CHAPTER VII

AUGMENTATION OF LMS
WITH SOME ADDITIONAL FEATURES
7.1 INTRODUCTION

Online learning with help of LMS employs many useful features that supplement the traditional classroom teaching. With the rapid proliferation of internet and web2.0 technologies, contemporary LMSs have witnessed an increasing number of adaptive and personalized e-learning features. In section 1.5 and 2.2.5, we have discussed that there is a continuous effort among the researchers to enrich conventional LMS by augmentation of additional features in terms of adaptivity and personalization. Personalizing students’ learning environment enables them to access a unique learning experience based upon their individual needs, rather than receiving instruction through a standard paced curriculum. Many researchers (Paramythis & Loidl-Reisinger, 2003), (Kardan et al., 2014) have pointed out that personalized learning i.e., learning tailored to the specific requirements and preferences of the individual, cannot be achieved, especially at a massive scale, using traditional approaches. The main challenge of the traditional LMSs is providing courses suitable to different learners with different knowledge background.

Adaptive curriculum sequencing, course management and content presentation are key issues to achieve individual learning goal for the learner (Brusilovsky, 2000), (Devedzic, 2003). The main assumption underlying these major adaptive features is that each student is unique. Therefore, the LMS should be able to initialize a student model (learner model) for determining the knowledge level of a student when she registers for a course, or completes a course and wants to move on to the next course, or wants some personalized assistance within the current course. This knowledge about a learner could come from multiple sources like test scores, assessment history, or at run-time, from how she uses and interacts with the system (Self, 1994), (Brusilovsky, 1996), (Esichaikul et al., 2011), (Grubisic et al., 2013). A runtime scaffolding for helping the tutor with personalized information about the learner is therefore helpful for personalized assistance. A learning path is a sequence of keywords that guides the learner to reach to a desired target term from a starting point (generally a known term) by gradually understanding the intermediate unknown terms. Adaptivity can also be provided by enabling a learner to select appropriate learning path that meet her personal requirements and needs and discard those paths, which are not in accordance with
these needs. Some approaches have been already endeavored towards selecting customized learning path that costs least time and effort (Zhao and Wan, 2006), (Alian & Jabri, 2009).

However, all these additional features address a board area of adaptivity and personalization of a learning environment that can have numerous manifestations. Instead of attempting an exhaustive enumeration on all of these, we explore three areas namely learning path, personalized learning, and adaptive leaning, which suffice the broad and partially overlapping categories. This chapter proposes some new techniques towards the above-specified domains to enrich conventional LMS with addition of some intelligent and adaptive features (Sengupta & Dasgupta, 2014), (Sengupta & Dasgupta, 2010), (Sengupta et al., 2012), (Sengupta et al., 2011(a)), (Sengupta et al., 2011(b). The proposed augmented LMS can support a learner with a learning path that recommends a series of term definitions, which could guide her to the understanding of a desired unknown term. A framework for runtime scaffolding is also proposed that assists personalized problem solving for the learners. Finally, the proposed adaptive techniques related to adaptive presentation and adaptive navigation provides a customized learning environment for the learner based on her existing knowledge and choice of learning style.

7.2 LEARNING PATH

A learning path for an unknown term is the sequence of terms that starts from a known term and ends at the unknown term with some intermediate terms, which may be known, unknown, or partially known to the learner. Figure 7.1 describes the idea of a learning path. The left circle in the figure represents the domain of known keywords and the right circle represents the domain of unknown words. Now, if a learner who has the initial knowledge on ‘S’, wants to learn the term ‘D’, she must start from the point ‘S’ of the known domain and will try to reach to ‘D’. This path from source to destination may include several hops in the unknown domain, which indicates the prerequisite knowledge to understand the target term. Our aim is to support the learner with a learning path like S-A-B-D, which would help her to reach the goal.
The non-linear arrangement of information in the form of hypertext has brought a revolutionary change in the teaching learning process (Abdel, 1997). In hypertext documents, links are established in such a way that the user can explore, browse and search for a particular item and can get information regarding relevant associated issues. The problem with self-regulated definition learning is that a learner often gets confused about the starting point of learning since the term definition itself contains multiple hyperlinked keywords that are known or unknown. Therefore, prerequisites required to understand a particular term are often discovered late by several forward and backward hops thorough these unknown terms (by following their links). Therefore, the main challenge is to find out – what are the terms that should be learnt before understanding the target term and what sequence must be followed. In other words, we need a path to follow to reach to the target term by gradually understanding the intermediate unknown terms. The solution to this problem is a recommendation of learning path. In this work, we discuss on two types of learning path- generic learning path (section 7.2.1) and leaner specific learning path (section 7.3.1). A generic learning path is a common solution for all the learners at a particular level who want to learn a particular term. We propose two alternative techniques to construct a generic learning path. The first approach uses FP technique of data mining and the second approach uses a definition tree constructed on a proposed data structure. On the other hand, a leaner specific learning path adapts to individualized criteria of learners. We propose a method for construction of a leaner specific learning path based on the Ant Colony Optimization technique.

7.2.1 Generic learning path using Frequent Pattern (FP) Technique

The Association rule and the FP tree are two popular and effective data mining techniques for discovering new knowledge from large and complex data sets. In the context of e-learning data, association rule can be used to reveal dependencies between the ‘terms’ used in different definitions. On the other hand, a FP can
depict the likelihood of a pattern (set of key words) appearing in a particular term definition. Our proposed technique of learning path construction is divided into two sections. First, a usage graph for FP is built from a set of inter-linked terms used in an available set of definitions. Then we assign weight to the edges of the graph by taking input from the usage of these terms in the definition learning either from the records of a large number of contact mode classes or from the large web based repository of definitions. Secondly, we search for FP of terms by discovering associations between the terms. A recursive searching algorithm is proposed which helps the learner in finding a suitable learning path. Finally, the resultant learning path is a sequence of nodes that starts from a node from known domain and ends at the target term. In this approach, we consider the domain of known words as the set of terms that the learner has already come across in her previous lessons. Thus, a generic learning path is common for all the learners who have completed a particular level of course.

7.2.1.1 Definitions

Let us first define some basic components of our methodology in the context of LMS

a) **Terms**: A concept or entity that has a definition.

b) **Keywords**: Terms used within the definition of another term.

c) **Transactions**: A transaction is a sequence of keywords that are used to define a term. If ‘D’ is a term, which is defined with help of other key words ‘A’, ‘B’, ‘C’ then a transaction ABCD is formed. The keywords are placed in the same order as they appear to define ‘D’.

d) **Association rule**: We define an association between two terms as \( a \rightarrow b \) when the percentage of transactions supports ‘a’ that also supports ‘b’, exceeds a predefined threshold value (\( \sigma \)), called confidence.

Let \( T= \{ABCD,GCDEF,BCFD,XYZV,DFACV\} \) and \( \sigma=50 \)
we check whether \( B \rightarrow C \) holds

Now support for B is written as: Support (B) = 3/5
Percentage of transactions that support B also support C is 2/3
i.e., Confidence (B\( \rightarrow \)C) = 2/3
Since it is greater than the predefined value of $\sigma$, we conclude that the association holds.

e) **Frequent pattern (FP):** A frequent pattern is a sequence of size $n$ that has largest number of terms participating in an association. A frequent pattern is generally discovered by mining through a weighted tree or graph.

### 7.2.1.2 Construction of a usage graph for FP

A usage graph for FP is a weighted directed graph that represents a set of transactions where the vertices are term items of a transaction and the edges represent inter-dependencies among the terms. The weights on the edges represent the frequency count of that sequence within the available dataset. Root is a special vertex that indicates start of any transaction.

Such graphs could be constructed from the web-based repository of the definitions or from the question answer sessions of contact mode classes. For creating transaction from recordings of contact mode classes, we have considered only ‘what is’ type questions, which are commonly answered by term definitions. Let $A=\{B, C, D\}$ be a transaction, representing answer to the question ‘what is $A$?’ with help of the keywords $\{B, C, D\}$. Then a branch is created in the graph from the starting node ‘root’ as $\{B \rightarrow C \rightarrow D \rightarrow A\}$ [Figure 7.2].

![Figure 7.2: A branch in usage graph for frequent pattern](image)

---

183
Let \( S = \{ s_1, s_2, \ldots, s_k \} \) is the set of all terms used in all definitions and \( T = \{ t_1, t_2, \ldots, t_m \} \) is the set of all the definitions. Now since, each \( t_i \) is composed of keywords taken from \( S \), we can say that each \( t_i \subseteq S \). However, a term is said to be independent if it does not use any other keywords in its definition. Independent terms are generally easily understandable by their own definition and they come at the next level to the root in the graph. Links between the terms can form a cycle when a term uses a keyword in its definition that is already used in any other sequence. Every association between the nodes comes with an initial weight of 1. Thereafter, while inserting any sequence \( t_i \to t_k \) to the graph, the frequency of the edge \( (w_{ij}) \) is increased by 1 if it already exists in graph. Weights on the edges are iteratively updated based on the available data set (records from the contact mode classes or web repository of definitions). At every round we calculate the confidence value of each edge and if its frequency crosses the predefined value of \( \sigma \) then we convert the sequence \( \to \) into an association \( \to \). In addition, there is always a trivial association from compound words to its parts e.g., data mining \( \to \) data.

### 7.2.1.3 Finding frequent pattern from the graph

In order to find a learning path for a target term ‘D’, we search for available frequent pattern associated with ‘D’ in the graph. We traverse from the target node towards the ‘root’ until we found ‘S’, a node which belong to the known domain of the learner. Thus, we get a path as \( S -- A -- B -- D \) where ‘S’ is the starting point and ‘D’ is the target term to learn, and ‘A’ and ‘B’ are the intermediate terms that the learner is advised to learn.

### 7.2.1.4 Algorithm for learning path for an unknown term

Input: \( \text{target\_term} \) /*what the learner wants to learn*/
Output: \( \text{result} \) /*a sequence of terms as the learning path*/

\[
\text{LEARNING\_PATH (target\_term)} \{
\text{if not (last\_visited\_word == ROOT || last\_visited\_word == known\_word)}
\text{for each next\_word in ASSOCIATED\_NEIGHBOR (target\_term )}
\}
\]
last_visited_word = next_word;
result := result + LEARNING_PATH(next_word); 
}

next_word = MAX SEQUENCED NEIGHBOR(target_term)
last_visited_word = next_word
result := result + LEARNING_PATH(next_word);
}
else
return target_term;
}

This algorithm for learning path starts from the target_term as given input. We first check if any term is connected to the target_term via association, then we apply the function recursively on that connected term. If there is no other node ‘associated’ with a term, we follow the sequence that has the highest weight. ASSOCIATED_NEIGHBOR returns the set of terms that are linked with the input term via association. MAX_SEQUENCED_NEIGHBOR returns the term that has maximum weight among all the terms linked with the input term via sequence. For each recursive round, we remember the last visited node as last_visited_word. The function terminates when we meet the ROOT node or any node from the known domain. Next, we explain the mechanism with a case study.

7.2.1.5 Case study

The case study is based on the keywords used in a web based course of “Object Oriented Technology using JAVA”. In this example, hyperlinked keywords within definitions are shown as italic.

Polymorphism: Polymorphism is the ability to exist in more than one form, i.e. two functions can have same name with different behavior. There are two types of polymorphism - Static polymorphism and Dynamic polymorphism.

Static polymorphism: In static polymorphism, the binding between object and functions takes place at compile-time. Static polymorphism is implemented by Method Overloading.
Dynamic polymorphism: In dynamic polymorphism, the binding between object and functions takes place at run-time. Dynamic polymorphism is implemented by Method Overriding. This feature is essential for Abstract class and Interfaces.

Overloading: Method overloading is a feature that allows the creation of several methods with the same name that differs from each other in terms of the type of the input parameters.

Overriding: Method overriding is a feature that allows a subclass to provide its own definition of methods, which also have the same signature as the method in its super class.

Abstract class: An abstract class is a class that is declared with the keyword ‘abstract’. Abstract classes cannot be instantiated, but they can be inherited in subclasses that may or may not include abstract methods.

Interface: An interface is an abstract data type that is declared using the ‘interface’ keyword. It contains a group of related method signature. Classes must override all the methods of an interface in order to implement it.

Let us first extract the keywords used in the definitions:

Polymorphism: function, Static, binding, Method Overloading, Operator Overloading, runtime, binding, runtime, virtual function, abstract class Overloading: Method, parameters
Overriding: definition of method, signature, super class
Abstract class: instantiated, subclasses
Interface: abstract, method signature, Classes, override, methods

Then, transactions are created as followed:

$t_1$: function $\rightarrow$ static $\rightarrow$ binding $\rightarrow$ Method Overloading $\rightarrow$
operator overloading $\rightarrow$ runtime $\rightarrow$ virtual function $\rightarrow$ abstract class $\rightarrow$ polymorphism
$t_2$: Method $\rightarrow$ Parameter $\rightarrow$ Overloading
$t_3$: Method definition $\rightarrow$ signature $\rightarrow$ super class $\rightarrow$ Overriding
$t_4$: Instantiation $\rightarrow$ subclass $\rightarrow$ Abstract class
t5=Abstract→method
signature→classes→override→methods→interface

Next, we construct the graph for FP from the above transactions [Figure 7.3], where nodes represent the keywords and the edges indicate their dependency in a definition. Initially all the edges are weighted with 1, we call them a sequence. Next, this graph is updated by several iterations with large number of records available from different contact-mode classes of web-based definitions on this particular topic. Available open source tools for audio to text transcription like Sphinx (Lee, 1989), Julius (Lee & Kawahara, 2009) could be used for constructing transactions from classroom records. Similarly open source NLP software like CoreNLP (Manning, 2014) and OpenNLP (2011) can be used for term extraction from the web corpus. For each new transaction, if the sequence already exists in the graph then we increase the frequency of that particular edge by 1 and calculate the confidence factor, so that if it is increased above the threshold value ($\sigma$), we mark the link as association (represented by thicker arrow). Figure 7.3 represents a graph at any such stage of iteration where some of the edges have turned to association and rest has remained sequence. The associations are chosen arbitrarily to demonstrate how the proposed methodology could be applied on the graph to reveal the largest FP.

Let us assume;

*Overriding* be the target term
The term ‘*method*’ belongs to the known domain
Then the proposed methodology applied on the graph and we get the learning path as:  
Method → Method Definition → Method Signature → class → Super class → Overriding
The interpretation of the search result for ‘overriding’ reveals that:
If a learner wants to learn the definition of “overriding” with the initial knowledge of the term “method”, the proposed methodology would suggest her to follow the learning path Method ➔ Method Definition ➔ Method Signature ➔ class ➔ Super class ➔ Overriding

### 7.2.2 Generic Learning Path with help of a proposed data structure

This approach is different from the above-discussed one in the structure of the nodes. Instead of considering only the term as a node, here we propose a more complex data structure to support different aspects of a definition. The proposed data structure is based on the existing classifications on ‘types of definition’.

From our study on different models (Borg, 2008), (Aguilar, 2009), (Sierra, 2008)
of definition, we have identified a composite pattern of seven components as integral parts of a definition, as follows:

Classification: A definition can be a generalization or specialization of another concept. This are often described by “is a” or “type of” phrases.

Comparison: A definition can be described with a reference to another concept. This often use “with similar to” or “in contrast to” phrases.

Anatomic: A definition can describe a constitutional structure of which it is made up. “Constitute of” or “collection of” phrases are used here.

Operational: A definition may describe the behavioral aspect, i.e., it may describe, “What does it do?”

Qualitative: A definition can use certain attributes like physical characteristics or general traits with help of adjectives and adverbs to describe the term.

Example: A the definition can be explained using examples. ‘examples’, ‘like’, ‘such as’, etc. are the phrases used in this regard.

Usage: the definition can contain information about the areas where the term is used. It generally refers to the domain of its origin or application.

### 7.2.2.1 Data structure for the nodes

Considering the above segments of a definition, we propose a data structure for the nodes of a ‘definition tree’. A node has seven components, each of which corresponds to one of the above-discussed aspects. We decompose a definition to identify the keywords and then map them to the respective segments. For each of the keywords inserted in a segment, we introduce a child node with definitions of that keyword. This way a tree is formed for each term definition. Self-referencing keywords are purposefully discarded to rule out the possibility of any cycle.

Data structure of the node of the tree:

<table>
<thead>
<tr>
<th>N_l</th>
<th>A_l</th>
<th>D_l</th>
<th>T</th>
</tr>
</thead>
</table>
Struct node {
N_i as number /* number of children of the i^{th} data field. */
A[ ] as node* /* array of size N_i and holds the addresses of the children of the
node */
D[ ] as String[7] /* is an array for storing segment of the definitions as follows: */
/*
D [1] for Classification part
D[2] for Comparison part
D[3] for Anatomic part
D[4] for Operational part
D[5] for Qualitative part
D[6] for Example part
D[7] for Usage part
*/
T_i as Flag // indicates whether the node is already traversed in an iteration
}

<table>
<thead>
<tr>
<th>N1</th>
<th>A1</th>
<th>N3</th>
<th>A3</th>
<th>N5</th>
<th>A5</th>
<th>N7</th>
<th>A7</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification</td>
<td>Comparison</td>
<td>Anatomic</td>
<td>Operational</td>
<td>Qualitative</td>
<td>Example</td>
<td>Usage</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.4: Data structure for nodes of definition tree

Let us take an example [Definition taken from http://www.dictionary.com]:

*Electron*: an elementary particle, which is a fundamental constituent of matter, having a negative charge of $1.602 \times 10^{-19}$ coulombs, a mass of $9.108 \times 10^{-31}$ kilograms, and spin of 1/2, and exists as an independent particle outside the nucleus of an atom.
7.2.2.2 Tree construction

A definition tree is constructed for a definition by iteratively expanding the semantic segments at each level with introducing new sub-tree for each of the used keywords. The iteration continues until we reach leaf nodes where no other keywords are used in the definition.

A definition tree has the following properties

1. The target term that the learner wants to learn is the root.
2. A leaf node does not use any key word to define the term.
3. Address of the child nodes (terms used in defining a parent term) are stored in the immediate left of that data segment i.e., addresses for the children of $d_i$ are stored in the $A_i$ of that node.

The proposed methodology starts from the root of the tree with the term definition for which we want to construct the learning path. The definition of the term is partitioned into seven components according to the classification as described in section 7.2.2.1. Each segment $d_i$ contains a set of keywords, for each of such key words there is a new definition available, which is again described by a seven-segmented node at the next level. Link to these child nodes is stored in the address section ($A_i$) of $D_i$. This way the tree is evolved with iterations. If a key word,
which is already present in the tree, comes again in a new definition then we simply discard it to avoid cycles.

Next, in order to support the learner with a learning path, we propose a priority based post order traversal technique on this tree structure.

### 7.2.2.3 Setting priority

Although all the seven components of a definition reside in a single node, it is obvious that not all these segments have same impact on revealing the understanding of the term; rather it depends on the domain of the term. For example, for definition of a conceptual term the impact of example is more important than the usage. On contrary, for definition of an object/entity the anatomy part is more important than the example. Similarly, for definitions of the methods, operational part is more important than any others are. Therefore, we search for different available definitions of a term from multiple sources, usually web. Then we prioritize the segments based on the usage history. We store this information in a list called Priority list. Unlike the previous method (section 7.2.1) that uses the actual frequency count of the links, here we settle with three predefined levels for priority, namely, High=1, Medium=2, Low=3. However, priorities can be changed in ad-hoc manner for the improvement of the quality of a learning path to meet learner’s criteria. Next, we discuss an algorithm for constructing a learning path from this definition tree.

### 7.2.2.4 The learning path algorithm

```c
LEARNING_PATH (Node ROOT)
{
    while (True)
    {
        Level = 0;
        It_No = 0;
        PUSH (Node NULL);
        PTR = ROOT;
        Set P = 3;
        PUSH (Node PTR);
```

PUSH_SUCCE (Node PTR);
while (PTR ≠ NULL)  //  [Until NULL is Popped from STACK]
{
    PUSH_LM_PATH (Node PTR);  //  [Push Left most path on to the STACK]
    PTR = POP (STACK);
    while (PTR.T == 1)
    {
        Append the Node PTR at the L_PATH(It_No):
        if (P == 3) {
            Add (T_List, PTR);
            Level = Level - 1;
        }
    }
    if (PTR.T == 0){
        Set PTR.T = 1;
        Level = Level + 1;
    }
}
//L_PATH(It_No) is Learning path at present iteration
if (L_PATH(It_No) == L_PATH(It_No - 1) ) {
    // no changes made in last iteration
    break;
    else
    It_No = It_No + 1;
}
}

PUSH_SUCCE (Node PTR)
{
    Flag = 0;
    for  i = Data_F_No  to 1 {
        if (PTR.Ni ≠ 0 & Priority.Di ≤ P) {
Flag = 1;

for j = N_i to 1 {
    Member = MEMBER_T_LIST(Node PTR.A_j);
    if (Member == NULL) {
        Known = KNOWN_TERM(Node PTR.A_j);
        Com = COM_USAGE(ROOT, PTR.A_j);
        if (Known == NULL & Com == True) {
            PUSH(Node PTR.A_j);
        }
    }
}

PTR = POP(STACK);
PTR.T = 1;
PUSH(Node PTR);
Level = Level + 1;
if (Flag == 0) {
    PTR = NULL;
} else {
    Return PTR;
}

SET_PRIORITY (Level, It_No) {
    if (It_No ≥ Level + 2)
        P = 3;
    else if (It_No ≥ Level + 1)
        P = 2;
    else
        P = 1;
    Return P;
}
PUSH_LM_PATH (Node PTR)
{
   while (PTR ≠ NULL) {
      P = SET_PRIORITY (Level, It_No);
      PUSH_SUCCE (Node PTR);
   }
}

COM_USAGE (Node ROOT, Node N)
{
   List1[] = ROOT.Usage;
   List2[] = N.Usage;
   Com = \frac{\text{list1} \cap \text{list2}}{\text{list1} \cup \text{list2}} ;
   if (Com ≥ σ)
      Return TRUE;
   else
      Return FALSE;
}

The above algorithm finds a learning path from the definition tree built on the data structure already defined in section 7.2.2.1. We store the priority of each data field (D_i) in the priority table. Procedure LEARNING_PATH generates the learning path for the user. It uses the following sub-procedures.

PUSH_SUCCE: It first checks whether all the child nodes of a term are already in the list MEMBER_T_LIST. If not then it pushes all the child nodes into a stack.

SET_PRIORITY: It sets the priority of the level, which depends on the iteration no (It_No). Root node priority is always set at 3, so that it can unfold its children at deeper levels with increasing number of iteration.

PUSE_LM_PATH: It pushes the left most path of a pointer-PTR. It continuously pushes the left path of PTR until NULL is encountered.

COM_USAGE: It checks the common usage of an input term. It returns true only if the usage is more than a threshold value (σ).
7.2.2.5 Case study

We illustrate our proposed methodology with a case study on the definitions from the course of “Elementary Cell Biology”. Keywords used in a definition are marked italic.

DNA: Deoxyribo Nucleic Acid or DNA is a double-stranded nucleic acid that contains the genetic information for cell growth, division, and function.

Nucleic Acid: Nucleic acids direct the course of protein synthesis, thereby regulating all cell activities. The two main types, DNA and RNA, are composed of similar materials but differ in structure and function. Both are long chains of repeating nucleotides.

Nucleotides: Nucleotides are molecules that, when joined together, make up the structural units of RNA and DNA. A nucleotide consists of a sugar molecule attached to a phosphate group and a nitrogen-containing base. The bases are adenine, cytosine, guanine, and thymine or uracil (for RNA).

RNA: Ribonucleic acid or RNA a nucleic acid which is generally single stranded and plays a role in transferring information from DNA to protein-forming system of the cell.

Protein: Proteins are natural polymer molecules consisting of amino acid units. The number of amino acids in proteins may range from two to several thousand.

Amino acid: A molecule consisting of the basic amino group (NH2), the acidic carboxylic group (COOH), a hydrogen atom (H), and a side-chain (R) that varies between different amino acids attached to the carbon atom.

Cell: The structural, functional and biological unit of all organisms. It is the smallest unit of life that is classified as a living thing.

Eukaryotic cell: A cell that contains membrane-bound compartments in which specific metabolic activities take place. Most important among these compartments is the nucleus.
Nucleus: Nucleus is a membrane-enclosed organelle found in eukaryotic cells. It contains most of the cell's genetic material, organized as multiple long linear DNA molecules in complex with a large variety of proteins.

Organelle: An organelle is a specialized subunit within a cell that has a specific function.

Genetic material: The genetic material of a cell or an organism refers to those materials found in the nucleus, mitochondria and cytoplasm, which play a fundamental role in determining the structure and nature of cell substances.

Mitochondria: Mitochondria are the powerhouses of the human cell; mitochondrion is a membrane-enclosed organelle found in most eukaryotic cells. They generate most of the cell’s supply of adenosine triphosphate (ATP), used as a source of chemical energy.

Cytoplasm: The cytoplasm is a thick liquid residing between the cell membrane holding all organelles, except for the nucleus. All the contents of the cells eukaryotic organisms (which lack a cell nucleus) are contained within the cytoplasm.

Molecule: The smallest unit of an element or compound, made up of two or more atoms held together by strong chemical bond.

Atom: The fundamental building block of an element.

Metabolic: It is the set of chemical reactions that take place in living organisms to maintain life.

ATP: An organic compound composed of adenosine and three-phosphate group. ATP transports chemical energy within cells for metabolism.

Now, we construct a definition tree structure [Figure 7.6] for a particular target term, say ‘Mitochondria’.
Next, we arbitrarily assign priority of the segments for this case study.

Let

\[ D_1, D_3, D_4 = 1 \]
\[ D_2, D_5 = 2 \]
\[ D_6 = 3 \]

Using the proposed methodology, we get learning paths for the term *mitochondria* in different iterations as:

**iteration 1:**

Cell → Organelle → Cytoplasm → Genetic Material →
Nucleotide → Nucleic Acid → DNA → Molecule →
Amino Acid → Protein → Nucleus → Eukaryotic → ATP → Mitochondria

**iteration 2:**

Cell → Organelle → Cytoplasm → Genetic Material →
Nucleotide → Nucleic Acid → DNA → Molecule →
Amino Acid → Protein → Nucleus → Metabolic → Eukaryotic → ATP → Mitochondria

iteration 3: Cell → Organelle → Cytoplasm → Genetic Material → RNA → Nucleotide → Nucleic Acid → DNA → Molecule → Atom → Amino Acid → Protein → Nucleus → Metabolic → Eukaryotic → ATP → Mitochondria

7.3 PERSONALIZED LEARNING

We endeavor two different approaches to incorporate personalized learning environment within LMS. In the first approach, we produce a personalized learning path for the learner that suits her personal criteria. We have used the Ant Colony Optimization technique for the construction of the learning path, which is described in next section (7.3.1). In the other approach, we provide scaffolding for helping the tutor with personalized information about the learner in a synchronous communication medium. The scaffolding is constructed using the Online Analytical Processing techniques, which is discussed in the section 7.3.2.

7.3.1 Construction of learning path using Ant Colony Optimization (ACO)

Here, we extend the generic solution of constructing learning path as discussed in the section 7.2.1 to fit to the personalized requirement of a learner. We use an ACO based approach to find out a personalized learning path for the learner, which would help her to understand an unknown target term more easily. The proposed methodology has three important parts. First, we propose a usage graph for identifying the frequent patterns. This graph is built from repositories of learning contents describing definitions of terms. Secondly, we develop the transactions from the web-usage data of definitions. Some of the links between the terms are identified as ‘association’ based on the frequency of their co-existence within the transactions. Finally, we apply the ACO technique to find out a learning path that suits an individual learner for a target term. Since the first two parts of this methodology, i.e., graph construction and assigning frequency to the edges, are same as the methods already discussed in section 7.2.1.2, here we focus on how ACO could be adapted in finding a personalized learning path.
7.3.1.1 Adapting ACO for a learning path

Here, we propose the ACO-based inductive technique for construction of the learning path. In ACO, an ant tries to find out the shortest path between the source and the destination node of a graph by following the pheromone value deposited on the edges. The more pheromone on the path, the better the probability is to select that path. In our approach, we consider the ant would start from the target node, and move along the edges of the graph towards the root, until it finds a node from known domain or it reaches the root. It maintains a data list and checks it in each step to understand which nodes are used to define the target term and then chooses an edge from the available links that leads to an unvisited node. The solution path is constructed once the ant reaches a known term or the root. At each construction step, the ant probabilistically chooses an edge among those that lead to yet unvisited vertices. The probabilistic rule is driven by the pheromone value and heuristic information; the higher the pheromone and the heuristic value associated to an edge, more is the probability of an ant to choose that particular edge. Once all the ants complete their tour, the pheromone value on the edges is updated. Each edge then receives an amount of additional pheromone proportional to the quality of the solution to which it contributes.

Each ant starts from a query node (unknown term) $q_k$ and iteratively adds nodes until any unvisited node exists in its partial tour. One iteration is completed once the path from $q_k$ to the root node or any node marked as known is established. Each term can be used only once and then marked as visited for the rest part of that iteration. This constraint is enforced by the rule that an ant must choose the next node only among the unvisited nodes. Each ant $a_k$ has a memory $M_k$ that contains the list of already visited nodes, in the sequence they are visited. This memory is used to define the feasible neighborhood $N_{ik}$ for node $i$, that comprises all nodes that are unvisited. This feasible neighborhood concept is implemented by associating a data list with each term. An optimal path should contain maximum number of associations in it. If the feasible neighborhood $N_{ik}$ does not contain any association, then it chooses the edge, which has the maximum frequency value and contains the target term in the data list.

Two parameters $\tau_{ij}$ for pheromone trail value and $\eta_{ij}$ for attractiveness are used, to define the probability of making a move from the $i^{th}$ node to the $j^{th}$ node.
The pheromone trail value $\tau_{ij}$ indicates goodness of the move from node $s_i$ to node $s_j$ ($s_i \rightarrow s_j$). In our work, the pheromone represents the occurrence of association of a term in describing other terms. The trail value is therefore a posterior indication of the desirability of that move. Trail value of a particular move increases when the number of edges in the association $a_{ij}$, is increased in a tour $T_k$ where $k = 1, 2, \ldots, m$, is the identification number of ant, and $m$ is total number of ant that make the move $s_i \rightarrow s_j$. The attractiveness $\eta_{ij}$ is the heuristic information of the move $s_i \rightarrow s_j$. Trail value $\eta_{ij}$ is proportional to frequency value $c_{ij}$ of the edge $e_{ij}$.

Formally, the probability with which ant $a_k$, currently at node $s_i$, chooses to go to node $s_j$ is

$$
p_{ij}^k = \begin{cases} 
\frac{[\tau_{ij}]^\alpha[\eta_{ij}]^\beta}{\sum_{l \in N_i}^k [\tau_{il}]^\alpha[\eta_{il}]^\beta} if s_j \in N_i^k \\
0 & Otherwise
\end{cases}
$$

Here $\alpha$ and $\beta$ are two parameters which determine the relative influence of the pheromone trail and attractiveness. In our methodology, we set both of them to 1 for the sake of simplicity. After each iteration $t$, i.e., when all ants complete a path, trail values are updated by:

$$
\tau_{ij}(t) = \rho \tau_{ij}(t - 1) + \Delta \tau_{ij}
$$

Where $0 < \rho \leq 1$ is the pheromone evaporation rate, which is not relevant in our approach since the edge frequency once achieved would not be reduced. $\Delta \tau_{ij}$ represents the sum of contributions of all ants that move $s_i \rightarrow s_j$ to construct their path. The amount of pheromone deposited on an edge by all the ants in their solution path is

$$
\Delta \tau_{ij} = \sum_{k=1}^m \Delta \tau_{ij}^k
$$

Where $m$ is number of ants and $\Delta \tau_{ij}$ is amount of trail deposited on $e_{ij}$ by ant $a_k$, which can be computed as

$$
\Delta \tau_{ij}^k = \begin{cases} 
Q \cdot C^{k} if s_{ij} \in T^k \\
0 & otherwise
\end{cases}
$$
Where $C^k$ is the number of association present in $T^k$ and $Q$ is a constant, which is also set to 1.

A Learning path is constructed by applying the following procedures:

- At the beginning of each iteration, each ant starts from the query node $q_k$.
- A move is selected with help of the pheromone value and heuristic value and the learning path is constructed by iteratively adding new nodes those have the target term in their data list.
- The iteration terminates when the path reaches to any known word or the root node.
- For a personalized solution, this path can be further adopted to the individual requirements of the learner. If a term $q_u$ is in the learning path, which is not well understandable to the learner and we can extend the path by applying the same algorithm on the target term $q_u$ in place of $q_k$. This method can be repeatedly applied until learner’s criterion is satisfied.

7.3.1.2 The learning path algorithm

Initialize: $\eta_{ij} = c_{ij}$ and $\tau_{ij} = \tau_{ij}(0) \forall e_{ij} \in E$.

$Target\textunderscore Term = Query\textunderscore Term$;

$LEARNING\_PATH (Node Target\_Term)$

{ 
  for each ant $k$ {
    while not (ROOT is Reached) {

      Choose Next\_Term using Probability Distribution $P_{ij}^k$;
      
      $Target\_Term = Next\_Term$;
      
      $T^k := T^k + Next\_Term$;
      
      update $M^k$ & $N_i^k$;
      
      if (Next\_Term == Association) {
        $C^k = C^k + 1$;
      }

    }
  }
}
CHAPTER VII- AUGENTATION OF LMS WITH SOME ADDITIONAL FEATURES

\[
T := \min(T^k) \quad \text{Where } k = 1,2,\ldots,m
\]

\[
\text{UPDATE\_TRAIL ();}
\]

\[
\text{If not (END TEST)}
\]

\[
\text{LEARNING\_PATH (Target\_Term);}\]

\[
\text{else}
\]

\[
\text{return } T;\]

\[
\}
\]

\[
\text{UPDATE\_TRAIL ()}
\]

\[
\{
\]

\[
\text{for each edge } e_{ij}
\]

\[
\text{for each ant } k
\]

\[
\{
\]

\[
\text{if}(e_{ij} \in T^k)\
\]

\[
\Delta \tau_{ij}^k = Q \cdot C ;
\]

\[
\}
\]

\[
\text{else } \{
\]

\[
\Delta \tau_{ij}^k = 0 ;
\]

\[
\Delta \tau_{ij} = \Delta \tau_{ij} + \Delta \tau_{ij}^k ;
\]

\[
\}
\]

\[
// \text{Update the TRAIL MATRIX}
\]

\[
\}
\]

\[
\}
\]
7.3.1.3 Case study

Using the same case study described in section 2.2.5, we now try to construct the learning paths for two terms Mitochondria and Eukaryotic using the proposed methodology. We consider \{cell, nucleolus, atom, protein, DNA\} as a subset of the set of definitions known to the learner.

First, we construct the usage graph for the case study [Figure 7.] based on methodology described in se. 2.1.2. Additionally, we use a data list to support the methodology. Table 7.1 represents an instance of data list at any time.

![Diagram of usage graph for frequent pattern](image_url)

Figure 7.7: Usage graph for frequent pattern [case study of 7.2.2.5]
Table 7.1: Data list for terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Data List</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATP</td>
<td>Mitochondria</td>
</tr>
<tr>
<td>Amino Acid</td>
<td>Ribosome, Protein</td>
</tr>
<tr>
<td>cell</td>
<td>Mitochondria, Eukaryotic, Organelle, DNA, Nucleic acid, Cytoplasm, Ribosome</td>
</tr>
<tr>
<td>Cytoplasm</td>
<td>Mitochondria</td>
</tr>
<tr>
<td>DNA</td>
<td>Nucleic acid, Eukaryotic, Nucleus</td>
</tr>
<tr>
<td>Organelle</td>
<td>Mitochondria, Eukaryotic, DNA, Cytoplasm, Nucleus, Ribosome</td>
</tr>
<tr>
<td>Eukaryotic</td>
<td>Mitochondria, Cytoplasm, Nucleus</td>
</tr>
<tr>
<td>Metabolism</td>
<td>Eukaryotic</td>
</tr>
<tr>
<td>Mitochondria</td>
<td>Cytoplasm</td>
</tr>
<tr>
<td>Molecule</td>
<td>Nucleic acid</td>
</tr>
<tr>
<td>Nucleus</td>
<td>Eukaryotic, DNA, Cytoplasm</td>
</tr>
<tr>
<td>Nucleic acid</td>
<td>DNA</td>
</tr>
<tr>
<td>Prokaryotic</td>
<td>Cytoplasm</td>
</tr>
<tr>
<td>Ribosome</td>
<td>Cytoplasm</td>
</tr>
<tr>
<td>Nucleotide</td>
<td>Nucleic acid, Ribosome</td>
</tr>
<tr>
<td>Protein</td>
<td>Nucleic acid, Nucleus, Ribosome</td>
</tr>
<tr>
<td>RNA</td>
<td>Nucleic acid, Ribosome</td>
</tr>
<tr>
<td>Genetic Material</td>
<td>Nucleus</td>
</tr>
</tbody>
</table>

Now, based on the learner’s level of knowledge the proposed methodology can construct a personalized learning path for the learner. Considering \{cell, nucleolus, atom, protein, DNA\} as a subset of the known words of the learner, our methodology proposes the following results.

I. For **Mitochondria**: the recommendation is \{ATP \rightarrow Eukaryotic \rightarrow Organelle \rightarrow Mitochondria \}
II. For Eukaryotic: the recommendation is \{Metabolism \rightarrow \text{Organelle} \rightarrow \text{Eukaryotic}\}

The above-specified learning paths are personalized solutions for a learner with the above-mentioned set of known words. However, a learner can further extend the above-mentioned path for further clarification. For example, in the second case (Eukaryotic), if the learner has confusion in the term Metabolism, the learning path can be extended to $A TP \rightarrow \text{Metabolism} \rightarrow \text{Organelle} \rightarrow \text{Eukaryotic}$ based on the same set of known words.

Next, we discuss on the other type of personalization of the learning environment, which is based on scaffolding.

7.3.2 Supporting synchronized communication with personalized scaffolding

Scaffolding learning theory suggests that instructors should construct different learning scaffolding stands according to level of ability and learning progress of different learners (Yang, 2000). Our work is focused on construction of scaffolding to provide useful learner’s information to the instructor that could be useful for personalized problem solving. In the proposed approach, we first investigate the log data of the LMS users to understand learners’ behavior. Then we apply OLAP techniques to prepare analytical reports on learner’s individual traits, which are then used to construct the runtime scaffolding. An architecture is also proposed that adopts and use this scaffolding to aid learner’s personalized problem solving within the LMS.

7.3.2.1 Tracking learner’s behavior from log

In order to support teachers with learner’s personal information, different attributes of learner’s records must be accessed. This includes learner profile, past behavior, preferences, interactions with other users, and characteristics of the learning environment that they exploit for learning activities. Below we discuss on some of the attributes that work as a source of learner’s information.
CHAPTER VII- AUGMENTATION OF LMS WITH SOME ADDITIONAL FEATURES

➢ **User log:**
Most contemporary LMS supports collection, organization and reporting of data on different learner activities. Data from user log about learning activity include learning session, assignment record, assessment performance, average time spent on a particular page, total time spent on a particular course, navigation history, download and uploads, and interactions with teachers and peers.

➢ **Learner’s preference:**
Learner’s preference can be set explicitly or implicitly. When explicitly stated, learner’s preference is stored in a separate file, which can be accessed. Otherwise, we can implicitly find out learner’s preferences from her different activities. Records of learner’s searching history within the LMS and in the web can reveal her searching preferences. Searching a particular key word repeatedly across different sites indicates urgency and importance of the key word for the learner. On the other hand, learner’s navigation style like whether the learner skips some modules or wants detailing of a particular topic, indicates the competence level of the learner on that topic. Similarly, bookmarking or favorite list maintained by the learner indicates learner’s desire to use and apply the knowledge.

➢ **Learner’s interaction record:**
Three primary components of LMS software are learners, teachers/instructors, and contents. A runtime learning environment perceives all sorts of interactions among these three components, which are very important to the construction of the scaffolding. Table 7.2 represents the possible ways of interaction between the components. However, since the scaffolding is only for providing useful information about the learner, below we discuss only the interactions of the learners.
Table 7.2: Possible interaction modes

<table>
<thead>
<tr>
<th>Learner</th>
<th>Facilitator</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learner</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cooperative</td>
<td>Asking question,</td>
<td>Self pace learning</td>
</tr>
<tr>
<td>And</td>
<td>Doubt clearance</td>
<td></td>
</tr>
<tr>
<td>collaborative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>learning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Facilitator</td>
<td>Sharing Knowledge &amp;</td>
<td>Making</td>
</tr>
<tr>
<td>Interacting</td>
<td>experience</td>
<td></td>
</tr>
<tr>
<td>And</td>
<td></td>
<td></td>
</tr>
<tr>
<td>facilitating</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content</td>
<td>Interacting interfacing</td>
<td>Interacting interfacing</td>
</tr>
<tr>
<td>Interacting</td>
<td>Interacting interfacing</td>
<td>Interacting interfacing</td>
</tr>
<tr>
<td>interfacing</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Learner-to-Content interaction:**
  Contents of e-learning are made more and more interactive for the user with the flourish of different multimedia elements like text, audio, video, animations, instructions, simulations, macros, and presentation tools etc. Therefore, details of the interaction of the learner with different content types must be recorded. For example, learner’s involvement in animation and simulation, type of files downloaded and uploaded, and the frequency of using macros etc. are useful to understand the confidence level of a learner.

- **Learner-to-Learner interaction:**
  Learners’ peer collaboration enables to link up different ideas and sharing of knowledge, which induces motivation and positive learning outcomes. Asynchronous collaborative approaches like mail, messaging, blogging and synchronous approaches like chatting or video conferencing are available in almost all contemporary LMSs [discussed in Table 1.2, section 1.5]. Such useful information if recorded properly, is a good resource for analyzing and understanding learner’s characteristics and interest of study.
o **Learner-to-Instructor interaction:**
In the LMS environment, a learner can interact with the instructor via chat, mail or blog. The nature of question (e.g. if it is subjective or objective), the frequency of asking question and the relevancy of the question to the subject are good indicators of learner’s knowledge level, interest, and confidence. This is particularly important for the instructor in order to facilitate the learner with personalized feedback.

➢ **Learning Path:**
The adaptive release of e-learning contents enables the learner to experience an individualized learning path within the LMS. It also allows the facilitator to create rules for delivering content to the learner based on specific tasks, which leads to the achievement of the course’s learning objectives. It can be implemented via prompts and clues that encourage learners to think about their learning process and to utilize appropriate learning strategies. Although, not too many contemporary LMS vendors provide this feature, information about how the learner uses such learning path is potentially a good source for understanding learner’s characteristics.

➢ **Learner’s assessment:**
Contemporary LMS software comes with integrated assessment tools that help instructors to a detailed evaluation of learners’ abilities (Razzaq, 2007). Assessment records of learning can be used as a reference for instructors to evaluate the learning accomplishments of students and diagnose their learning difficulties (Lee, 2008). Therefore, assessment records enable the construction of a rich learner profile that records learner’s progress and characteristics.

### 7.3.2 OLAP operations
OLAP operations can reveal useful information about learners’ behavior when applied on the properly arranged (multidimensional data model) learner’s record. This work uses star schema architecture and creates a fact table with five dimension tables to represent learner’s record as shown in Figure 7.8. Operations like Roll-up, drill-down, slice, dice, and pivot are performed on the database to extract useful information about the learner.
7.3.2.3 Using the scaffolding

Figure 7.9 depicts the overall framework of the scaffolding. The scaffolding provides a purely temporary stage, which is instantiated with the start of every live session of the instructor with either a learner or group of learners. This work uses five years (2007-12) log data from the LMS server (Moodle) installed at Bengal Institute of Technology for offering different subjects in undergraduate
courses. We have used GISMO (Mazza & Milani, 2004), an open source Business Intelligence (BI) tool, to analyze Moodle log files and then performed OLAP operations to present learners’ information in the form of reports and graph representations. Since our source of records is a single Moodle database system, we have managed to skip the complexity of the Extraction-Transformation-Load (ETL) process. The scaffolding is constructed at runtime using OLAP operations on the learner records, when the Learner-Instructor interactive session initiates. It holds the necessary information about the learner about her subject competence, base-knowledge, weakness, preference etc. In addition, it produces useful report on aggregate data that reveals the position of the learner in her peer group. It enables the instructor, in the synchronous question answer or doubt clearance session, to reframe her answer or illustration that suits the learner most. Snapshots of some of the reports that support the scaffolding are shown in Figure 7.10(a), 7.10(b), and 7.10(c). Figure 7.10(a) describes learner’s position in her peer group in terms of performance. A comparison of learner’s assessment with others is presented. Figure 7.10(b) shows comparison of time spent by learner at different units with the average time spent by all the learners at those units. Figure 7.10(c) represents the total number of activities done by the learner in the learning units. These include blogging, chatting, asking questions, sharing resources etc. on a particular learning unit. It is compared with the average number of activities done on that unit.

![Figure 7.10(a): Analyzing learner’s performance record](image-url)
7.4 ADAPTIVE LEARNING

A iLMS is the integration of ITS and web based LMS (Brusilovsky & Peylo, 2003). It is intended towards making web-based learning more flexible, autonomous and adaptive to the needs of each learner (discussed in section 2.2.5). Here, we introduce a prototype implementation of iLMS, named CompTutor, to bring in some adaptive features of LMS (Sengupta & Dasgupta, 2014).

Conventionally, a iLMS provides individualized tutoring to the learners with help of five modules, i) student model ii) domain model iii) tutor model, iv)
communication model and v) interface model. The student model, tutor model and domain model are inherited from ITS, whereas the other two, communication model and interface model are adaptation of web technology within LMS. Figure 7.11 represents the generic architecture of iLMS.

The first layer is composed of student model, domain model and tutor model. It is the core layer of iLMS and its features are inherited from the traditional ITS. The Second layer represents the web architecture of how the system is distributed and deployed across the web. The third layer is the web interface layer that represents how the users interact with the system. The learners who are the user of the system are at layer-4, the last layer. Next, we explain the components of the architecture with help of our prototype implementation CompTutor.

### 7.4.1 Module for domain model

Domain model, also known as expert model, refers to the topics and curriculum along with the facts and rules of that particular subject domain. It assumes that knowledge can be described in terms of facts and principles that can be learned in
an incremental fashion. This model tries to abstract what the learner should learn in the course and what goal should be achieved in that course. Considering, one has to be a subject expert to have such type of knowledge, the success of building a domain model depends on the teacher's expertise.

We consider a course in LMS as a sequence of LOs where each LO has the following properties:

   i) The topic should have unique identifier
   ii) The topic name should be defined
   iii) Keywords are identified and stored in a separate list
   iv) Every keywords are hyperlinked that point towards the LO that describes it
   v) Prerequisite are identified in terms of keywords
   vi) Examples are stored separately; each example should define two parts: one, what is the difficulty to solve the problem without the new knowledge and two, how the new knowledge is used to solve the problem.
   vii) Tables, diagrams, image are stored separately.

Next, we focus on some basic principles of domain modeling. Figure 7.12 shows how CompTutor provides an authoring interface to accomplish the tasks for domain model. This interface helps the author to deliver the contents, prerequisites, keywords and images in accordance with the requirement of the domain model. Once the submission of this page is done, the author is requested to identify the ‘keywords’ from the content. Then all the keywords and the prerequisites are asked to link with their corresponding LOs. In case there is no available LO for a keyword, the author is requested to provide a short note on that keyword which may come as a pop-up help for the learner. Finally, the author provides a set of multiple-choice questions for an assessment test. The author can also specify some domain rules in these question–answers, like selection of any particular wrong answer from the list may infer a common misconception on a particular concept or keyword.
7.4.2 Module for student model

Student modeling is the core of iLMS architecture. iLMS are different from conventional LMS by its ability to respond to the individual student's learning style and delivery of customized contents and instructions. According to Brusilovsky & Millan (2007), a student model/learner model mainly focuses on three tasks:

i) It collects data about the learner from all possible sources. This data can be obtained either explicit, by asking the learner to provide static information or to solve specific problems, or implicit, by tracking the learner’s navigation and other interactions with the system.

ii) It uses learner’s data to create a model-based representation of her knowledge and learning process. This often takes the form of "buggy" models that represent the student's knowledge in terms of deviations from
an expert's knowledge (Martens & Uhrmacher, 2004). The system then uses this model to predict what type of response the student could make in subsequent situations. Then it compares the prediction to the students' actual response, and uses that information to refine the learner model.

iii) The student model must account for the state of the learner’s knowledge. It helps the tutor model in selecting optimal pedagogical strategies for presenting subsequent domain information to the learner. One major challenge for student model is the fact that students do not always respond consistently, particularly when their knowledge is fragile and they are uncertain about the correct responses (Frey, 2000).

In CompTutor, we collect learner’s preference information at the start of each session. We define a session as the time between when the learner initiates an interaction with a topic (LO) and when the learner completes a topic and goes to the next topic or exits the system. In CompTutor, the student model stores information about learner’s preference for the entire session. The learner can select from the two available options - beginners level and advanced level. In beginners level, links for ‘help’ and ‘hint’ are provided and more hyperlinks are available than the advanced level. A learner can select from any one of the learning style between ‘example followed by concept’ and ‘concept followed by example’. A learner can also select some keywords from the prerequisite list for her recapitulation. Figure 7.13 shows the interfaces to accomplish these tasks.
7.4.3 Module for tutor model

The tutor model is also known as instructional model or pedagogy model. It is a representation of the methods used to provide customized information to the learner. This model regulates the instructional interactions of the system with the learners. It deals with pedagogical issues (Martens & Uhrmacher, 2004) i.e. how to teach, in what order, typical mistakes and remediation, typical questions a learner might ask, hints that one might offer to a learner who is stuck. It tries to model the way in which the learner’s problems are addressed by teacher in the face-to-face interactions. According to Contreras et al. (2007), a tutor model should have the following properties:

i) It should analyze the student model and interpret the level of knowledge that the learner possesses
ii) It should define the actions that the learner can perform to achieve the goals in a correct way
iii) It should adjust the pedagogy style in accordance with learner’s preference and need.
In *CompTutor*, we focus on three important tasks of tutor modeling namely sequencing, navigation and learning style. First, we attach a recapitulation part at the beginning of the course in addition to the standard sequence as defined by the author, before presenting it to the learner. The required recapitulation for individual learner can be identified from the learner model or learner herself can set the preference. We use a set of keywords identified as ‘hitch -words’ for that particular learner on that learning session for this purpose. The tutor model can also decide to re-launch the course to the learner after evaluation of the assignment; it can then alter the sequence by adding/ removing some LOs based on learner’s experience recorded with this course. Secondly, we control the navigation by adjusting the hyperlinks with in the context. The domain model provides all possible hyperlinks for the keywords and then the tutor model adjusts the links based on the knowledge level of the learner. For example, inactivate the hyperlinks of the key words that are already covered in previous lessons or the learner has shown confidence on them. Finally, in case of adapting learning style, we provide two simple alternatives, ‘concept followed by example’ and ‘example followed by concept’; whichever suitable is chosen by the learner.

Figure 7.14 shows how *CompTutor* adapt to personalized sequencing and navigation for the learner. In this particular case, we can see that the definition of “partial functional dependency” has come before the start of the topic “2nd normal form”. This is because the learner has opted for a recapitulation on this topic. The link of the next button is dynamically created which points either to start of the specified topic or to any other recapitulation topic as preferred by the learner. The recapitulation topic presents only a brief narration about the topic without revealing further links or examples. However, if the learner wants more, she can get the complete set of LOs on that topic by clicking on the ‘more details’ button.
7.4.4 Module for communication model

In order to introduce intelligent behavior within the system, communication model distributes the logic of student model and tutor model across different levels of the web architecture. For example, learning experiences of the students at different nodes within an institution are collected in a local server and then we apply some logic to understand the common pattern of mistakes on a particular course. Then information available from all such local servers at different sites are integrated at a central server and consequently analyzed to update the student model. In addition, the tutor model can enhance the course to adapt to the received feedback from the learners e.g., adding more links, changing picture, or adding diagram. This change will be reflected from the next time any local server launch the course from the central server. Thus, communication between the distributed components of CompTutor helps us to store and analyze learners’ information at different levels of the architecture. Figure 7.15 reflects a schematic diagram for the iLMS architecture.
7.4.5 Module for interface model

The user interface model controls the interaction between the users and the system (Figure 7.11). It translates the system's internal representation to an interface language that is understandable to the student (Larsen et al., 2010). This model is also responsible for other users’ (Teacher, Author, Admin) interaction with the learner and the system. The rich variety of hypermedia based web interfacing techniques is used in this model. Some of the instances of interface related to the domain model [Figure 7.12], student model [Figure 7.13] and tutor model [Figure 7.14] are depicted above.

7.5 CONCLUSION

This chapter discusses about some additional features of LMS in terms of personalized and adaptive learning. Some methodologies are proposed in the field of learning path, scaffolding for personal guidance and adaptive sequencing and navigation. Two different types of learning path are proposed. The first type of learning path is a general recommendation for learning a particular unknown term starting from a particular known term, for all the learners at a particular level.
Two different approaches are taken to the construction of generic learning path, one using FP tree and other using a proposed data structure. The second type of learning path is a personalized recommendation for individual learners, which is based on ant colony optimization technique that uses learners’ existing knowledge base, to construct a personalized learning path. For both the types, a learner is supported with a learning path that recommends a series of term definitions, which would reach her to the understanding of the desired unknown term.

In the context of personalized learning, besides learning path, a framework for runtime scaffolding is also proposed to offer useful information about the learner to the instructor in the live question answer session over a synchronous medium. This enables the instructor to solve individualized problems of the learner more effectively. Finally, in the adaptive section we propose some techniques related to adaptive presentation and adaptive navigation. We demonstrate all the proposed methodologies with a prototype implementation- *CompTutor*. 