International Journal on Data Science and Technology
2022; 8(1): 22-35
http://www.sciencepublishinggroup.com/j/ijdst
doi: 10.11648/j.ijdst.20220801.14
ISSN: 2472-2200 (Print); ISSN: 2472-2235 (Online)

Customized Learning in Online Tutoring Systems by Mining Learning Units from Tasks and Examples

Ritu Chaturvedi[1,*], **Christie I. Ezeife**[2]

[1]School of Computer Science, University of Guelph, Guelph, Canada
[2]School of Computer Science, University of Windsor, Windsor, Canada

Email address:
chaturvr@uoguelph.ca (R. Chaturvedi)
*Corresponding author

To cite this article:
Ritu Chaturvedi, Christie I. Ezeife. Customized Learning in Online Tutoring Systems by Mining Learning Units from Tasks and Examples. *International Journal on Data Science and Technology*. Vol. 8, No. 1, 2022, pp. 22-35. doi: 10.11648/j.ijdst.20220801.14

Received: February 9, 2022; **Accepted:** March 10, 2022; **Published:** March 30, 2022

Abstract: In recent years, technology has enabled Universities and Colleges to offer web-based courses, in which, teachers (or experts) design, curate and upload all course material required to teach the course online so that students can learn at their own pace, time and location. This research proposes a tutoring framework called Example Recommendation System (ERS) that is based on example-based learning (EBL) instructional method. ERS focuses on students devoting their time and cognitive capacity to studying worked-out examples so that they can enhance their learning and apply it to graded tasks assigned to them. ERS uses regular expression analysis to extract basic learning units (LU) (e.g. scanf is a LU in C programming) from all task solutions and worked-out examples and represents this knowledge in vector space. Then, these vectors are mined to generate a customized list of worked-out examples for each assigned task. The prime contribution of ERS's extraction module is its extendibility to new domains without requiring highly trained experts. Besides extendibility, ERS extracts LUs with 81% correctness for the domain of "Programming in C" and 95% for domain of "Programming in Miranda". ERS's data mining model used for customization has 93% accuracy and 88% f_score. ERS's educational impact is also evident from experiments that show that students score an average of 89% in tasks for which they use ERS's recommended worked-out examples, as opposed to an average of 73% for those tasks that students attempt without ERS's assistance.

Keywords: Customized Learning, ITS, Domain Model, Tasks and Examples, Knowledge Extraction, Regular Expressions, K-nearest Neighbors

1. Introduction

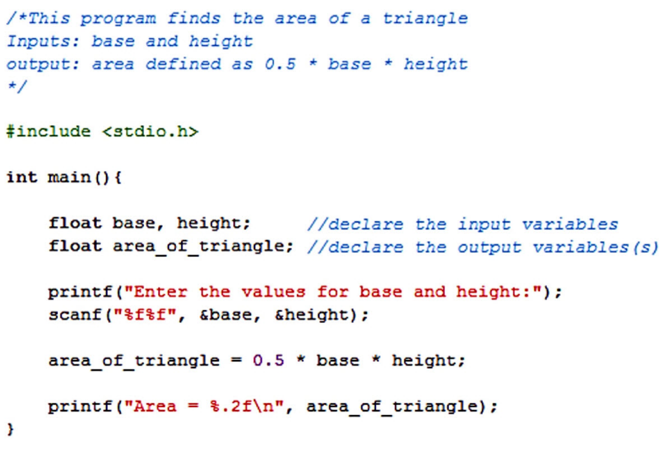
There are several diverse learning environments to teach a course in today's technological world such as traditional in-class, distance learning, web-based online systems and blended environments that combine classroom teaching and web-based technology. According to Moore's definition [1], distance learning 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 [2])) and Intelligent Tutoring Systems (ITS) (e.g. Wayang Outpost [3]).

For an ITS to achieve the core functionalities of adaptation and intelligence, it requires to capture student data that defines both their learning behavior (e.g. time spent on a worked-out example) 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. Section 1.1 explains briefly the domain and student components required by any ITS.

1.1. Domain and Student Model

A domain model defines the expertise required to teach the domain. In traditional classroom teaching, such expertise comes from a combination of teacher and text book put together. To gain such expertise, an ITS's domain model requires to store all basic concepts that define that domain, and the resources required to teach or learn these concepts. These resources include among others, worked-out examples and gradable tasks such as quizzes, tests and assignments. A worked-out example (WE) in this paper is defined as a complete or partial worked-out solution explaining or demonstrating one or more domain concepts (similar to examples in textbooks). It is worth noting that this definition of a worked-out example is somewhat different than worked-out examples defined by Atkinson et al. [4] (in which each worked-out example presents a solution in a step-by-step fashion). Figure 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 one or more concepts in the domain. Typically, a task is graded so that student performance in the course can be measured objectively using these grades. 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". Being the expert, an ITS domain model also stores complete solution of every task. This paper uses the same structure to define a task solution and a worked-out example (WE). Concepts can be defined by domain experts at various levels of detail or granularity. With reference to the domain of C programming, a concept can be defined broadly as a lesson in a textbook (such as a lesson on loops) or very finely as an element in a specific type of a loop (such as header of a for loop). Experts for this research define concepts at a fine level of granularity and call them as Learning Units (LU). For example, "scanf" is an LU in the domain of C programming that allows users to enter values from the keyboard. Similarly, "fraction" is an example of a 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. Sections 3 and 4 of this paper propose automated methods to design and build domain models that are easily extendable to other domains without requiring much effort required of experts.

A student model (SM) is an approximate, partial, mainly qualitative representation of the student's knowledge about a specific domain [5]. 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 the 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 [5, 6]. 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). An example of dynamic student characteristic is performance (e.g. marks scored in tests). Student modeling is a process of storing student data and making inferences about student's characteristics, their learning behavior and abilities [7]. ITS, like many other application areas, adapt data mining techniques to perform functions such as automatically capturing student actions and making inferences on them [8, 9, 10, 11]. Decision trees, k-nearest neighbors and Bayesian Networks (BN) are the most commonly used predictive mining methods in ITS systems [10, 11]. Research on existing EBL-based ITS [12, 13, 14] 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.



```
/*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. 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".

1.2. Outline of the Paper

This paper is organized as follows: Section 2 presents related work and our motivation for this research. Section 3 presents architecture used for the proposed ERS (Example Recommendation System), the main ERS algorithm, and a simplified application of ERS. Section 4 evaluates the educational impact of ERS as a tutor, whereas section 5 evaluates the individual ERS components. Section 6 presents conclusions and future work.

2. Related Work

Example-based learning (EBL) [15, 16] 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 shows a worked-out example of a C program that finds the area of a triangle. Renkl [16] 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 exceeds his/her processing capability [17] (e.g. extraneous material) and therefore makes learning ineffective. Following these principles, this paper 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.

An extensive and comprehensive survey of existing systems that use worked-out examples to teach a domain has been done in an earlier publication [18]. One of the pioneers in example-based learning systems is ELM-PE [19] that used a technology which helped students solve new problems in LISP programming language by presenting them with successfully solved problems such as examples. It also focused on adaptive navigation by hiding and disabling the links to those pages that were not ready to be learned. SEATS [20] was a web-based system designed for well-structured domains to evaluate if adaptive and structural example-based learning is effective. It provided help to students by presenting side-by-side examples and highlighting their common structural components. This paper elaborates the methodologies used in the framework used by NavEx [12] and PADS [13] that are very similar in their objective of adaptively finding and presenting most relevant examples to students, yet very different in the methodologies used. NavEx and PADS and other existing systems cited in this paper are only a part of an ITS. Section 1 (introduction) states the four essential components (domain, student and teaching models and an interface) that ITS typically consist of. The proposed system focuses on the domain model and student model components. Every ITS requires the domain resources to be extracted and represented conveniently so that customization methods can be applied to them. We refer to such extraction as KE (Knowledge Extraction) and KC (Knowledge Customization). KE can be done either automatically or manually. NavEx [12] uses parsers and syntax tree representation to automatically extract concepts from C programs. It transforms each worked-out example (WE), given as a C program, into its equivalent syntax tree, and then parses this tree to extract the WE's main concepts. Other existing systems ([14, 21, 22]) also use such syntax tree representation to extract concepts from their domain examples. Mokbel et al. [23] divide their solution's syntax trees into sub graphs using spectral clustering and then measure the proximity between solution parts represented as TFIDF weights. Representing each solution as a syntax tree for matching examples requires a complex expert knowledge of parsers and compiler construction, and makes the existing systems less adaptable to other problem domains.

PADS [13] uses a manual method of extraction called IOC [24], in which nine experts gave their opinion of whether or not a concept should belong to an exercise (even though PADS was experimented with exercises, the methodology applies to worked-out examples as well). Then, an index value is calculated for each concept i in exercise k as

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

where $N = |concepts|, n = |experts|$ 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.

In order to customize resources to student needs, NavEx [12] stores student knowledge on a concept as a continuous value[25]. In PADS [13], progress of a student on a concept is stored as 1 (low), 2 (medium) and 3 (high). Customization (KC) in NavEx is done by simply matching each student's SM contents with that of the example's pre-requisite concepts – if student has mastered all concepts of an example, it is marked as 'Ready". A threshold value to compute the number of clicks that a student had to make to master the example concepts was defined as 0.8*(#all_concepts - #mastered_concepts) / (#all_concepts) * #clicks_possible, where #clicks_possible for each example was given by an expert. PADS [13] uses decision trees to predict the difficulty level of an exercise for a student S based on seven feature attributes and one target attribute. The feature attributes are extracted from different sources such as expert opinion (e.g. expert opinion on difficulty level of the algorithm), student model (e.g. student's performance on main concepts of an exercise) and web logs (e.g. login times during a week). At the end of an exercise, each student is asked if he/she completed the exercise with any help – if S did it without any help or could not do it, then target attribute is set to 0 (=> inappropriate); if S did it with the help of course materials or peer collaboration, then target attribute is set to 1 (=> appropriate). The methods used by the existing systems [12, 13, 23] are very subjective, dependent on a highly expensive resource such as experts with complex knowledge on areas such as parser generation and are less-focused as they generate a broad range of examples independent of any task . For example, PADS uses expert opinion for three of its seven feature attributes. Both NavEx and PADS compare their SM with N examples (N = #examples in the database). KC in NavEx [12] uses number of clicks to determine whether a student has sufficiently explored the example but students can easily game this system by just clicking randomly to move on to the next example. PADS [13] uses just the student's opinion to find if the example is at an appropriate level or not. The existing systems also lack in adapting to situations where a student may have explored the example but has not mastered its concepts yet. Adding some objectivity to the examples (e.g. compute scores for each ungraded example) can be a solution to this problem. These limitations motivated us to develop an architecture that improves learning in students using ITS by presenting them with a more focused and concise list of examples in real-time.

This paper proposes automated efficient extraction methods for extracting domain model components such as worked-out examples such that they can be easily extended to other domains. These methods also allow for a convenient representation of domain knowledge so that mining methods can be applied to customize the ITS resources to student needs.

3. Proposed System: Example Recommendation System (ERS)

A literature review of existing methodologies [12, 13, 23] used in the area of adapting examples to a student's mastery of domain concepts indicates that these methodologies can be broken down into 3 phases : Knowledge Extraction (KE), Knowledge Organization (KO) and Knowledge Customization (KC). This paper focuses on the Knowledge Extraction and Knowledge Customization components.

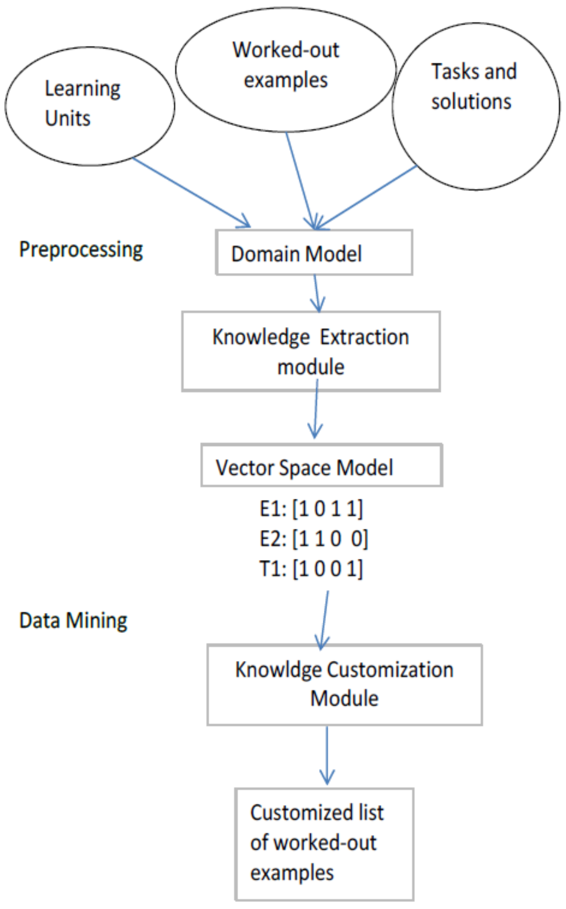


Figure 2. Architecture of the proposed System.

3.1. Contributions

This paper addresses the limitations highlighted in section 2 (related works) by proposing a novel architecture for a knowledge-empowered task-based example recommendation system (ERS) for learning in an ITS as follows:

1. Knowledge extraction (KE) module of ERS uses regular expressions to extract LUs of examples and tasks. Regular expressions require expert knowledge that is easily defined and maintained, can be easily extended to a higher scope within the current domain and can also be adapted to other subject domains. In addition, they allow for more uniform conversion of examples and tasks to similar LUs so that the most relevant examples can be mined.
2. Knowledge Customization (KC) module of ERS generates a customized list of examples for the task that is being answered. Such a list reduces the cognitive overload on students and allows them to focus on the most relevant examples that will assist them in accomplishing the task. ERS uses k-nearest neighbor (k-nn) classification algorithm to generate such a list for task t_i and assign a difficulty level to t_i. Algorithm k-nn searches for the examples that are closest to a given task based on the learning units (LU) covered by examples and those required for the task.

Figure 2 shows the basic architecture of the proposed framework that clearly integrates its basic components such as extraction and customization. Knowledge extraction module takes the set of learning Units (LU) in the domain given by experts and resources such as worked-out examples and task solutions as input. It then extracts LUs from them using regular expressions and represents each resource as a vector of n binary values, where n is the total number of LUs in the domain (e.g. if n = 4, a worked-out example in that domain could be represented as [1,0,1,1]). These vectors are given as input to the KC (knowledge Customization) module that uses data mining techniques to extract the k most relevant examples related to a given task.

Sections 3.2 and 3.3 present the core algorithms and methods used in knowledge extraction from ITS resources, their representation and customization of resources to student needs. Section 3.4 demonstrates an example that clearly integrates ERS modules to accomplish its objective of improving learning.

3.2. Knowledge Extraction in ERS

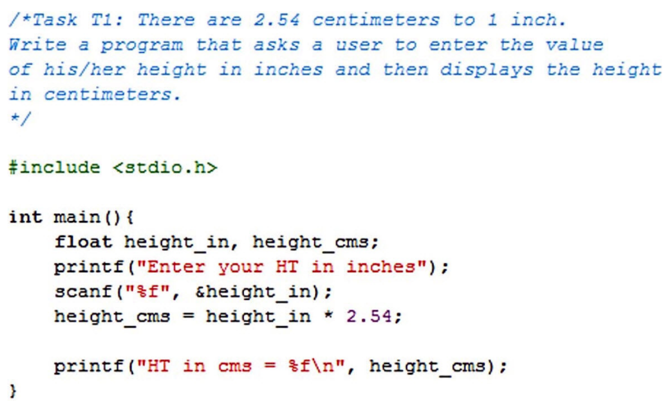
The goal of knowledge extraction (KE) module of ERS is to automatically transform each task solution and worked-out example in its domain to a binary vector of size n, where n = number of learning units (LU) in ERS. The main motivation to design and implement new KE algorithms for EBL-based ITS such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

3.2.1. Domain Model

Domain model of ERS is built by a pool of experts who are Computer Science instructors and students (graduate and undergraduate majors) and consists of the following items: (1) Worked-out examples (e.g Figure 1) (2) Tasks (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") (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 shows a solution for task T1. (4) Learning units (LUs) : are defined by experts within the scope of ERS. ERS experts are chosen from a pool of undergraduate students with Computer Science major. 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}. They typically use the hierarchy of concepts given in a textbook on "Programming in C" to decide the order of difficulty [26]. (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 3.2.2 defines regular expressions and the process of using regular expressions for KE module of ERS.

3.2.2. Knowledge Extraction Using Regular Expressions

ERS domain experts define concepts at a fine level of granularity and call them as learning units (LU) (section 1.1). This research uses regular expression analysis in order to represent each worked-out example (WE) or task as a collection of various LUs that are to core to that WE or task. A regular expression (RE) is defined as a set of characters that describe a pattern [27]. 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 —). For example, '+' symbol matches the preceding character one or more times (e.g. *Joh+n* matches strings such as *John* and *Johhhhn* but not *Jon)*, whereas symbol '*' matches the preceding character zero or more times (e.g. *Joh*n* matches *Jon* in addition to *John* and *Johhhhn).* ERS experts define a unique RE for each LU included in its domain. For example, RE used by ERS for learning unit LU7 is defined as: <LU7, printf \(\" (\%[dcf])+ \" \, [a-zA-Z_][a-zA-Z0-9_]* (\, ([a-zA-Z_][a-zA-Z0-9_]*))+ \) ;>. The next subsection presents an algorithm called KERE that automates this process of extracting LUs from any given C program (be it a worked-out example or the solution of a task) using regular expressions.


```
/*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. *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".*

Algorithm 1 Knowledge extraction using regular expressions (KERE)

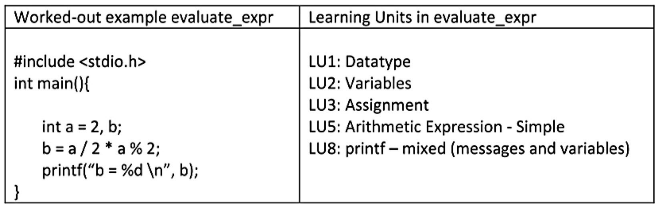
Inputs: learning Units (LU) in the domai, set of regular expressions {RE} - one for each concept in LU, string TE_{soln} of task / example solution
Output: LU_TE: list of {LUs that TE_{soln} covers, number of times a LU occurs in TE_{soln}}
Method:

```
***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
```

3.2.3. Proposed Algorithm for Knowledge Extraction (KERE)

This section defines KERE (Knowledge Extraction using Regular Expressions), an algorithm that identifies 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 1) 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 that LU appears in it. Figure 4 shows a worked-out example and the LUs extracted from it by KERE {LU1, LU2, LU3, LU5, LU8}.



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

Figure 4. Worked-out example evaluate_expr and its LUs extracted by KERE.

3.3. 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.

3.3.1. Computing the Similarity Between Worked-Out Examples and Task Solutions

The core algorithm of this module is driven by distance / similarity functions such as Euclidean distance [28]. Similarity function used in ERS 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 common similarity functions used with binary data are Jaccard's coefficient (JC) [29] and Hamming distance [30]. JC works best with binary asymmetric attributes [28] and therefore are more applicable to ERS's domain data. Jaccard's coefficient between two binary vectors x and y is measured as

$$JC(x,y) = \frac{f_{11}}{f_{11} + f_{01+}f_{01}} \tag{2}$$

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

$$HD(x,y) = \frac{f_{01} + f_{10}}{f_{11} + f_{01+}f_{01} + f_{00}} \tag{3}$$

For example, $HD(x, y) = 1/2$.

3.3.2. MGREPD (Modified Method to Generate Relevant Examples and Predict Difficulty of a Task)

MGREPD is designed to overcome the limitations of GREPD that was proposed in an earlier work by authors on customization [31]. GREPD (Generate Relevant Examples and Predict Difficulty of a task) takes as input the (transformed) vector representation of all worked-out examples and tasks solutions in ERS's domain, and then uses k-nearest neighbor predictive mining algorithm [28] to output the list of worked-out examples closest to the given task and also 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 n values, LU_1 to LU_n (where n = total number of LUs in ERS). For example, if n = 10 (the first 10 LUs in table 2 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 [31], which takes both local (TF calculates weights locally, taking into account a specific LU and worked-out example) and global information on worked-out examples (IDF computes the weights globally). GREPD had 2 limitations: (1) The dataset generated by KR module of ERS is mostly binary and asymmetric and cosine similarity used in GREPD is not suitable for such data (2) The actual class labels for each worked-out example used in the k-nn algorithm of GREPD were manually given by experts. MGREPD is designed to overcome these limitations. MGREPD 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 to assign a difficulty level DL to each worked-out example as 'E' for easy and 'D' for difficult. DL of any example is based on the total number of LUs it contains (numLU) and the LU of the highest difficulty (highestId) (as provided in the domain model). If the highestId LU belongs to the complex set, DL of that example is set to 'D' or difficult. If the highestId LU is simple, and numLU is less than nine, then DL of that example is set to 'E' or easy; otherwise, it is set to 'D' or difficult. The set of LUs are divided into complex and simple by experts and storesd as such in the domain model.

Algorithm MGREPD

MGREPD (algorithm 2) computes the similarity between a task solution and each worked-out example using Jaccard's coefficient (equation 2), sorts these JC values and generates a list l of k examples closest to the task using k_nearest neighbor predictive mining algorithm (k-nn) [28]. 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.

Algorithm 2 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 - uses k-nn to find the k most relevant examples of a task using the JC between the task and examples
1. Compute Jaccard's coefficient $JC_T$ between $LU\_T$ and each row of $LU\_EX$
2. Sort the $JC_T$ values computed in step 1 in descending order and store the top k of them in $List_{relevant}$
3. for each worked-out example $e_i$ in $List_{relevant}$
Find the difficulty level of $e_i$ as "E" or "D" and assign it as DL($e_i$)
Let countE = number of examples in $List_{relevant}$ with DL = E
Let 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
```

Algorithm 3 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 $Tq$
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 MGREPD of Knowledge Customization (KC) module to select the most relevant list of examples for task $Tq$.
***end of ERS_main
```

3.4. 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.

3.4.1. The ERS Algorithm

The ERS_main algorithm calls the core algorithms for each of its components KE and KC as shown in Algorithm 3. Section 3.4.2 demonstrates an example application of ERS_main.

3.4.2. An Example Application of the Proposed ERS System

Example 1: Given the domain model, this example demonstrates how ERS's KE module extracts knowledge, and mines the examples to generate a customized list for students attempting task T1E in its KC module, given k = 2.

Domain model (subset of ERS's domain D1)

1. Learning Units: There are currently 17 LU in its domain, each LU represented as a tuple <id, description): {<L1, datatypes>, <L2, variable>, <L3, assignment>, <L4, Simple Arithmetic Expression>, <L5, Compound Arithmetic Expression>, <L6, printf-constant messages only>, <L7, printf-format specifiers only>, <L8, printf-mixed>, <L9, scanf - single input>, <L10, scanf - multiple inputs>, <L11, relational Expression>, <L12, logical expression>, <L13, relational-logical>, <L14, arithmetic-relational-logical>, <L15, simple while>, <L16, simple for>, <L17, if-else>}
2. Worked-out examples: E1, E2, E3, E4, E5 as shown in table 1.
3. Tasks and task solutions: T1E and its solution shown in table 1. For simplicity, there is only one task in the scope of this domain.

Table 1. *Worked-out examples and Task solution used in example 1.*

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 $\geq$ 18 && age $\leq$ 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 $\geq$ 18 && age $\leq$ 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);

Solution of example 1 : Steps 1 - 4 of algorithm ERS_main (algorithm 3) are executed as follows:

1. *Step 1: KE: knowledge extraction:* The core algorithm of this component is KERE (algorithm 1).
   (a) Inputs to KERE: <set of LUs, let n=#LUs (n = 17), REs for each LU, worked-out examples>
   (b) Method
      i. KERE is applied to worked-out examples in table 1 to obtain 5 binary vectors of size 17 (one for each worked-out example). 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. We call the resulting Boolean matrix as ∂.
   (c) Output of KERE: 5 * 17 Boolean matrix ∂ shown in table 2.

Table 2. *Dataset ∂ : 5 * 17 matrix to represent examples and their LUs (defined in table 1).*

	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	0e	0	0	0	0e
E2	0	0	1	1	0	0	0	1	0	0	0	0	0e	0	0	0	0e
E3	0	0	1	1	0	0	0	1	0	0	0	1	0e	0	0	1	0e
E4	0	0	0	0	0	0	0	0	0	0	0	1	0e	1	0	0	0e
E5	0	0	0	0	0	1	0	0	0	0	0	1	0e	1	0	0	1e

1. *Step 2: Knowledge Customization :* The core algorithm here is MGREPD (algorithm 2).
   (a) Inputs to KC: <dataset ∂ (table 2), k = 3, DL for examples in table 1 = {'E', 'E', 'D', 'E', 'E'}, Task T1E as a binary vector LU_TE = [0,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0]>
   (b) Method:
      i. MGREPD finds the similarity between LU_TE and each row in ∂ as [0.25, 0.5, 0.33, 0, 0.1667] using jaccard's coefficient, sorts them in descending order and stores top k in $List_{relevant}$. Thus $List_{relevant}$ for task T1E = [E2, E3, E1], implying that students working on task T1E will find examples E2,

E3 and E1 as the most useful examples to succeed in task T1E.

ii. T1E is assigned a difficulty level of 'E' (easy), since the difficulty level of 2 (out of 3) examples in T1's neighborhood is 'E',

(c) Outputs of KC: $<List_{relevant}$for task T1E, predicted difficulty level of T1E = 'E'$>$

4. Evaluation of ERS

This section evaluates the core ERS algorithms KERE and MGREPD. Section 3 highlights the datasets created and used by ERS. Section 4.1 presents an experimental analysis of the knowledge extraction component (described in section 3.2) and section 4.2 presents the results and analysis of the knowledge customization module of ERS (described in section 3.3). Section 4.3 demonstrates the experiments done with different student groups in an attempt to evaluate ERS as an effective tutor. The domain model created for ERS has 250 worked-out example, 31 tasks and tasks solutions and 27 LUs.

4.1. Experiment 1 on Knowledge Extraction

KERE (Knowledge Extraction using Regular Expressions), as described in section 3.2 identifies the presence of LUs in a given task solution or worked-out example. Section 4.1.1 demonstrates how it is easily extendable to a domain that teaches Miranda functional programming language and also to a non-programming Math domain. Section 4.1.2 validates the correctness of KERE by comparing the LUs generated by KERE with LUs generated manually by using a method called IOC. Section 4.4 briefly illustrates the tutoring effectiveness of ERS.

4.1.1. 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. This paper 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 [12, 14, 23]) - 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. KERE's extendibility is tested using a programming language called Miranda [18], which is from a programming paradigm different than C. Miranda is a functional language, as opposed to C, which is procedural. KERE is implemented for Miranda with its domain model D2 consisting of (1) 16 learning units (LU), (2) RE for each LU (3) 101 worked-out examples and (4) 12 tasks (and their solutions) [18].

4.1.2. Validation of KERE in Extracting the Correct LUs

In order to validate the correctness of the LUs extracted by KERE, its results are compared with a manual extraction method called Item-Objective Congruency (IOC) used commonly in the area of Content Validity [13, 24]. The IOC method collects and analyses judgments from several experts on the relevance of an item (e.g. a learning unit) for an instructional resource (e.g.worked-out example). 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 using equation 1. Expert ratings are collected and compiled for each worked-out example in the domain. The LUs extracted by KERE are then validated against these expert ratings using simple matching coefficient (SMC). SMC between two binary vectors of the same size is given in equation 4.

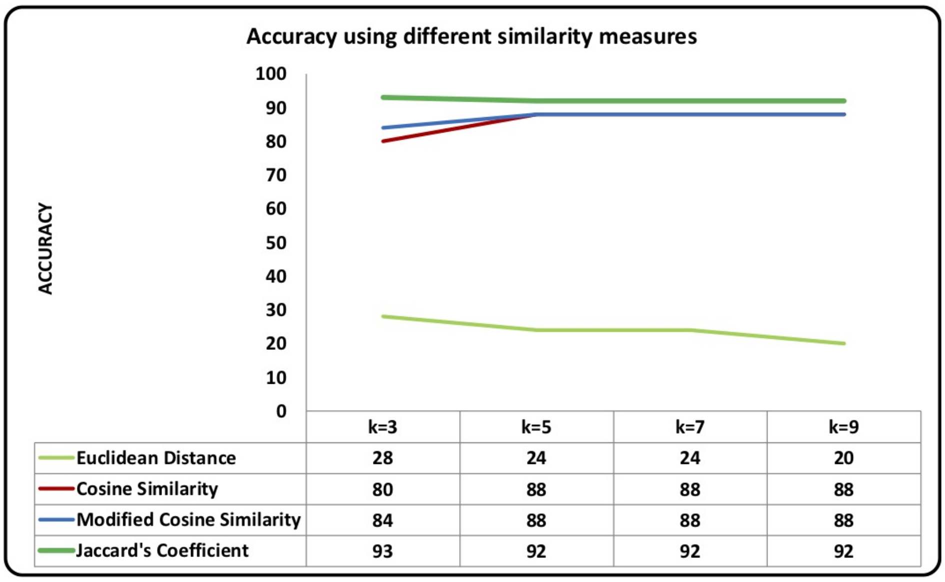
$$SMC(x,y) = \frac{f_{11} + f_{00}}{f_{11} + f_{00} + f_{01+}f_{01}} \tag{4}$$

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. For example, $SMC([10011], [10101]) = 3/5 = 0.6$. KERE extracts LUs for domain of Miranda with a 95% accuracy, indicating that experts agree very strongly with the LUs extracted by KERE.

4.2. Experiment 2 on Knowledge Customization

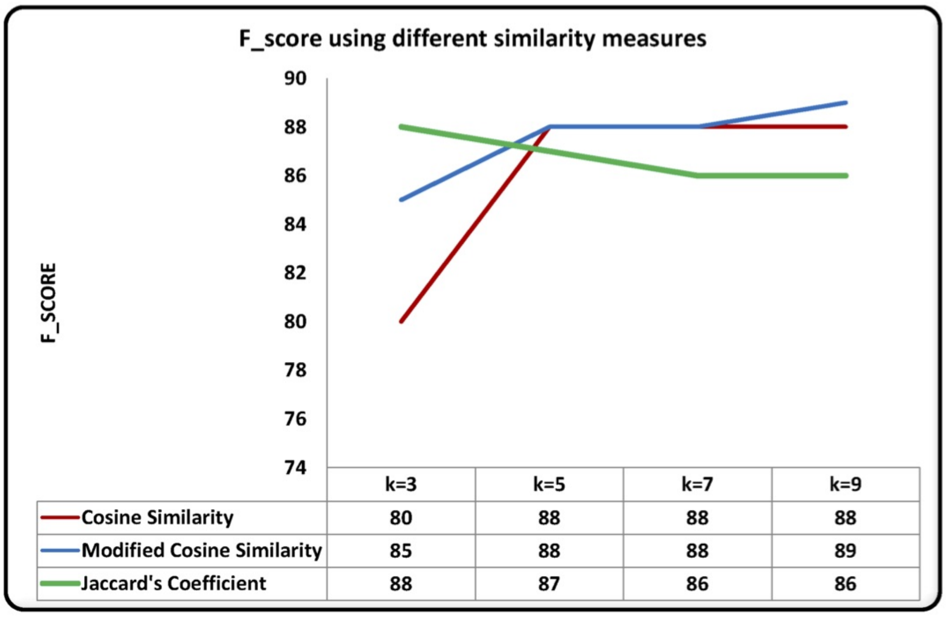
MGREPD algorithm builds a k-nearest neighbor prediction model using different similarity functions that are observed to be applicable to ERS datasets. MGREPD uses leave-one-out cross validation (LOOCV) to evaluate its model. In each iteration of LOOCV, one sample (e.g. ith worked-out example) from the 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 attributes numLU and highestId as explained in section 3.3.2. 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) [32] and measures such as accuracy and f-score are used [33]. 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"). In essence, it is the number of correct predictions divided by the total number of predictions. *Accuracy* is not meaningful when dealing with class labels that are imbalanced or when one of the class labels is uncommon. For example, 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 number of actual true values correctly predicted as true divided by the total number of values predicted as true) and *recall* (defined as number of actual true values correctly predicted as true divided by 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* and is defined as shown in equation 5.

$$f_{score} = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = \frac{2 * recall * precision}{recall + precision} \quad (5)$$



	k=3	k=5	k=7	k=9
Euclidean Distance	28	24	24	20
Cosine Similarity	80	88	88	88
Modified Cosine Similarity	84	88	88	88
Jaccard's Coefficient	93	92	92	92

Figure 5. *Comparison of Accuracy of MGREPD using Jaccard's coefficient and other similarity measures such as Cosine similarity.*



	k=3	k=5	k=7	k=9
Cosine Similarity	80	88	88	88
Modified Cosine Similarity	85	88	88	89
Jaccard's Coefficient	88	87	86	86

Figure 6. *Comparison of f_score of MGREPD using Jaccard's coefficient and other similarity measures such as Cosine similarity.*

Figures 5 and 6 show results published in our earlier work for algorithm GREPD 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 [31]. In this work, accuracy and f_score were compared using Euclidean distance and cosine similarity with the proposed measure called modified cosine similarity (MCS) and it was found that results of using cosine and modified cosine similarity on dataset ℓ are not very different for values of k greater than 3. This paper compares the results of GREPD with the proposed algorithm MGREPD, that uses Jaccard's coefficient to measure the similarity between a task and worked-out examples. As the graph in figure 5 shows, MGREPD using Jaccard's coefficient 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 test data records. 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 3). A comparative evaluation of performance of MGREPD using JC and hamming distance as similarity functions is shown in figures 7 and 8. These graphs indicate that MGREPD that uses JC performs better than hamming distance for ERS domain D1. 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.

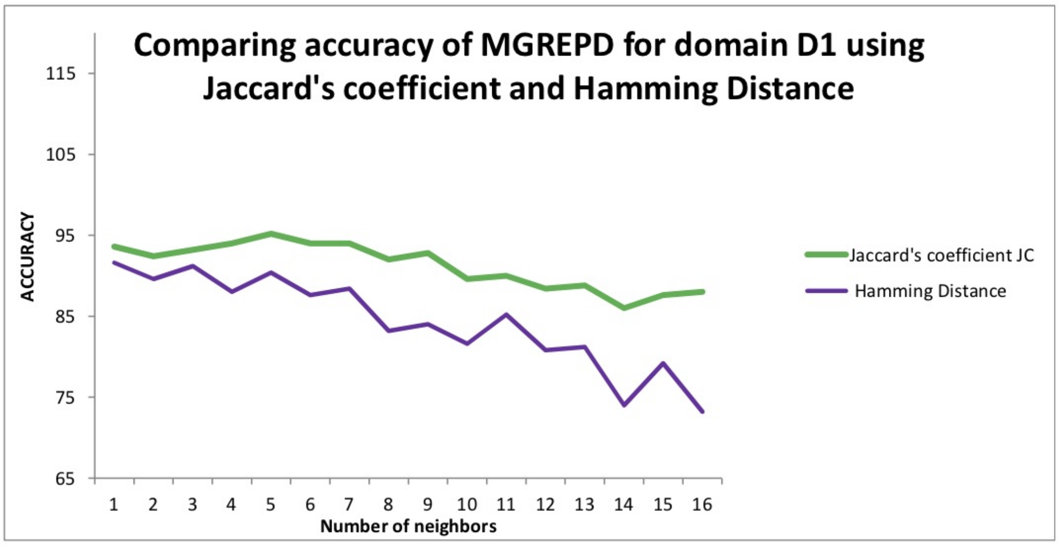


Figure 7. Comparison of accuracy of Jaccard's coefficient verses Hamming Distance for domain D1.

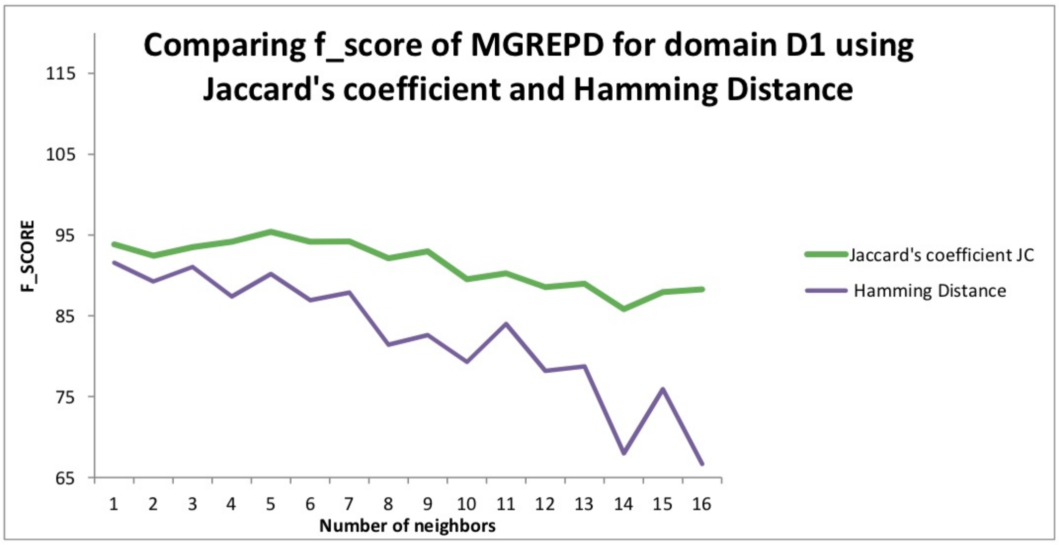


Figure 8. Comparison of f_score of Jaccard's coefficient verses Hamming Distance for domain D1.

4.3. ERS as a Tutor

This section describes the methods adapted to evaluate ERS as a tutor, in order to validate its prime goal of analyzing the effectiveness of worked-out examples carefully selected and recommended by ERS for a given task and their impact on 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. The total number of students used in this study from Winter 2015 were 48 and those from Fall 2015 were 34. 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.

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. It was observed through experiments that the class average for assignments in group I (WithExamples) is 89%, whereas the average for assignments in group II (NoExamples) is just a 73%. In scenario 2, performance of two different groups of students that use the same set of assignments is compared. For Group I in this scenario (Winter 2015), students are not required to use examples; whereas group II students (Fall 2015) are required to use the examples recommended by ERS. Average for group II was found to be 89% as compared to 83% measured for students in group I.

5. Conclusions, Limitations and Future Work

The goal of this paper is to develop an example-empowered task-centered learning framework for online courses, to implement the well-adapted theory of example-based learning. The novelty of our approach is three-fold: (1) Clear separation of the framework into independent components such as knowledge extraction, representation and customization. This enables us to treat each component independently so that any changes made to any one component does not impact the working of other components. (2) Easiness of defining the domain model without requiring highly trained experts, thus enabling its extendability to newer domains. (3) Mining worked-out examples to recommend the most relevant ones specific for a given task in order to help students perform better in assigned tasks.

The existing frameworks in this area so far ([12, 13]) have either used manual methods or highly complicated and demanding automated methods for knowledge extraction (such as syntax trees to represent examples and tasks). These methods not only limit the systems extendability to other LUs in the current domain, (since each new LU added to the system will require experts to be well-versed with syntax trees and parsers or with the entire domain LUs (for manual methods)), but they also limit the extendability of KE methods to other domains. ERS's KE methods (sections 4.2 and 6.2) demonstrate how easy it is to define regular expressions for learning units of any domain and how this method of extraction contributes substantially to the ITS and EDM community. There are obvious limitations of using this method to extract LUs from certain subjects in Arts and Social Sciences that have less rigorous structure and more semantic ambiguity by both manual and automatic methods and remains a focus of future work. The authers have also extended their knowledge extraction techniques [34] using a novel keyword based search tree (k-BST) method that recommends relevant code fragments by extracting existing keywords, matching with relevant coding examples by k-means clustering, and recommending the relevant coding examples back to the user. For reasons similar to the ones stated above, the straight-forward approach of using regular expressions for knowledge extraction might not be suitable to incorporate concepts such as polymorphism in object-oriented languages. Our structured knowledge base of examples and tasks allows both simple querying and use of data mining techniques to enrich the process of knowledge customization. One of the challenges that students face in an online course is abundance of information, which includes both relevant and irrelevant information (relevant to the assigned task). The purpose of KC module is to effectively search for only those examples that cover LUs similar to the current task, thereby reducing the cognitive overload in terms of number of relevant examples presented. Both scenarios in section 4.3 indicate that students have a higher likelihood of getting high marks in tasks if they study the worked-out examples recommended by ERS. A limitation of the proposed approach is that it generates the same set of examples (for a given task) for all students, and therefore does not adapt to students needs.

Future work also includes integrating association rule mining and sequential pattern mining techniques into the framework so that we can refine the process of knowledge organization and customization to identify sets and sequences of LUs that often occur together and can belong to the same group, or recommended next to students, even if it is not obvious from the examples in the database. Further, socially adaptive mechanisms such as number of ‘likes’ given to an example by students can be used to rate examples and recommend them to future students. Another future direction is to automate the knowledge extraction (KE) such that the regular expression (RE) unit of a domain is defined and integrated for any new domain.

Acknowledgements

This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an operating grant (OGP-0194134) and a University of Windsor grant.

References

[1] J. L. Moore, C. Dickson-Deane, and K. Galyen, "E-learning, online learning, and distance learning environments: Are they the same?" *The Internet and Higher Education*, vol. 14, no. 2, pp. 129–135, 2011.

[2] U. o. Windsor. (2014) https://blackboard.uwindsor.ca/. [Online]. Available: https://blackboard.uwindsor.ca/

[3] I. Arroyo, R. Walles, C. R. Beal, and B. P. Woolf, "Tutoring for sat-math with wayang outpost," in *Advanced Technologies for Mathematics Education Workshop*, 2003.

[4] R. K. Atkinson, S. J. Derry, A. Renkl, and D. Wortham, "Learning from examples: Instructional principles from the worked examples research," *Review of educational research*, vol. 70, no. 2, pp. 181–214, 2000.

[5] R. Lukasenko, "Development of student model for support of intelligent tutoring system functions." Ph.D. dissertation, Faculty of Computer Science and Information Technology, Riga Technical University., 2012, summary of Doctoral thesis submitted to Institute of Applied Computer Systems.

[6] Y. Wang and J. Beck, "Class vs. student in a bayesian network student model," in *Artificial Intelligence in Education*. Springer, 2013, pp. 151–160.

[7] K. Chrysafiadi and M. Virvou, "Student modeling approaches: A literature review for the last decade," *Expert Systems with Applications*, vol. 40, no. 11, pp. 4715–4729, 2013.

[8] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, and D. Steinberg, "Top 10 algorithms in data mining." *Knowledge and Information Systems*, vol. 14.1, pp. 1–37, 2008.

[9] S. SOMYUREK, "Student modeling: Recognizing the individual needs of users in e-learning environments." *International Journal of Human Sciences*, vol. 6,2, pp. 429–450, 2009.

[10] C. Romero and S. Ventura, "Educational data mining: a review of the state of the art," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews.*, vol. 40(6), pp. 601–618, 2010.

[11] R. Chaturvedi and C. I. Ezeife, "Data mining techniques for design of its student models." in *Fifth ACM International Conference on Educational Data Mining-EDM, June 21 - 23, Chania, Greece.*, 2012, pp. 232–233.

[12] M. Yudelson and P. Brusilovsky, "Navex: Providing navigation support for adaptive browsing of annotated code examples." in *In Proceedings of 12th International Conference on Artificial Intelligence in Education, AIED.*, 2005, pp. 18–22.

[13] L. Li and G. Chen, "A coursework support system for offering challenges and assistance by analyzing students web portfolios." *Educational Technology & Society*, vol. 12,2, pp. 205–221, 2009.

[14] R. Hosseini and P. Brusilovsky, "Example-based problem solving support using concept analysis of programming content," in *Intelligent Tutoring Systems*. Springer, 2014, pp. 683–685.

[15] T. Gog and N. Rummer, "Example-based learning: Integrating cognitive and social-cognitive research perspectives." in *Edu. Psych. Rev., 22*, 2010, pp. 155–174.

[16] A. Renkl, "Toward an instructionally oriented theory of example-based learning," *Cognitive Science*, vol. 38, no. 1, pp. 1–37, 2014. [Online]. Available: http://dx.doi.org/10.1111/cogs.12086

[17] R. E. Mayer and R. Moreno, "Nine ways to reduce cognitive load in multimedia learning." *Educational psychologist*, vol. 38, no. 1, pp. 43–52, 2003.

[18] R. Chaturvedi, "Task-based example miner for intelligent tutoring systems," Ph.D. dissertation, School of Computer Science, University of Windsor, 2016.

[19] G. Weber and A. Mollenberg, "Elm-pe: A knowledge-based programming environment for learning lisp." *Proceedings of ED-MEDIA 94–World Conference on Educational Multimedia and Hypermedia (Vancouver, British Columbia, Canada, June 25-30)*, pp. 557–562, 1994.

[20] A. Davidovic, J. Warren, and E. Trichina, "Learning benefits of structural example-based adaptive tutoring systems," *IEEE Transactions on Education*, vol. 46, no. 2, pp. 241–251, 2003.

[21] R. Hosseini, I.-H. Hsiao, J. Guerra, and P. Brusilovsky, "What should i do next? adaptive sequencing in the context of open social student modeling," in *Design for Teaching and Learning in a Networked World*. Springer, 2015, pp. 155–168.

[22] R. Hosseini, P. Brusilovsky, and J. Guerra, "Knowledge maximizer: Concept-based adaptive problem sequencing for exam preparation," in *Artificial Intelligence in Education*. Springer, 2013, pp. 848–851.

[23] B. Mokbel, S. Gross, B. Paassen, N. Pinkwart, and B. Hammer, "Domain-independent proximity measures in its." in *Sixth ACM International Conference on Educational Data Mining-EDM 2013, July 6 to 9, 2013, Tennessee, USA*, 2013, pp. 334–335.

[24] R. J. Rovinelli and R. K. Hambleton, "On the use of content specialists in assessment of criterion-referenced test item validity." in *Annual Meeting of American Educational Research Association (60th, San Francisco, California). April 19-23.*, 1976.

[25] M. Yudelson, P. Brusilovsky, and V. Zadorozhny, "A user modeling server for contemporary adaptive hypermedia: An evaluation of the push approach to evidence propagation," in *International Conference on User Modeling*. Springer, 2007, pp. 27–36.

[26] C. Ezeife, *Problem Solving and Programs with C*. Nelson Thomson Learning Ltd., 2010.

[27] J. Friedl, *Mastering Regular Expressions*. OReilly Media., 2006.

[28] T. Pang-Ning, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2005.

[29] P. Jaccard, *Etude comparative de la distribution florale dans une portion des Alpes et du Jura*. Impr. Corbaz, 1901.

[30] R. W. Hamming, "Error detecting and error correcting codes," *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950.

[31] R. Chaturvedi and C. Ezeife, "Mining relevant examples for learning in its student models," in *2014 IEEE International Conference on Computer and Information Technology*, 2014, pp. 743–750.

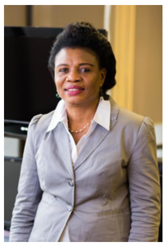
[32] R. Payam, T. Lei, and L. Huan, "Cross-validation," in *Encyclopedia of Database Systems, Springer US.*, M. T. O. Edited by Ling Liu, Ed., 2009, pp. 532–538.

[33] Z. Markov and D. Larose, *Data mining the web - Uncovering Patterns in Web Content, Structure and Usage*. John Wiley, 2007.

[34] R. Chaturvedi, V. Brar, J. Geelal, and K. Kong, "Concept extraction: A modular approach to extraction of source code concepts," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 2018, pp. 1860–1866.

Biography

Ritu Chaturvedi completed her Ph.D. in May 2016 from the school of Computer Science, University of Windsor. Ever since she graduated, she has held academic positions in reputable Canadian Universities such as University of Toronto and University of Guelph, where she currently teaches as an Assistant Professor. Rituâs research interests are in Data Mining and Predictive modeling, especially in educational data mining that caters to web-based tutoring systems such as Intelligent Tutoring Systems. She has authored several technical publications such as those in IEEE conferences and IGI Globalâs International Journal of Data Warehousing and Mining.



Christie I. Ezeife received her Ph.D in Computer Science from the University of Manitoba, Canada in 1995. She has held academic positions at various universities including her current University of Windsor where she has been a Full Professor of Computer Science since 2009. She has graduated over 40 Ph.D and MSc students. Her research interests include exploring efficient techniques for physical database design, mining, recommendation and their application to database systems including data warehouses and web data. She has authored several comprehensive articles in reputed journals such as ACM Computing Surveys, as well as two books on Problem Solving and Programs with C by Thomson Learning Publishers, which had been successfully used for teaching first year students for many years.