Mining Relevant Examples for Learning in ITS Student Models

Ritu Chaturvedi and C.I.Ezeife*
School of Computer Science
University of Windsor
Windsor, Canada
Email: rituch@uwindsor.ca and cezeife@uwindsor.ca

***Abstract*—An Intelligent Tutoring System (ITS) provides direct customized instruction or feedback to students while they perform a task in a tutoring system without the intervention of a human. One of the main functions of an ITS system is to present its students with course materials that are most appropriate to their current knowledge of domain concepts, example being one of the course materials. ITS systems typically compare and analyze student model (SM) components for student's current knowledge of concepts (main topics, e.g. scanf in C programming) that are required to understand the next example (e.g. codes for scanf) suitable for learning a task (e.g. write C code to read 2 integers from the keyboard). Existing systems such as NavEx and PADS perform an exhaustive matching of student knowledge level with all examples in the database.**

This research proposes a task-based technique for managing and classifying examples for more effective retrieval of relevant examples for learning a task. We propose a system called EASK for translating task and example solutions into concepts for similarity matching, which is more readily available, easily extendible and adaptable to other domains. Examples and tasks are represented as vectors of weights computed with term frequency measure TFIDF that signify the importance of a concept for an example. Examples most similar to a task are found by using a classification method called k-NN, which finds the closeness between different objects such as examples and tasks using cosine similarity measure and selecting the k objects (examples) with highest similarity scores. As a by-product, k-NN also predicts the class label (difficulty level) of the task. Our proposed model achieves this prediction with 89% accuracy.

***Index Terms*—adaptation, data mining, classification, example-based Learning, student model**

I. Introduction

An ITS is a computer system that provides direct customized instruction or feedback in real-time to students using tutoring systems, without the actual physical presence of a teacher. Functions of an ITS include adaptation and intelligence. Adaptation is in terms of navigation (e.g. display of material to be read next) and presentation (e.g. presenting the material of exact difficulty). Intelligence shown by the tutoring system is based on the objectives for which the ITS is designed (e.g. providing support to students through hints or examples at the appropriate time).

Every ITS system has modules which assist in meeting the ITS objectives. The modules consist of the domain or expert module defining knowledge to teach (e.g. correct solutions of every example used in the ITS), tutor module defining knowledge of how to teach (e.g. giving instructions to students when they make a mistake), student module defining knowledge of when to teach (e.g. student S's current mastery in concept c indicates that s is ready for lesson2) and a user interface module which allows a student to interact with the first 3 modules. Student module of the ITS stores information about each student using the ITS in a student model. A student model (SM) is an approximate, partial, mainly qualitative representation of the learner's knowledge about a specific domain [1]. It can store student attributes that are static (e.g. learning style of a student), dynamic (e.g. student's knowledge level on concepts) or affective (e.g. emotional state of the student). 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), ontologies or more function-specific network structures such as Bayesian networks [1]. An entity relationship model of an RDB shown in Figure 1 allows for a simple representation of data on independent entities (in rectangles) such as students, concepts and examples and their relationships (in diamond boxes) and yet, complex queries can be carried out on them. Since data stored in an RDB is held in separate tables for each entity and relationship, it caters for future requirements. For example, all domain concepts and their pre-requisites that can possibly be used in the future can be added to the RDB, although the current examples and SM do not use them.

Our SM is represented as a relational database that stores for each student, his/her current knowledge on the domain concepts, as shown in figure 1. Each example / task has one or more solutions and covers one or more concepts. A concept is either a pre-requisite for other concepts or requires other concepts as pre-requisites (shown as recursive relationship). SM data can now be used as input to data mining techniques such as classification to extract information about students and provide them guidance when needed.

K-nearest-neighbor (k-NN) classifier takes as input a parameter k (k=number of neighbors), a test sample t, a set of labeled training samples (e.g. set of existing examples with a class label of the examples' difficulty level) and a similarity measure (e.g. cosine similarity) for determining the distance in order to perform the following steps :

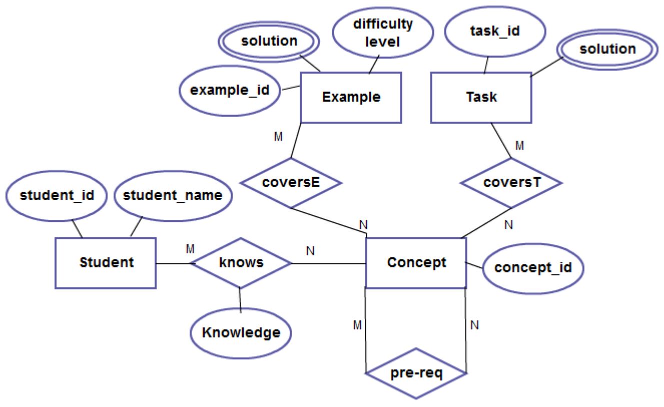


Figure 1. Relational model of students, examples and concepts

a. calculate the distance between the test sample t and all training samples

b. Sort the distances 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

d. find the class label that gets majority (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, assuming k = 5, the difficulty level of task T8 (circle) in figure 2 is predicted to be D (difficult) since 4 of its 5 neighbors (majority) have a difficulty level of D. Our research uses k-NN to search for a task's k nearest examples using TFIDF (Term Frequency – Inverse Document Frequency) and cosine similarity and predict its difficulty level (Section 3 has the details). This generates a static list of most relevant examples for a given task, which is then compared with the current knowledge level of students for concepts required to do that task and appropriate strategies such as 'c5 is a prerequisite of this task, so let's work on examples e1, e4, e5 first' are then generated adaptively (where 'c5' is a concept; e1, e4 and e5 are the suggested examples).

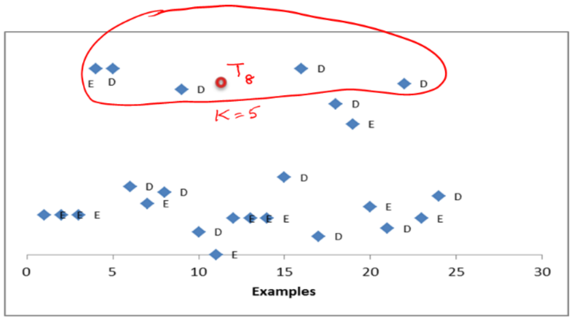


Figure 2. Difficulty level of task T8 using k=5 nearest neighbors is predicted as D (difficult).

II. Related Work

A survey on existing methodologies [2], [3], [4]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 : Concept extraction (CE), Knowledge representation (KR) and Knowledge adaptation (KA). KR as explained in section 1 defines methods and structures used to represent student knowledge. Although many other existing systems were extensively studied, this research compares CE, KR and KA of two systems NavEx[4] and PADS[2] that were very similar in their objective of adaptively finding and presenting most relevant examples to students, yet very different in the methodologies used. CE can be done either automatically or manually. NavEx[4] uses a parser to automatically extract concepts from C programs and then proposed and used a splitting algorithm to develop a hierarchy of concepts and their pre-requisites. PADS [2] is a part of a web-based tutoring system that uses a method called IOC [5], 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 examples as well). Then, an index value is calculated for each concept i in exercise k

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

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. KR in NavEx [4] is Boolean – 'known' or 'not known' for each concept. In PADS [2], progress of a student on a concept is stored as 1 (low), 2 (medium) and 3 (high). The adaptation (KA) in NavEx simply matched 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 [2] used a decision tree method 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: Source 1: SM of S for mastery level of main concepts of exercise, sub-concepts of the exercise and of algorithm analysis; Source 2: expert opinion on algorithm complexity and number of lines in program code of each exercise; Source 3: web logs for S's grade in the previous exercise and login times during the last 2 weeks. 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). Mokbel et al. [3] divide their solution's syntax trees into sub graphs using spectral clustering and then measure the proximity between solution parts. Each solution part is represented as a vector of TFIDF weights. The methods used by the existing systems [2], [3], [4] are very subjective, dependent on a complex knowledge of experts on areas such as parser generation and are less-focused as they generate a broad set of examples independent of any task . For example, PADS uses expert opinion for 3 of its 7 feature attributes. Both NavEx and PADS compare their SM with N examples (N = #examples in the database). Our approach compares only k out of N examples. NavEx and Mokbel et al. [3] require each example to be represented as syntax tree , although our research suggests that to present relevant examples, it is sufficient to extract concepts and evaluate if those concepts are relevant to the examples eliminating the need for a syntax tree representation. 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. Our proposed research represents domain examples as vectors of concepts so that a weighting mechanism such as TFIDF can be applied to them, and thereafter a prediction model based on a proximity measure such as cosine similarity can be used to compare the closeness of a given task with the examples in the database. KA in NavEx[4] 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 [2] 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. ask a question on the example) can be a solution to this problem. Further, social 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.

III. Overview of the system / techniques used by EASK

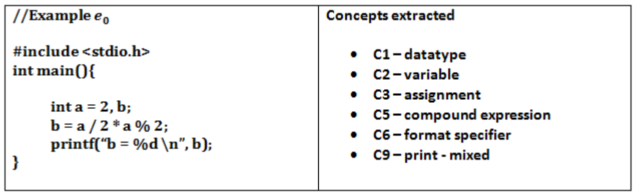
In this research, a task and an example are defined as shown in definitions 3.1, 3.2 and 3.3.

Definition 3.1: A task is a question on a subject such as a C programming language to be answered by a student. An example of a task t1 is "Write a C program to find the area of a triangle".

Definition 3.2: A task solution is the solution of the question asked in a task. A task solution for task t1 is "float base, ht, area; printf("Enter the base and height"); scanf("%f%f", &base, &ht); area=base*ht; printf("Area = %.2f", area);".

Definition 3.3: An example is a solution of questions on the subject. Some examples that may be relevant to task t1 above are e2: "float a; scanf("%f", &a);", e15: "int a, b, c; scanf("%d%d",&a, &b); c = a*b;" and e0 as shown in figure 3.

Concept extraction (CE) is done using IOC [5], where a number of experts indicate whether or not a concept belongs to an example. CE extracts the concepts covered by the task a student is assigned and by all the examples in the database. Figure 3 shows an example and concepts extracted from it. Then, index value I_{ik} is calculated to measure the importance of this concept for the example. A value of Iik> 0.6 indicates that example k requires concept i. We use a relational model for knowledge representation.



//Example e_0	Concepts extracted

#include <stdio.h>
int main(){
int a = 2, b;
b = a / 2 * a % 2;
printf("b = %d \n", b);
}
	• C1 – datatype
	• C2 – variable
	• C3 – assignment
	• C5 – compound expression
	• C6 – format specifier
	• C9 – print - mixed

Figure 3. An example and its concepts extracted using CE

Each student data has a static content such as id, name and a dynamic content such as knowledge level on a domain concept. Since there are more than one concepts that a student can work on, there is an M:N relationship (knows) between students and concepts as shown in figure 1. Similarly, each concept is a pre-requisite to one of more other concepts. This is represented using a recursive M:N relation between concepts and their pre-requisites (related). For example, concept C3 (arithmetic operators) is a pre-requisite of concept C4 (expressions). Each example requires one or more concepts and a concept can be covered by one or more examples. This is represented by a M:N relationship (coversE). A similar relation exists between task and concept (coversT). For KA, examples can be presented to students either independent of any given task or in context of a given task T. Our research focuses on the latter – task-based retrieval of relevant examples(Section 3 has a detailed algorithm). In the first step of the algorithm, a weighting mechanism using TFIDF weights is used to generate a static list of examples closest to the task T. Although a Boolean value can be used in order to denote the presence or absence of a concept in an example, a weight (usually between 0 and 1) better signifies the importance of a concept for an example [5]. Each example e_i in the database (and/or task) can be represented as a vector of concepts c_1 to c_n (where n = total number of concepts in the database) denoted as $ei = [wi1, wi2, \ldots win], wij(j = 1..n)$ represents a weight assigned to concept j of example e_i. These weights can be assigned using methods such as TF (Term Frequency) and TFIDF. TF is the number of times a concept (more commonly known as a term in IR) occurs in an example computed as

$$w_{ij} = TF_{ij} = n_{ij} \qquad (2)$$

where n_{ij} is the number of times concept $c_j$occurs in example e_i. TFIDF [6] uses TF combined with inverse document frequency (IDF obtained by dividing total number of examples in the database by number of examples that contain the concept and then taking its logarithm, computed as

$$w_{ij} = TFIDF(c_j,e_i) = TF_{ij} * log\left(\frac{N}{n_j}\right) \tag{3}$$

where N is the total number of examples in the database and n_j is the number of examples that contain concept c_j. TF calculates weights locally, taking into account a specific concept and an example, whereas IDF component of the TFIDF method computes the weights globally by considering the proportion in which a concept occurs with respect to the total number of examples in the database (N) that contain the concept. Task T, is also represented using a similar vector of weights for each concept as $[w_{t1}, w_{t2}, ..w_{tn}]$. For example, task T_8 is defined as "Write C code to compute and print the sum and product of 3 integers". Task solution of T_8 and examples e_1, e_4, e_{15} and e_{16} are shown in table 2 and TFIDF weights of task (T_8 and examples (e_1 and e_{16}) are shown in table 1, where $C_1..C_{10}$ are 10 concepts used in our database.

Table I
TFIDF WEIGHTS FOR TASK T_8 AND EXAMPLES E_1 AND E_{16}

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
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

Exact matching is not possible when using TFIDF weights (unlike if Boolean values were used), and therefore, a proximity measure such has Euclidean distance or cosine similarity is used to find the proximity of a task with examples in terms of concepts required for a task / example. Cosine similarity is chosen over Euclidean distance in this research since Euclidean distance between task T and example e_i measured as

$$d(T_j, e_i) = \sqrt{\sum_{j=1}^{n}(T_j - e_{ij})^2} \tag{4}$$

where n = length of the vectors (= total number of concepts in the database) greatly depends on the length of the vectors being compared [6]. For example, when comparing the similarity between a task T and 2 examples of different lengths e_s (small) and e_l(large), there will be a great distance between T and e_l as compared to T and e_l, although they both could be equally relevant to the concepts . Cosine similarity between task and example e_i is computed as

$$CS(T, e_i) = \frac{\sum_{j=1}^{n}(w_{ij} * w_{tj})}{\sqrt{\sum_{j=1}^{n} w_{ij}^2} * \sqrt{\sum_{j=1}^{n} w_{tj}^2}} \tag{5}$$

Cosine similarity used with TFIDF weights best meets the objective of finding those examples that are most relevant to the current task because it assigns a higher weight to those concepts that can best distinguish the relevance of examples. It gives a similarity of the topic of the contents in the task and examples, not the similarity of the size of the contents. It is used in the area of information retrieval but has not been explored in ITS systems yet. For example, if the task in hand is T_8 and examples compared are e_1, e_4, e_{15} and e_{16} (table 2). The Euclidean distance between T_8 and e_1, e_4, e_{15} and e_{16} (table 3) indicates that example e_1 is closer in distance to task T_8 than example e_{15}, although, in terms of concepts required, table 3 shows that e_{15} is more relevant to T_8. The distance values are also incorrectly indicative of the fact that examples $e_{15}$and e_4 are exactly the same. Cosine similarity between T_8 and the examples indicates that examples relevant to T_8 (in order of similarity) are e_{16}, e_{15}, e_4 and e_1. Some examples may result in the same cosine values or very similar (<=0.05) values when compared to a given task. To break such a tie, the formula we use for similarity is a modified cosine method (MCS) that multiplies the cosine value to the number of matching concepts between the example and the task (we call it TR for tie resolution). Equation 6 defines MCS between a task and an example.

$$MCS(T, e_i) = CS(T, e_i) * TR \tag{6}$$

Finding the k-largest example-task MCS values is similar to finding the k nearest neighboring examples to the current task. This led us to use the k-NN prediction method to achieve 2 objectives: find k most relevant examples for a task using modified cosine similarity and TFIDF weights and predict the difficulty level of the current task being pursued. k-NN is a simple, non-parametric method which takes a test data point x (e.g. a vector of weights as in our example) and finds the k points closest to x using some function (e.g. cosine similarity). It then classifies x based on majority voting on classes of k. For example, if x = task T_8 , the k (=4) examples closest to x using cosine similarity are e_1, e_4, e_{15} and e_{16} and the difficulty level of the examples respectively is (easy, difficult, difficult, difficult), then task T_8 is predicted to be a difficult task. Although the prime objective of this research is to find the most relevant examples, it meets another important objective of predicting the difficulty level of the task. This can assist students in self-managing to succeed in the task (e.g. student might want to read more resources since the task is predicted as a difficult task). Choice of k is important in k-NN for the results to be accurate. Research recommends the value of k to be $\sqrt{n}$ for effective results[6]. In order to evaluate our model and to estimate its performance, a special case of k-fold cross validation (CV) called leave-one out (LOOCV) is used. In k-fold CV, the dataset is first divided into k partitions or folds. Then k iterations are performed such that each iteration uses k-1 folds for training the model and 1 fold for validating it. K-fold CV performs very well if the dataset is large and the test data has the same or similar distribution as the training data. LOOCV is a special case of k-fold CV where k equals the total number of samples in the dataset (N). Therefore, in each iteration, one sample of the dataset is chosen as the test data and rest of the samples (N-1) are used to train the model. This method is widely used when the data set is small and rare [7] and is found to give unbiased performance estimates such as accuracy and f-score.

Table II
TASK T_8AND EXAMPLES e_1, e_4, e_{15} AND e_{16}

Example / Task	Code

e_1	int a;
	scanf("%d", &a);

e_4	int a,b,c;
	scanf("%d%d%d", &a, &b,&c);

e_{15}	int a,b,c;
	c = a*b;

e_{16}	int a,b,c;
	c = a*b;
	printf("c = %d \n", c);

T_8	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);

Table III
EUCLIDEAN AND COSINE SIMILARITY FOR TASK T_8

	Euclidean	Cosine
e_1	2.4	0.2
e_4	3.16	0.21
e_{15}	3.16	0.27
e_{16}	2	0.4

IV. EASK FOR KNOWLEDGE ADAPTATION

This paper proposes an EASK system which builds a list of examples that are most appropriate and useful to students working on a task while using an ITS. This is accomplished using algorithms that are simple, quick, accurate and easily adaptable across other domains. Four datasets defining student models, concepts, examples and tasks and their relationships are used as input by the main algorithm EASK. Figure 1 depicts the relationship between these datasets. EASK first generates concepts of each example and task using CEET, then calls GREPD that uses a k-NN model to generate a list of examples closest to the task and thereafter, calls GAL that analyzes student models (SM) by computing average score for each example in this list using concept scores in the SM in order to present them with examples that will help them learn concepts that they are not good at yet. For example, if a student asks for examples while attempting task T_8, and his grades on concepts named variable, assignments, simple expression and print are 90, but on concept scanf is 55, it indicates that this student needs to study examples on scanf to improve his/her grades on it, and therefore, examples e_1 and e_4 will be presented but not e_{15} and e_{16}.

A. Datasets (Entities and Relationships)

Four entities used in this research and their relationships are listed below.

Student	dataset: SM of 10 students: Attributes: (studentId, studentName, grades in concepts C1 to C11).
Concept	dataset : 11 concepts from the domain of C programming extracted using IOC[5] : {Datatype, Variables, Assignment, Simple expressions, compound expressions, format specifiers (FS), print (simple messages), print (FS only), print (messages and FS combined), scanf(1 FS), scanf (n FS)}
Example	Dataset: A set of 20 examples and 5 tasks. Attributes : (ExampleId, ExampleTitle, {Solution}, Difficulty level)
Task	Dataset: A set of 5 tasks. Attributes: (TaskId, TaskTitle, {Solution})
knows	: many-to-many relationship between students and their knowledge on concepts stored as a relationship attribute called knowledge.
pre-req:	many-to-many recursive relationship of concept with itself, either in the role of a pre-requisite to another concept or as concept that requires other concepts as pre-requisites.
coversE:	many-to-many relationship of an example and concepts it covers or requires.
coversT:	many-to-many relationship of a task and concepts it covers or requires.

B. Proposed Algorithm EASK

Algorithm EASK (Examples Adaptable to Student's Knowledge) is described below in three steps.

Step 1: CEET(E,T) - Represent each example in the database as matrix of concepts : CEET (Concept Extraction of Examples and Task) takes each example in E and task T, and tokenizes them into n concepts (n = total number of concepts used in the database) and their pre-requisites using IOC method [5], so that they are now represented as vectors of size n. Each element of the vector stores the number of times a concept appears in it (e.g. if concept 1 appears once and concept 2 appears 3 times in example e_4, then e_4's vector will be [1, 3, .]).

Input: Examples E and task T

Output:
1. 25 X 11 matrix of examples and concepts CM_EX
2. 1 x 11 matrix of task and concepts CM_T

Algorithm 1 CEET : Concept Extraction of Example and Task

```
1. Extract concepts and their pre-requisites for each example in E and task T using expert opinions to decide the concepts covered by each example. Five experts gave opinions on whether a concept should belong to an example.
2.Use equation 1 to calculate index value $I_{ik}$ to measure the importance of this concept for this example. If $I_{ik}$> 0.6, then it is assumed that example k requires concept i.
```

Step 2: GREPD (CM_EX, CM_T, k, DL) - Find similarity between each example and task using k_NN classification : GREPD (Generate Relevant Examples and Predict Difficulty of a task) takes as input the concept matrix of examples CM_EX and concept_matrix of task that the student is working on currently CM_T, an integer value k and a matrix DL that stores the expected difficulty level of each example assessed by an expert. It then computes the TDIDF weights using equations 2 and 3 for each example and task, finds the similarity between the task and each example using a modified cosine similarity formula (MCS) given as equation 6 and generates a list of k examples closest to the task using k_NN reviewed in section 1. It also predicts difficulty level of the task by using difficulty level of its neighbors and assigning a level to the task that maximum number of its neighbors have.

Input: CM_EX, CM_T, k, DL

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. List of k examples L1
2. Predicted difficulty level of task

Algorithm 2 GREPD : Generate Relevant Examples and Predict Difficulty of a task

```
1. Compute TFIDF weights for each row of example concept matrix CM_EX and of task concept matrix CM_T using equation 3 (details in section 3).
2. Compute modified cosine similarity ($MCS_T$) between CM_T and each row of CM_EX using equation 6 (details on calculating MCS in section 3).
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’.
```

Step 3: GAL(L1, S) - Generate a list of most appropriate examples matching student’s current knowledge on concepts: GAL (Generate Adaptable List of examples) takes list L1 of size k (<=N) generated in step 2 by algorithm GREPD and student’s SM to generate a new list of examples that are adaptable to the student’s current knowledge of concepts stored as grades out of 100. Each concept’s grade is converted into a score using the following scheme: score=2, if grade >=80; 1 if between 50 and 80, 0 if < 50 and -1 if null (null indicates that the student has not been graded on it yet). For each example in the list L1, an average score is computed. An example with a low score indicates that the student needs to work on its concepts and therefore is included in the output list of suggested examples.

Input: L1, S

Other Variables:
1.C_T: Set of all concepts of an example
2. sum_s: Cumulative sum of student scores on concepts
3. avg_sj: Average student score on concepts of example j

Output : AL = List of examples most appropriate for a student’s current mastery of concepts

Algorithm 3 GAL :Generate Adaptable List of examples

```
1. for each student s in S
2.    for each example j in list L1
3.       Let C_Tj = set of all concepts of j
4.       Let zj = number of concepts in C_Tj = | C_Tj |
5.       sum_s is initialized to 0
6.       for each concept m in C_Tj
7.             Add score achieved by s in concept m
               to get cumulative concept sum sum_s
8.       end for
9.       compute average student score for j as
         avg_sj = sum_s / (zj * 2)
10.   end for
11.   Sort avg_sj in descending order and store in AL.
12. end for
```

C. Example Application of EASK on Sample Data

Example 1:
Problem Definition: Generate a list of examples AL for students s1 and s2 for task T_8 (shown in table 2). SM for student s1 = [2,2,1,0,1,-1,1,0,0,1] and for student s2 =[2,-1,-1,-1,1,-1,-1,1,-1,-1,1]. Each entry in the SM is a score on a concept; score = 2 implies grade >=80, score = 1 implies grade between 50 and 80, score = 0 implies grade < 50 and score = -1 implies that student has not been graded on these concepts yet.

Solution:
Step 1: CEET Input: E = 25 examples and task T = T_8.

• It extracts concepts for task T8 as c1,c2,c3,c5,c8,c9,c11. Concept c4 is a pre-requisite of c5, c6 is a pre-requisite of c8 and c10 is a pre-requisite of c11, they are also included in the set of concepts required for task T_8. So the final matrix CM_T for T_8 is [1,1,1,1,1,0,1,1,1,1,1]. The concept_matrix CM_DB consisting of 25 rows (one for each example) and 11 concepts is also used as input but is not shown here due to space constraints.

Step 2: GREPD
Input to GREPD is CM_T, CM_EX, k=5 and DL = [EEEEDDEDEDEEEEEDDDDEEDDEED]

• TFIDF for each example in the database is computed using equation 2. TFIDF weights for examples e_1, e_{16} and task T_8 are shown in table 2.

• MCS values between 25 examples and task T_8 are computed using equation 4, and are shown for examples e_1, e_{16} and task T_8 in table 3. These values are sorted in descending order (higher the MCS, closer is the example) and top k values are picked as k-nearest neighbors of T_8 . With k=5, L1 = { $e_4, e_5, e_9, e_{16}, e_{22}$}. Difficulty level of examples e_5, e_9, e_{16}, e_{22} is 'D' and of example e_4 is 'E', therefore the difficulty level of T_8 is predicted to be 'D'.

Step 3: GAL Input to GAL is L1 = { $e_4, e_5, e_9, e_{16}, e_{22}$} from GREPD and student models for s1= [1,2,2,1,0,1,-1,1,0,0,1] and s2 = [2,-1,-1,-1,1,-1,-1,1,-1,-1,1].

For student s1

- for example e_4
  - C_T$_{e4}$ = {c1, c11}= {1,1}.
  - z$_{e4}$ = 2
  - sum_s1$_{e4}$ = 2
  - avg_s1$_{e4}$ = 2 / 4 = 0.5
- for example e9
  - C_T$_{e9}$ = {c1, c6, c9, c11}= {1,1,0,1}.
  - z$_{e9}$ = 4
  - sum_s1$_{e9}$ = 3
  - avg_s1$_{e9}$= 3 / 4 = 0.375

Similarly, scores computed for each example in L1 = [$e_4 : 0.5, e_{22} : 0.6, e_{16} : 0.45, e_5 : 0.4, e_9 : 0.375$]

• AL for s1 is {e_9, e_5, e_{16}}, indicating that s1's knowledge on concepts of e_9 is the least and therefore he/she needs to study it to be able to succeed in task T_8 A similar reasoning holds for examples e_5, and e_{16}.

For student s2

- for example e_4
  - C_T$_{e4}$ = {c1, c11}= {1,-1}.
  - z$_{e4}$ = 2
  - sum_s1$_{e4}$ = 0
  - avg_s1$_{e4}$ = 0 / 4 = 0
- for example e_9
  - C_T$_{e9}$ = {c1, c6, c9, c11}= {1,1,1,-1}.
  - z$_{e9}$ = 4
  - sum_s1$_{e9}$ = 2
  - avg_s1$_{e9}$= 2 / 8 = 0.25

Similarly, scores computed for each example in L1 = [$e_9 : 0.25, e_{22} : 0.2, e_5 : 0, e_4 : 0, e_{16} : -0.1$]

• AL for s2 is {e_{16}, e_{22}, e_9}, indicating that s2 has to work on concepts covered by these examples to succeed in task T_8.

Scores for each of the ten students in the database are computed for every example in list L1. As shown in table 4, each student's list of suggested examples is adaptive to the grades achieved in concepts covered by the 5 examples closest to task T_8. A final list of 3 examples is suggested to each student. SM for student s5 is a special case because his/her average score in each example is the same and therefore every example in the list is presented.

Table IV
FINAL LIST OF SUGGESTED EXAMPLES FOR TASK T_8 FOR EACH STUDENT

	e_4	e_5	e_9	e_{16}	e_{22}	Suggested list of examples to study for task T_8
s1	0.5	0.4	0.4	0.45	0.5	e_9, e_5, e_{16}
s2	0	0	0.3	-1	0	e_{16}, e_{22}, e_9
s3	0.5	0.5	0.5	0.1	0.2	e_{16}, e_{22}, e_9
s4	0	0	0.4	0.1	0.2	e_4, e_5, e_{16}
s5	0.5	0.5	0.5	0.5	0.5	$e_4, e_5, e_9, e_{16}, e_{22}$
s6	-0.5	-0.5	-0.1	0.4	0.4	e_4, e_5, e_9
s7	0.25	0.25	0.3	0.4	0.26	e_4, e_5, e_9
s8	0	0	0.4	0.6	0.7	e_9, e_{16}, e_{22}
s9	0.25	0.25	0.3	0.5	0.4	e_4, e_5, e_9
s10	0	0	0.4	0.8	0.8	e_9, e_{16}, e_{22}

V. RESULTS AND EVALUATION OF EASK

A. Evaluation of CEET

For Concept Extraction (CE), CEET uses a methodology partly similar to existing methods [2], [3], [4] in terms of extracting features as concepts but our method is more efficient since it does not need to consider their syntactic relationships and therefore, does not have to encode solution codes as syntax trees. Instead of using complex knowledge of experts on parsers and syntax trees, our research uses teaching assistants(TA) or graduate assistants(GA) who have majored in the domain (e.g. TA in a course on C programming) as trained experts to generate the concept matrix. This is in support of the need for a more accessible and adaptable knowledge representation such as the proposed model.

B. Evaluation of GREPD

To evaluate GREPD, leave-one-out method of cross-validation and measures such as accuracy and f-score are used. In each iteration, one sample (ith example or task) from the complete dataset (of size N) is considered to be the test data and the rest of the (N-1) samples are taken as training data and are given as input to GREPD. GREPD then finds the k-nearest neighbors of the test data and predicts its difficulty level. Accuracy A measures the ability of the model to match the actual value of the class label (e.g. easy predicted as easy and difficult predicted as difficult) and is defined as

$$A = \frac{\#correct\_predictions}{\#predictions} \tag{7}$$

F-score F combines $precision$ (how many of the actual true values predicted as true / total number of values predicted as true) and $recall$ (how many of the actual true values predicted as true / total number of true values) and is defined as

$$F = \frac{2 * precision * recall}{(precision + recall)} \tag{8}$$

The best accuracy value computed for the proposed modified cosine (MCS) was 88% and f-score was 89%, for k=9. A comparative performance evaluation is shown in figure 4 for 4 different values of k. It indicates that Euclidean distance is certainly not a good similarity measure for finding k nearest neighbors for an example or a task. Although these results for cosine similarity are comparable to our method MCS, the tie resolution used in MCS ensures that the retrieved examples were more appropriate in terms of matched concepts with the current task. For example, e_4, e_5, e_{22} have CS of 0.7129, 0.7129 and 0.6569 when compared with task T_8. But since e_{22} matches more concept values with T_8 (8), its MCS is higher than e_4 and e_5 and therefore, it gets retrieved as a more relevant example than e_4 and e_5.

C. Evaluation of GAL

GAL algorithm is in its early stages and results of a class of 10 students indicate that GAL works well in presenting examples most relevant to their current knowledge of concepts, as shown for 2 students in section 4.3.

D. Evaluation of EASK

EASK that uses algorithms GREPD and GAL has an overall reduced complexity of $O(|S| * k) = |S| * log(n)$, where k , as compared to NavEx[4] and PADS[2], which have a complexity of $O(|S| * N)$. Using a relational model for KR (knowledge representation) enables us to use the power of SQL, Oracle Apex and PL/SQL stored procedures for step 2 of this algorithm.

VI. Conclusions and Future Work

We propose a model to facilitate the well-adapted theory of example-based learning [8] – a model that is simple, fairly accurate, easily extendible and adaptable to other domains. It is highly important for ITS to recommend appropriate examples to students in real-time for effective and improved learning. Our model makes significant contributions in this direction by:

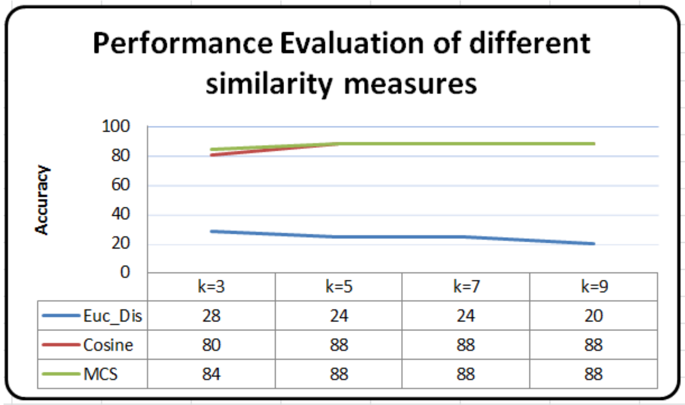
• Allowing for a simple and quick build of concept matrix using experts that are easily accessible and highly trained in the domain.

• Effectively searching for only those examples that cover concepts similar to the current task, thereby reducing the number of relevant examples.

• Predicting the difficulty level of the current task.

• Computing student scores for each example in the relevant list using a novel formula to recommend only those ones that are adaptable to the student's current scores.

Although our model predicts the difficulty level of a task, we do not use this information in our current algorithms. Nevertheless, it could prove very useful for ITS to know the difficulty level of a task or an example, in areas such as developing curriculum strategies adaptable to each SM, without having to go through experts. The proposed algorithms do not consider the sequence in which examples are attempted by students and the impact it has on the success of a task and therefore this could be a potential future direction. For example, if Example1 is followed by Example 3, then Task1 = Successful with a 65% confidence. Social adaptive features such as student 'likes' for an example can also be considered.



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

Figure 4. Performance evaluation of different similarity measures

VII. 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] 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.
[2] 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.
[3] 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.
[4] 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.
[5] 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, SanFrancisco, California). April 19-23.*, 1976.
[6] Z. Markov and D. Larose, *Data Mining the web - uncovering patterns in web content structure and usage.* Wiley, 2007.
[7] R. Payam, T. Lei, and L. Huan, "Cross-validation," in *Encyclopedia of Database Systems, Springer US.*, M. T. z. Edited by Ling Liu, Ed., 2009, pp. 532–538.
[8] T. Gog and N. Rummer, "Example-based learning: Integrating cognitive and social-cognitive research perspectives." in *Edu. Psych. Rev., 22*, 2010, pp. 155–174.