

   (d) Finding the best value of k for both MGREPD and KOM16 using validity indices.

   (e) Designing algorithms that can index the repository of worked-out examples for any domain using their metadata such as LUs and difficulty level of these examples.

   (f) Integrating association rule mining and sequential pattern mining (Srikant & Agrawal, 1996; Ezeife & Yi, 2009) into the framework to discover interesting relationships from student model data (e.g. students performance) as well as domain model data (e.g. LUs) and then use these associations to make further recommendations.

We hope that these contributions, and future directions will inspire researchers in the Educational Data Mining and Intelligent Tutoring System communities to further improve upon ITS that are based on Example-Based Learning method of teaching.

List of References

Agrawal, R., Srikant, R., et al. (1994). Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215 (pp. 487–499). 24

Alchin, M. (2013). Django is python. In *Pro Django* (pp. 11–40). Springer. 162

Arroyo, I., Walles, R., Beal, C. R., & Woolf, B. P. (2003). Tutoring for sat-math with wayang outpost. In *Advanced Technologies for Mathematics Education Workshop*. 1, 40

Bache, K. & Lichman, M. (2013). Uci machine learning repository. 114, 117

Beck, J. & Xiong, X. (2013). Limits to accuracy: how well can we do at student modeling? In *Educational Data Mining 2013*.

Brusilovsky, P. (2001). Webex: Learning from examples in a programming course. In *In Proc. of WebNet* (pp. 23–27). 41, 42, 43

Brusilovsky, P. & Peylo, C. (2003). Adaptive and intelligent web-based educational systems. *International Journal of Artificial Intelligence in Education*, 13(2), 159–172. 2

Burow, R. & Weber, G. (1996). Example explanation in learning environments. In *Intelligent Tutoring Systems, Springer Berlin Heidelberg,* (pp. 457–465). 28, 31

Caruana, R. & Niculescu-Mizil, A. (2004). Data mining in metric space: an empirical analysis of supervised learning performance criteria. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 69–78).: ACM.

Chai, S., Yang, J., & Cheng, Y. (2007). The research of improved apriori algorithm for mining association rules. In *IEEE International Conference on Service Systems and Service Management.* (pp. 1–4). 26

Chaturvedi, R., Berliane, E., & Ezeife, C. I. (2015a). Ers at https://ritu100.cs.uwindsor.ca/. xix, 94, 164

Chaturvedi, R., Donais, J., & Ezeife, C. I. (2015b). Ers at https://ritu106.cs.uwindsor.ca/. xix, 93, 163

Chaturvedi, R. & Ezeife, C. (2014). Mining relevant examples for learning in its student models. In *2014 IEEE International Conference on Computer and Information Technology* (pp. 743–750). xv, 9, 70, 72, 73, 75, 108

Chaturvedi, R. & Ezeife, C. (2015a). Mining boolean data using martrix algebra. In *IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), Oct. 2015* (pp. 901–906). 9, 12

Chaturvedi, R. & Ezeife, C. I. (2012). Data mining techniques for design of its student models. In *Fifth ACM International Conference on Educational Data Mining-EDM, June 21 - 23, Chania, Greece.* (pp. 232–233). 7, 9

Chaturvedi, R. & Ezeife, C. I. (2013). Mining the impact of course assignments on student performance. In *Sixth ACM International Conference on Educational Data Mining-EDM, July 6 to 9, Tennessee,USA.* (pp. 308–309). 9

Chaturvedi, R. & Ezeife, C. I. (2015b). Task-based example mining for learning in an iintelligent tutoring system. Submitted to JEDM (Journal Of Educational Data Mining). 9, 39, 64, 71

Chaturvedi, R., Martinovic, D., & Ezeife, C. I. (2015c). Mining game features to predict emotions. In *The International Conference on Business Tourism and Applied Sciences ICBTS 2015, Toronto.* 9, 112, 117

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, (pp. 321–357). 135

Chen, P. P. (1976). The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1, 9–36. 65

Chrysafiadi, K. & Virvou, M. (2013). Student modeling approaches: A literature review for the last decade. *Expert Systems with Applications*. Article in press. 4

Cox, R. (2007). Regular expression matching can be simple and fast. 91, 103

Domingos, P. (1999). The role of occam's razor in knowledge discovery. *Data mining and knowledge discovery*, 3(4), 409–425. 50

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, 55(10), 78–87. 128

Dubé, D. & Feeley, M. (2000). Efficiently building a parse tree from a regular expression. *Acta Informatica*, 37(2), 121–144. 103

Dunn, J. C. (1974). Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1), 95–104. 113

Encyclopedia.com (2002). "data mining and sports." computer sciences. 2002. retrieved march 16, 2016 from encyclopedia.com: http://www.encyclopedia.com/doc/1g2-3401200511.html. 6

Ezeife, C. (2010). *Problem Solving and Programs with C*. Nelson Thomson Learning Ltd. 56

Ezeife, C. I. & Yi, L. (2009). Fast incremental mining of web sequential patterns with plwap tree. *Data mining and knowledge discovery*, 19.3, 376–416. 150

Ferri, C., Hernández-Orallo, J., & Modroiu, R. (2009). An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1), 27–38.

Friedl, J. (2006). *Mastering Regular Expressions*. O'Reilly Media. 61

Frost, R. (2015). http://richard.myweb.cs.uwindsor.ca/. 57, 98

Gog, T. & Rummer, N. (2010). Example-based learning: Integrating cognitive and social-cognitive research perspectives. In *Edu. Psych. Rev., 22* (pp. 155–174). 7, 8, 31, 45

Goreva, N., Yudelson, M., & Marshall, B. (2007). Using webex in a web application programming course. *Issues in Information Systems*, 8(1-2), 52–57. 121

Greer, J. E. & Mark, M. A. (2016). Evaluation methodologies for intelligent tutoring systems revisited. *International Journal of Artificial Intelligence in Education*, 26(1), 387–392. 40

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1), 10–18. 136

Hamming, R. W. (1950). Error detecting and error correcting codes. *Bell System technical journal*, 29(2), 147–160. 71

Han, J. & Kamber, M. (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann. 24, 26

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1), 53–87. 24

Hosseini, R. & Brusilovsky, P. (2013). Javaparser: A fine-grain concept indexing tool for java problems. In *The First Workshop on AI-supported Education for Computer Science (AIEDCS 2013)* (pp. 60–63). 8, 30, 35, 36, 42, 43, 46, 49

Hosseini, R. & Brusilovsky, P. (2014). Example-based problem solving support using concept analysis of programming content. In *Intelligent Tutoring Systems* (pp. 683–685).: Springer. 8, 30, 35, 36, 48, 50, 95

Hosseini, R., Brusilovsky, P., & Guerra, J. (2013). Knowledge maximizer: Concept-based adaptive problem sequencing for exam preparation. In *Artificial Intelligence in Education* (pp. 848–851).: Springer. 42, 48

Hughes, J. (1989). Why functional programming matters. *The computer journal*, 32(2), 98–107. 98

Jaccard, P. (1901). *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz. 10, 71

Jeni, L., Cohn, J. F., De La Torre, F., et al. (2013). Facing imbalanced data–recommendations for the use of performance metrics. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on* (pp. 245–251).: IEEE.

Käser, T., Koedinger, K., & Gross, M. (2014). Different parameters-same prediction: An analysis of learning curves. In *Educational Data Mining 2014*.

Li, L. & Chen, G. (2009). A coursework support system for offering challenges and assistance by analyzing students web portfolios. *Educational Technology & Society*, 12,2, 205–221. xvii, 8, 29, 33, 34, 35, 40, 41, 42, 43, 46, 48, 98, 125, 129

Li, Q., Wang, H., Yan, Z., & Ma, S. (2001). Efficient mining of association rules by reducing the number of passes over the database. *Journal of Computer Science and Technology*, 16(2), 182–188. 26

Li, T. (2005). A general model for clustering binary data. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM, 2005.* (pp. 188–197). 80

Loney, K. (2008). *Oracle Database 11g The Complete Reference*. McGraw-Hill, Inc.

Lukasenko, R. (2012). *Development Of Student Model For Support Of Intelligent Tutoring System Functions*. PhD thesis, Faculty of Computer Science and Information Technology, Riga Technical University. Summary of Doctoral thesis submitted to Institute of Applied Computer Systems. 2, 4

Mabroukeh, N. (2010). *Semantic-based Web Recommendation System.* PhD thesis, Univeristy of Windsor, Windsor, Ontario, Canada. 129

Mark, M. A. & Greer, J. E. (1993). Evaluation methodologies for intelligent tutoring systems. *Journal of Artificial Intelligence in Education*, 4, 129–129. 40

Markov, Z. & Larose, D. (2007). *Data mining the web - Uncovering Patterns in Web Content, Structure and Usage.* John Wiley. 21, 70, 71, 80, 106, 107, 112, 129, 138

Maulik, U. & Bandyopadhyay, S. (2002). Performance evaluation of some clustering algorithms and validity indices. In *Pattern Analysis and Machine Intelligence, IEEE Transactions on* ,, volume 24 (pp. 1650–1654). 113

Mayer, R. E. & Moreno, R. (2003). Nine ways to reduce cognitive load in multimedia learning. *Educational psychologist*, 38(1), 43–52. 7

McCuaig, J. & Baldwin, J. (2012). : (pp. 160–163).: ERIC. 130

Millan, E., Loboda, T., & Perez-de-la cruz, J. (2010). Bayesian networks for student model engineering. *Computers & Education.*, 55,4, 1663–1683. 5

Minaei-Bidgoli, B., Kashy, D. A., Kortemeyer, G., & Punch, W. (2003). Predicting student performance: an application of data mining methods with an educational web-based system. In *Frontiers in education, 2003. FIE 2003 33rd annual*, volume 1 (pp. T2A–13).: IEEE. 129

Mokbel, B., Gross, S., Paassen, B., Pinkwart, N., & Hammer, B. (2013). Domain-independent proximity measures in its. In *Sixth ACM International Conference on Educational Data Mining-EDM 2013, July 6 to 9, 2013, Tennessee,USA* (pp. 334–335). 30, 35, 48, 50, 95

Moore, J. L., Dickson-Deane, C., & Galyen, K. (2011). E-learning, online learning, and distance learning environments: Are they the same? *The Internet and Higher Education*, 14(2), 129–135. 1

Muldner, K. & Conati, C. (2007). Evaluating a decision-theoretic approach to tailored example selection. In *Proceedings of the 20th international joint conference on Artifical intelligence., Morgan Kaufmann Publishers Inc.* (pp. 483–488). xvii, 29, 36, 38

Nghe, N. T., Janecek, P., & Haddawy, P. (2007). A comparative analysis of techniques for predicting academic performance. In *Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports, 2007. FIE'07. 37th Annual* (pp. T2G–7).: IEEE.

Ordonez, C. (2003). Clustering binary data streams with k-means. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. ACM, 2003.* (pp. 12–19). 80

OSBORNE/GAEBLER (1992). *Reinventing Government. How to Entrepreneurial Spirit is Transforming the Public Sector.* Addison Wesley.

Pang-Ning, T., Steinbach, M., & Kumar, V. (2005). *Introduction to Data Mining.* Addison-Wesley. xx, 6, 16, 17, 20, 21, 37, 69, 71, 80, 82, 83, 102, 117, 144

Pardos, Z. A. & Heffernan, N. T. (2010). Modeling individualization in a bayesian networks implementation of knowledge tracing. In *User Modeling, Adaptation, and Personalization* (pp. 255–266). Springer.

Pardos, Z. A., Heffernan, N. T., Anderson, B., & Heffernan, C. L. (2007). The effect of model granularity on student performance prediction using bayesian networks. In *User Modeling 2007* (pp. 435–439). Springer. 128

Pardos, Z. A. & Yudelson, M. (2013). Towards moment of learning accuracy. In *AIED Workshops.*

Park, J. S., Chen, M.-S., & Yu, P. S. (2005). An effective hash based algorithm for mining association rules. In *ACM SIGMOD International Conference On Management of Data.*, volume 24(2) (pp. 175–186). 26

Payam, R., Lei, T., & Huan, L. (2009). Cross-validation. In M. T. Ö. Edited by Ling Liu (Ed.), *Encyclopedia of Database Systems, Springer US.* (pp. 532–538). 106, 107, 136

Pei, Jian, J. H. B. M.-A. & Zhu., H. (2000). Mining access patterns efficiently from web logs. In S. B. Heidelberg. (Ed.), *Knowledge Discovery and Data Mining. Current Issues and New Applications.* (pp. pp. 396–407).

Pelánek, R. (2015). Metrics for evaluation of student models. *Journal of Educational Data Mining.*

Qiu, Y., Qi, Y., Lu, H., Pardos, Z., & Heffernan, N. (2011). Does time matter? modeling the effect of time with bayesian knowledge tracing. In *Educational Data Mining 2011.*

Quinlan, J. R. (2014). *C4. 5: programs for machine learning.* Elsevier. 23

Renkl, A. (2014). Toward an instructionally oriented theory of example-based learning. *Cognitive Science*, 38(1), 1–37. 7, 8, 31, 45

Romero, C. & Ventura, S. (2010). Educational data mining: a review of the state of the art. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews.*, 40(6), 601–618. 7, 15

Rovinelli, R. J. & Hambleton, R. K. (1976). On the use of content specialists in assessment of criterion-referenced test item validity. In *Annual Meeting of American Educational Research Association (60th, SanFrancisco, California). April 19-23.* 33, 98

Sarkar, A., Paul, Apurba., a. M. S. K., & Kumar, D. (2012). Modified apriori algorithm to find out association rules using tree based approach. In *IJCA Special Issue on International Conference on Computing, Communication and Sensor Network CCSN2012.* (pp. 25–28).: Foundation of Computer Science, New York, USA. 26

Schulze, K. G., Shelby, R. N., Treacy, D. J., Wintersgill, M. C., Vanlehn, K., & Gertner, A. (2000). Andes: An intelligent tutor for classical physics. *Journal of Electronic Publishing*, 6(1). 39

Shen, Y., Chen, Q., Fang, M., Yang, Q., Wu, T., Zheng, L., & Cai, Z. (2010). Predicting student performance: A solution for the kdd cup 2010 challenge. In *Proceedings of the KDD Cup 2010 Workshop held as part of the 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* 129

Shute, V. J. & Regian, J. (1993). Principles for evaluating intelligent tutoring systems. *Journal of Artificial Intelligence in Education*. 39

SOMYUREK, S. (2009). Student modeling: Recognizing the individual needs of users in e-learning environments. *International Journal of Human Sciences*, 6,2, 429–450. 5, 7

Srikant, R. & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *5th Int'l Conference on Extending Database Technology: Advances in Database Technology* (pp. 3–17). 150

Thai-Nghe, N., Drumond, L., Krohn-Grimberghe, A., & Schmidt-Thieme, L. (2010). Recommender system for predicting student performance. *Procedia Computer Science*, 1(2), 2811–2819. 129

VanLehn, K. (1998). Analogy events: How examples are used during problem solving. *Cognitive Science*, 22.3, 347–388. 31, 45

Vygotsky, L. (1987). Zone of proximal development. *Mind in society: The development of higher psychological processes*, 5291. 42

Wang, H. & Xiangwei, L. (2011). The research of improved association rules mining apriori algorithm. In *Eighth International Conference on Fuzzy Systems and Knowledge Discovery.*, volume 2 (pp. 961–964). 26

Wang, Y. & Beck, J. (2013). Class vs. student in a bayesian network student model. In *Artificial Intelligence in Education* (pp. 151–160).: Springer. 4

Wang, Y. & Heffernan, N. (2013). Extending knowledge tracing to allow partial credit: Using continuous versus binary nodes. In *Artificial Intelligence in Education* (pp. 181–188).: Springer.

Weber, G. & Brusilovsky, P. (2001). Elm-art: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education*, 12, 351–384. 28, 31, 35

Windsor, U. o. (2014a). https://blackboard.uwindsor.ca/. 1, 53

Windsor, U. o. (2014b). https://clew.uwindsor.ca/. 53, 56

Woolf, B. P. (2010). *Building intelligent interactive tutors: Student-centered strategies for revolutionizing e-learning.* Morgan Kaufmann. 39, 40, 41

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., & Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14.1, 1–37. 7

Yu, W., Wang, X., Wang, F., Wang, E., & Chen, B. (2008). The research of improved apriori algorithm for mining association rules. In *11th IEEE International Conference on Communication Technology.* (pp. 513–516). 26

Yudelson, M. & Brusilovsky, P. (2005). Navex: Providing navigation support for adaptive browsing of annotated code examples. In *In Proceedings of 12th International Conference on Artificial Intelligence in Education, AIED.* (pp. 18–22). 8, 28, 32, 35, 36, 40, 41, 42, 43, 46, 48, 50, 64, 90, 95, 125, 128

Yudelson, M. V., Koedinger, K. R., & Gordon, G. J. (2013). Individualized bayesian knowledge tracing models. In *Artificial Intelligence in Education* (pp. 171–180).: Springer.

Zhang, K. & Shasha, D. (1989). Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6), 1245–1262. 35

Zhang, Y., Capus, L., & Tourigny, N. (2007). A learner model for learning-by-example context. In *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing.*, volume 3 (pp. 778–785).: IEEE. 29

Appendix A

List of Abbreviations

Abbreviation	Full
ITS	Intelligent Tutoring System
EBL	Example-based learning
LU	Learning Unit in Domain on C Programming
DM	Domain Model
SM	Student Model
ERS	Example Recommendation System
KE	Knowledge Extraction
KC	Knowledge Customization
KO	Knowledge Organization
KERE	Knowledge Extraction using Regular Expressions
SMC	Simple Matching Coefficient
GREPD	Generate Relevant Examples and Predict Difficulty of a task
MGREPD	Modified Generate Relevant Examples and Predict Difficulty of a task
JC	Jaccard's Coefficient
DL	Difficulty Level
KOM16	Knowledge Organization modifying steps 1 and 6 of k-means

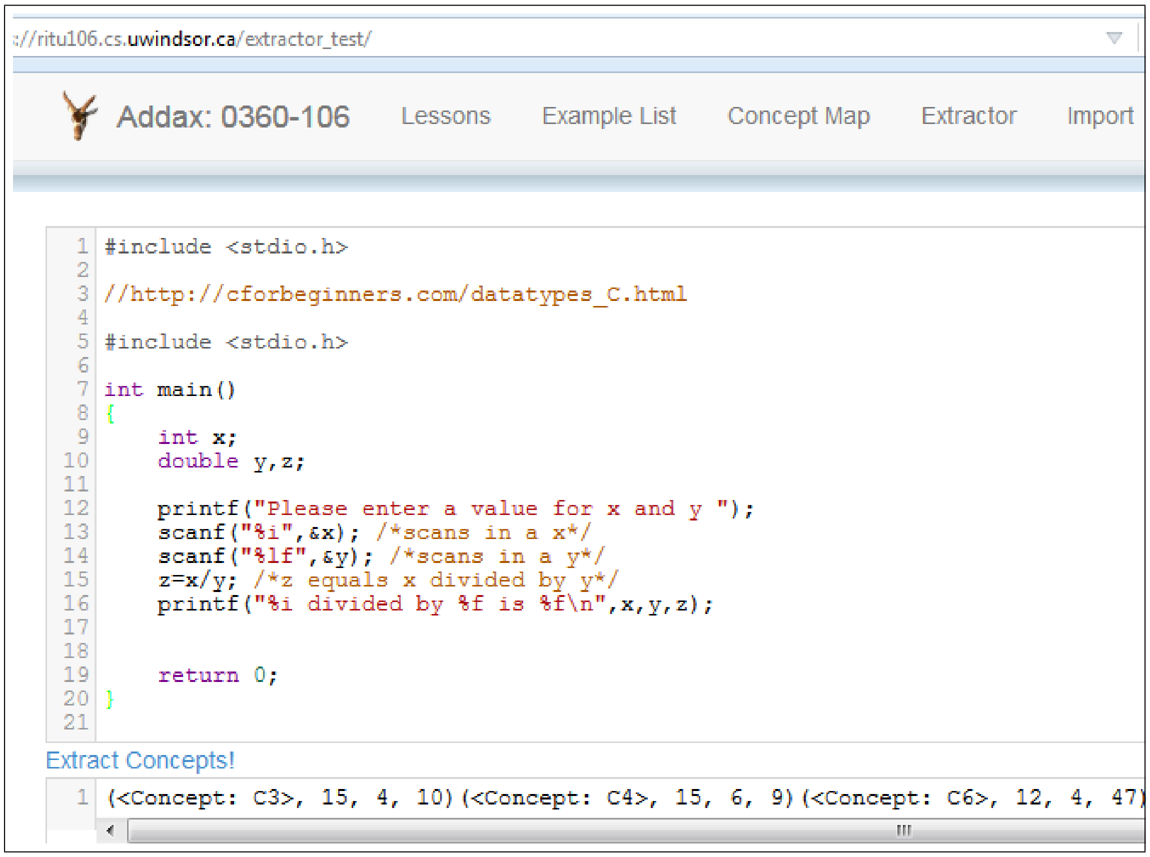
Appendix B

Implementation of ERS in Domain D1 and Domain D2

ERS for Domain D1 (C programming for beginners) successfully extracts LUs from worked-out examples and task solution given as C code. It has been launched as a web-driven application to assign tasks and recommend worked-out examples to students in a course on C programming for beginners, taught at the University of Windsor. ERS for domain D2 (Miranda) can extract LUs from worked-out examples and task solution given as Miranda programs, as shown in section B.2.

B.1 Tutoring using ERS in Domain D1 (C Programming for beginners)

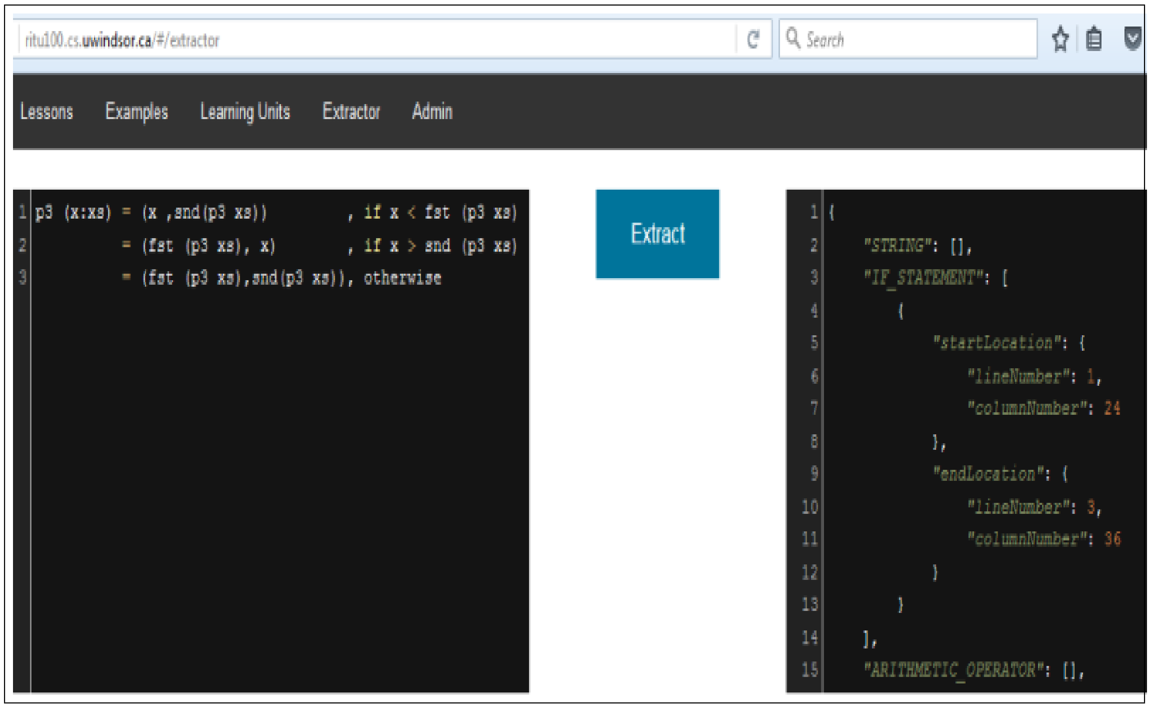
Figure B.1 is an image of the knowledge extraction module implemented for domain D1. The system is implemented as an HTML web application powered by Python 2.7 and Django 1.6 (Alchin, 2013). Please note that LUs are shown as concepts (C) in this image (e.g. C3 in this image corresponds to LU3 in table 3.1). The LUs (details can be found in 3.1) extracted for this worked-out examples are LU3, LU4, LU6, LU8 and LU11.



```c
#include <stdio.h>

//http://cforbeginners.com/datatypes_C.html

#include <stdio.h>

int main()
{
    int x;
    double y,z;

    printf("Please enter a value for x and y ");
    scanf("%i",&x); /*scans in a x*/
    scanf("%lf",&y); /*scans in a y*/
    z=x/y; /*z equals x divided by y*/
    printf("%i divided by %f is %f\n",x,y,z);


    return 0;
}

```

```
(<Concept: C3>, 15, 4, 10)(<Concept: C4>, 15, 6, 9)(<Concept: C6>, 12, 4, 47)
```

Figure B.1: Knowledge Extraction module of ERS for domain D1 at https://ritu106.cs.uwindsor.ca(Chaturvedi et al., 2015b)



```
p3 (x:xs) = (x ,snd(p3 xs))          , if x < fst (p3 xs)
          = (fst (p3 xs), x)         , if x > snd (p3 xs)
          = (fst (p3 xs),snd(p3 xs)), otherwise
```

```
{
    "STRING": [],
    "IF_STATEMENT": [
        {
            "startLocation": {
                "lineNumber": 1,
                "columnNumber": 24
            },
            "endLocation": {
                "lineNumber": 3,
                "columnNumber": 36
            }
        }
    ],
    "ARITHMETIC_OPERATOR": [],
```

Figure B.2: Knowledge Extraction module of ERS for domain D1 at https://ritu100.cs.uwindsor.ca(Chaturvedi et al., 2015a)

B.2 Knowledge Extraction in Domain D2 (Programming in Miranda)

Figure B.2 shows an image of the knowledge extraction module implemented for domain D2. The system is implemented using Java. Although the image shows just the IF statement (LU46) being extracted, it extracts other LUs (details found in 3.2) such as LU48, LU50, LU52, LU53 and LU54.

Vita Auctoris

Ritu Chaturvedi commenced her PhD in May 2012 and completed it in May 2016. She has been with the School of Computer Science, University of Windsor, in various capacities such as Lecturer (2001 - 2007 and 2010 - 2011), Learning Specialist (2007 - 2010) and Sessional Instructor (since 2011). Prioir to that, she taught for 8 years in Banaras Hindu University, India. She did her Masters in Computer Applications in 1988 from NIT, Rourkela, India and her B.Sc. in Physics Honours from Utal University, India in 1985.