Curriculum Tree : A Knowledge-based Architecture for Intelligent Tutoring Systems

Tak-Wai Chan

Institute of Computer Science and Electronic Engineering
National Central University
Chung-Li, TAIWAN 32054
R. O. C.

email: chan@ncu.dnet.ncu.edu.tw
fax: 886-3-4255830

Abstract. This paper describes a knowledge-based architecture, called curriculum tree, for building intelligent tutoring systems. Primarily based on the subject domain knowledge structure, the architecture naturally incorporates the global curriculum planning and monitors the local learning activities. The curriculum tree can also be viewed as a structure of various teaching knowledge at different stages of learning. By adopting rule inheritance, the architecture allows additional additivity and flexibility for developing an intelligent tutoring system incrementally as well as efficiency for running rules in each learning episode. Thus, curriculum tree is an architecture towards building large scale intelligent tutoring systems. In this paper, we shall also discuss how the curriculum tree architecture is used in building Integration-Kid, a Learning Companion System which is a particularly complex type of intelligent tutoring system, in the domain of learning indefinite integration.

1 Introduction

There are tremendous factors have to be considered in building an intelligent tutoring system (ITS). The goal of this work is to develop a model of knowledge-based architecture, called *curriculum tree*, which allows non-AI experts to construct their own ITS programs for a complete tutoring course. This paper discusses curriculum tree and describes how it is used in building a *learning companion system*, called *Integration-Kid* [1, 2, 3], in the domain of learning indefinite integration. Learning Companion System (LCS) is an intelligent tutoring system, but is alternative to one-on-one tutoring. In an LCS, apart from modeling the computer as a teacher, it models after an additional agent, called the learning companion.

At the early stage of the implementation of Integration-Kid, we soon discovered that its complexity goes far beyond the capability of a simple-mined knowledge-based system; nor can be specifically and accurately handled by sophisticated general purpose knowledge-based systems such as KEE. There have been efforts in building intelligent systems for instructional design [5, 7, 12, 14, 15]. Also, a number of researchers use artificial intelligence planning techniques for dynamic instructional planning [8, 9, 11]. Different from these works, our focus is directed to the architecture of knowledge-based programming environment. The intended purpose is to solve some software engineering problems arisen from building ITS such as to make the complexity of knowledge-based

ITS manageable so that effectiveness and efficiency of the development process can be maintained.

2 Domain Structure, Learning Goals, and Learning Activities

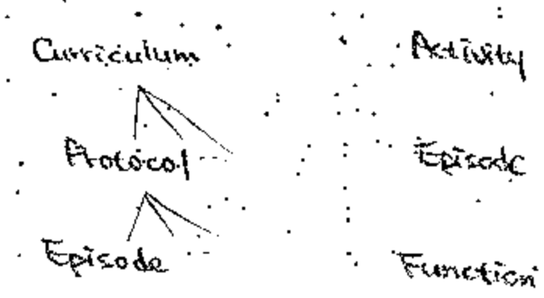
In education, the term, *curriculum*, is defined as the repertoire of the pre-designed learning activities or teaching goals based on the constraints of the domain knowledge. (The domain structure defines the learning goal hierarchy which, in turn, is the basis of the design of learning activities.) Thus, teaching or tutoring can be viewed as the execution of previously planned activities accompanied by monitoring the process. This view is in contrary to that of Wasson's [13] claim: the learning goal structure is not the same as the domain knowledge structure, and that the curriculum designed based on pedagogical principles takes more than the domain structure.

A teacher's plan can be characterized by its global curriculum planning with local decision making in monitoring its execution. Therefore, the system architecture for ITS should allow the designer (or teacher) to constrain the range of students' experiences with thoughtful curriculum planning, support local monitoring of the learning activities, and be easy to modify both at the global curriculum planning level as well as at the local monitoring level. Unfortunately, as Peachey and McCalla [11] noticed, a serious weakness of most ITS is their lack of a global curriculum plan. This is because ITS researchers have neglected instructional design and curriculum structure [5], and, in particular, they have largely ignored task analysis [6], a common method used by instructional designers to identify instructional objectives and the relevant pedagogical relations among them [10].

3 Curriculum Tree

The LCS learning activities can be described by three levels of abstraction. The global level is the *curriculum level*. The curriculum is the whole discourse of learning activities in a certain structure. The second level is the *protocol level*. A protocol organizes learning activities in a certain format. In Integration-Kid, examples of protocols are *working independently*, *one working and one watching*, etc. [2]. The last level is the *episode level*. An episode is a basic unit of learning activities, which usually has a beginning and an end. An episode can also be viewed as an instance of a protocol.

In short, a curriculum consists of a set of protocols, a protocol is composed by a set of episodes. In other words, protocols and curriculum are abstract descriptions of episodes. We can represent these three levels of abstraction by a tree structure, called *curriculum tree* (Figure 1). In terms of this curriculum tree structure, the curriculum is the whole tree or the root node of the tree, the protocols are the internal nodes above the episode nodes which are the leaf nodes. We take bottom up approach to describe the architecture. Thus, before we further discuss curriculum tree, we first discuss the design of episodes. Then come back to the curriculum tree and describe how the design of the curriculum tree organizes the episodes and the whole process of learning activities.



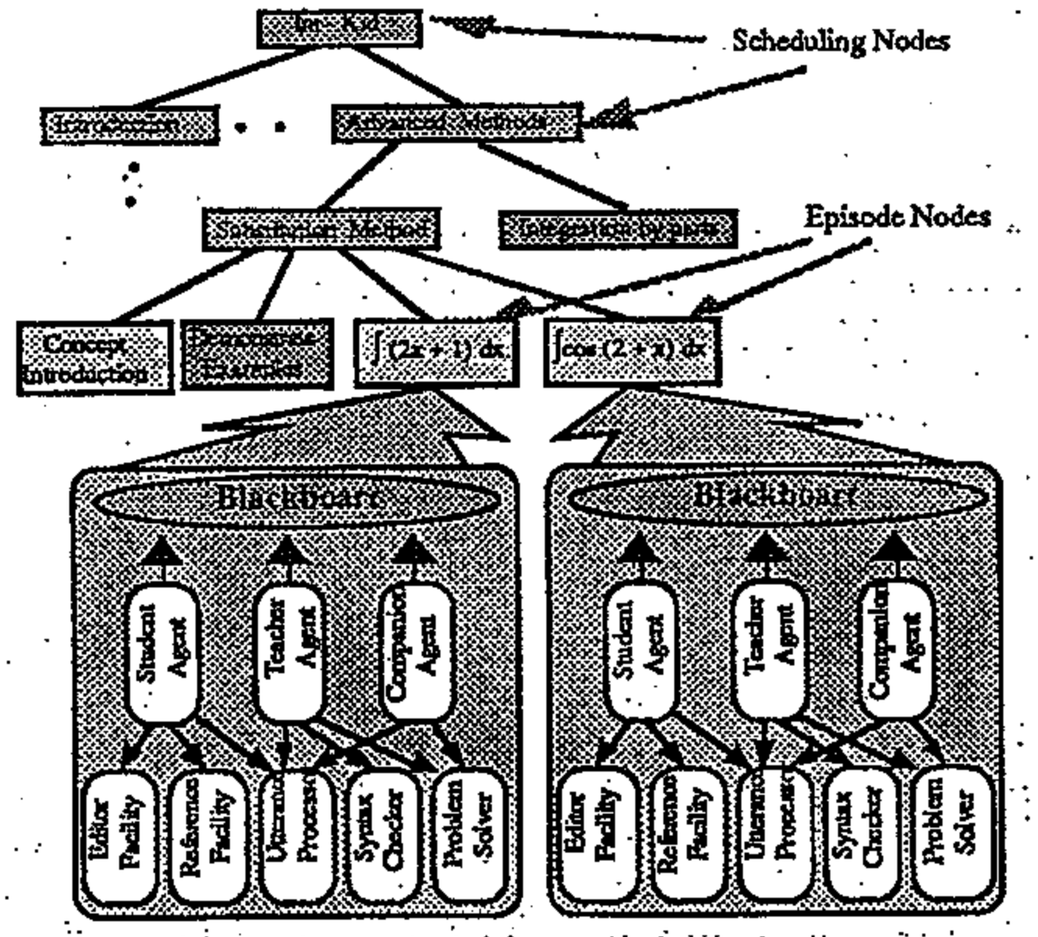


Fig. 1. Curriculum Tree where Each Episode Node is a Blackboard System

3.1 Coordinating Three Agents' Interaction via Blackboard

The system at the episode level (leaf nodes of figure 1) simulates the three agents' interaction. Within an episode, there are two major issues: the representation of each agent and the interaction among them. Thus, we represent the three agents separately in the system. Each agent is a set of *rules of behavior* modeling the behavior of the agent. The three agents communicate through a *blackboard* via a simple *agent scheduler* which controls the agents when look at the blackboard and execute. The student agent contains those rules that interpret the student's input and put it on the blackboard for the other two agents to react. Rules of behavior are represented as production rules, and so an agent is effectively a production system [4].

Behavior of the human student is driven by his own intelligence. But, for the teacher and the companion, their general communicative behavior in learning is supported by their domain knowledge. Their problem solving abilities, part of their domain knowledge, are modeled by the problem solver and are called by the right hand side (RHS) of the rules of behavior. Such problem solving ability can be viewed as part of an agent's behavior. For the student agent, apart from those rules that interpret the student's input, there are rules that allow the student to control the system at his own pace. Figure 1 illustrates some of the sub-programs that support the rules of behavior of different agents.

3.2 Rule Inheritance and Scheduling via Curriculum Tree

Here we relate the design architecture of an episode discussed above to the curriculum tree (Fig. 1). Basically, the curriculum tree serves two purposes in the overall design of Integration-Kid: it arranges and stores the rules of behavior into the structure of the domain knowledge, forming a curriculum tree; the curriculum tree also serves as a platform for scheduling episodes. Thus, the curriculum tree organizes the actual program of the learning activities according to the domain knowledge structure.

In the development of the program, rules of behavior have been written for the agents. When hundreds of rules of behavior are accumulated, writing an additional rule will become difficult. Besides feeling disoriented when working hundreds of rules together at the same time, there are problems of complexity and efficiency both in constructing and running the rules. Some of the rules' conditions in one protocol of activity (part of the curriculum) are different from the rules in other protocols but some are the same. Moreover, rules with the same conditions may have different RHS. To distinguish rules, one might need to index the rules according to different parts of the curriculum. Such indexing causes complexity on the left hand side (LHS) of the rules which would be difficult to understand. Also, running with hundreds of rules at a time will substantially slow down the speed to respond to the student. In fact, at a given point in the curriculum, the student's input to the system corresponds to a certain expectation by the system. That means, at any given point in the curriculum, there are only a limited number of appropriate rules.

There are two types of nodes in a curriculum tree. The leaf nodes are called *episode nodes* and the internal nodes are called *scheduling nodes*. Episode nodes represent episodes in the discourse of learning. Scheduling nodes are responsible for passing data and choosing a subsequent episode after running an episode. Now, each episode node in the curriculum tree is a blackboard system on its own right in which three separate agents (teacher, companion, and student) represent three different production systems and share a common blackboard locally scheduled by the simple agent scheduler. This means that the rules of behavior for each of the episodes would be different from each other.

We do not need to store all the relevant rules for each episode in the episode node. Taking advantage of the structure of the curriculum tree, the rule base of each agent in an episode consists of rules inherited from its ancestor nodes plus some resident rules which are particular to that episode. Thus, rules that are common to nearby episodes will be stored in their ancestor nodes. Rules that are more commonly used, for example, rules for calling editor functions, will be stored at higher level nodes of the tree. Rules such as for referencing tables of integration rules which are present all the time throughout the discourse of learning are stored at the top node. Therefore, rules of behavior in the nearby episodes will have more overlap than those that are distant. Also, rules of behavior in the lower level nodes override those rules with the same LHS in the higher level nodes. As a result, rules which simulate the protocol of activities are stored in the internal nodes above the leaf nodes of those corresponding problems. Rules that are specific for a particular problem reside in the corresponding leaf node, an episode node.

For each agent, its rules of behavior are distributed in a tree. So there would be three trees for storing the rules of behavior corresponding to the three agents. But the three trees are of the same structure. Thus, we can represent these rule distributions in the same tree. For scheduling, each scheduling node, apart from passing down inherited rules of behavior to different agents, is a production system itself. It has local data and a set of *scheduling rules* to pass data up and down, to traverse the tree, and finally to choose an episode node.

As can be seen, a curriculum tree is in fact a tree of knowledge-based systems working together. The global intelligent behavior is demonstrated by the ability to schedule episodes while the local intelligent behavior appears in the interactions among the three production systems via a local blackboard within an episode. In fact, the upper part of a curriculum tree represents the global and rather static plan of the whole discourse of activities. In the lower part of the tree, decisions are made in a dynamic way when more

information about the learners is available. This illustrates a simple view of the teacher's plan which is from global to local and from static to dynamic. Furthermore, apart from the rules of behavior, the scheduling information is distributed over the whole tree, initially represented in the blackboard of an episode node which is then further interpreted and abstracted up the tree in the process of choosing the next episode. The whole discourse of activities is thus a sequence of choosing and running episodes.

Integration-Kid is, a particular curriculum tree, written in Common Lisp and run on T.I. Explorer. There are 77 nodes in the curriculum tree of Integration-Kid. 56 of them are leaf nodes, that is, episode nodes.

4 Views of Design Approach

The curriculum tree is essentially a tree of knowledge-based systems working together based on the domain structure. It can be viewed as a two-level blackboard architecture. At the local level, the episode node is a blackboard system where the three agent production systems are the knowledge sources. All behavior will then be determined by the rules of behavior including those modeling a protocol of activities. The rules of behavior may in turn call some supporting functions. At the global level, every single episode node is a knowledge source. During scheduling for selecting episode, a scheduling node in an internal node of the curriculum tree is activated. This scheduling node, a production system by itself, is the scheduler. The scheduling strategy is defined by the scheduling rules of the node and the data on the blackboard. This view provides the conceptual separation between managing a tree of a knowledge-based system and running a particular knowledge-based system.

Another view (fig. 2) of our overall design of LCS for integration perhaps can be based on the language constructs that have been built on top of the LISP language. The curriculum tree is a higher level language for describing the LCS curriculum structure of learning activities. These learning activities can be organized in terms of the learning goal hierarchy which can be seen as a mapping on the domain knowledge conceptual dependency structure. The basic entity of the code for learning activities is a rule of behavior of an agent, and for scheduling the basic entity is a scheduling rule. These basic units of code are produced from another higher level language, the production system. All these rules are organized in the curriculum tree. The system runs locally by the inference engine of the blackboard system in an episode and globally by the inference engine of the scheduling nodes in the curriculum tree.

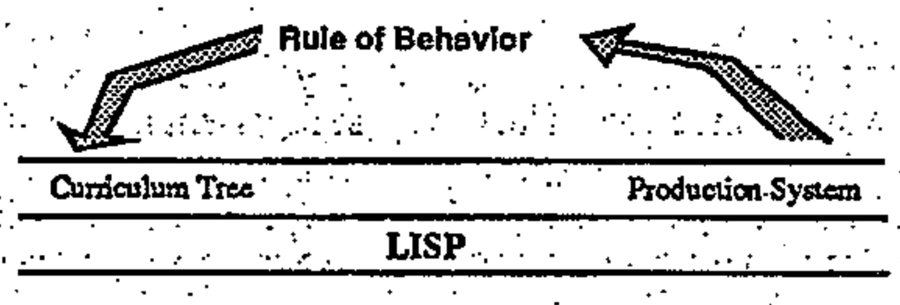


Fig. 2. Language Levels of LCS System Design

5 Development of ITS via Curriculum Tree

We outline the process of development (Fig. 3) of an ITS using curriculum tree which is essentially a summary of our experience building the Integration-Kid. Given a domain, careful analysis of the domain reveals the domain structure which forms the learning goal

hierarchy. Then start generating possible protocols of ITS interaction, reflecting different stages of learning indicated in the tree. Some modifications of the curriculum tree may be needed to show the structure of the protocols. The actual implementation of the protocols is the construction of the rules of behavior and scheduling rules. Now, we may focus on a particular episode and develop rules of behavior for different agents in that episode. To test the protocol, we may write scheduling rules in the scheduling nodes above the episode node so that the system will go down from the root node to the episode node and run it.

As can be seen from the construction of curriculum tree to the rules of behavior of an episode, the development process is top down. After debugging, some of the rules of behavior developed for that episode can be extracted out and cached in the upper nodes, leaving those rules that are particular to the episode remain below. Rules that have been collected on the upper nodes can be regarded as the rules for simulating the protocols which are independent of particular episodes. This is a bottom up process. The problem solver which is regarded as a supporting function for the rules of behavior may then be built. As can be seen, during the development, the curriculum tree is expanded incrementally on both sides and downwards. Finally, after testing the system with human subjects, modifications at different levels — curriculum tree structure and rules of behavior, are needed. In particular, constructing remedial episodes and new scheduling rules may improve the adaptability of the system to different students.

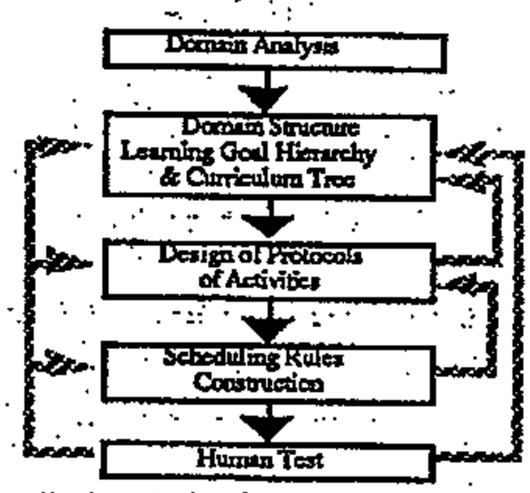


Fig. 3. Development Process of ITS

6 Discussion

Although the curriculum tree is an authoring shell for building ITS, the domain knowledge component remains, as expected, to be developed by the designer, asides from the rules of behavior. Besides, we observe two limitations of the curriculum tree. The curriculum tree seems to be a natural architecture to develop ITS for formal subjects such as integration. But, for some informal subjects such as diagnosing circuits where the focus is a particular skill, the curriculum tree seems to be less useful. The second limitation we noticed is that the power of the scheduling facility of the curriculum tree has not been fully exploited in Integration-Kid. We believe that there are two reasons. First, Integration-Kid does not maintain an explicit student model so that abstraction power of student model in the tree has not been used. Second, unlike some ill-structured domains, for example, psychology, where scheduling is more complex and demanding than the well-structured domain of integration, it is tempting to simplifying the scheduling part of the system.

Apart from being a natural architecture for formal subjects, the curriculum tree illustrates several important advantages which we believe are a step towards building a large scale ITS:

(1) *Clarity.* With the curriculum tree construct, the procedural knowledge of learning activities can be decomposed by the domain conceptual dependency knowledge. The decomposition provides the author of the system a clear view of the flow of control. Furthermore, since visual examination is possible, the system can be modified and understood more easily. The whole picture of the system is not lost in a tangled web of specific rules, nor is it hidden in many lines of procedural code.

(2) *Simplicity.* The correspondence between a domain concept and the tactic used to teach a particular skill is a conceptual simplification for designing learning activities. Also, noting that episodes are discrete events, the curriculum can be divided into separate episodes. The process of learning discourse can be viewed as an alternating sequence of running episodes and scheduling. Thus rules can be distinguished either as simulating behavior or as scheduling, in place of complex rules where the conceptual knowledge is implicitly embedded. All these simplicities are due to the separation of domain conceptual knowledge and procedural knowledge achieved by the curriculum tree.

(3) *Maintainability.* Modularity, reusability, and traceability of the curriculum tree contribute maintainability. Production systems are well known for modularity since every production can be regarded a module. Distribution of rules over the nodes of the curriculum tree provides another level modularity. Every node is a module of rules with the node name as the description of module. Also, every leaf node is a particular module since it can be regarded as a self-contained knowledge-based system. Reusability of rules is attained by rule inheritance. Because of rule inheritance, modifying a rule of behavior in an episode node will only affect that episode, but changing a rule of behavior in a scheduling node may affect all the episodes beneath it. Such simple propagation of the change makes modification simple and effective. Finally, traceability is strongly supported by the use of a tree editor as the developmental environment. It makes the program toward better structure. For example, reconfigurations of the curriculum tree such as adding or deleting a subtree or an episode are straightforward. Furthermore, since human beings are much better at picking out errors and omissions from a graphical or visual representation than from listings, the visual identification of locations to be changed and the visual confirmation of those changes reduces errors. Thus, not only can the designer make changes faster, fewer errors are likely to be introduced in the process of making those changes; thus enhancing the debugging process.

(4) *Fast Prototyping.* We can lay out a first draft of a curriculum tree with some episode nodes at the bottom based on some domain structure. During development, we are able to intertwine top down and bottom up construction of the tree. This simple, natural, and flexible style of development is the key for fast prototyping of ITS.

(5) *Efficiency.* Although the characteristics of curriculum tree representation do provide a certain comprehension efficiency for the author, the more common view of efficiency focuses on the computer system that would execute. Efficiency in processing performance is mainly attained by running a smaller number of rules in an episode which is due to the decomposition of the learning activities by the curriculum tree into separate episodes. The processing of such a system can be viewed as a sequence of running an episode efficiently and then choosing another set of relevant rules for another episode and so on. This effectively meets the efficiency demand of the

student where there is quick response from the system when he is learning in an episode and a more relaxed response when the episode has finished.

References

1. T. W. Chan, A. B. Baskin: Studying with the Prince : The computer as a learning Companion. International Conference of Intelligent Tutoring Systems, 1988, June, Montreal, Canada, pp.194-200, 1988
2. T. W. Chan, A. B. Baskin: Learning Companion Systems. In C. Frasser & G. Gauthier (Eds.) Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education, Chapter 1, New Jersey: Ablex Publishing Corporation, pp.6-33, 1990
3. T. W. Chan: Integration-Kid: A Learning Companion System. The 12th International Joint Conference on Artificial Intelligence, Sydney, Australia, Morgan Kaufmann Publishers, Inc., pp. 1094-1099, 1991
4. T. W. Chan: Some Techniques for Building Mathematical Intelligent Tutoring Systems. In H. Nwana (Eds.) Mathematical Intelligent Learning Environments, Intellect Books (to appear), 1992
5. S. J. Derry, L. W. Hawkes, & U. Ziegler: A plan-based opportunistic architecture for intelligent tutoring. International Conference of Intelligent Tutoring Systems, Montreal, Canada, pp. 116-123, 1988
6. W. Dick, L. Carey: The systematic design of instruction. Glenview, IL: Scott, Foresman & Company, 1985
7. A. Lesgold, Curriculum for intelligent tutoring systems, in Mandal & Lesgold (Ed.), Learning issues for intellignet tutoring systems, 1988.
8. S. A. MacMillan, D. Emme, & M. Berkowitz: Instructional planner: Lessons learned. In J. Psotka, L. D. Massey, & S. A. Mutter (Eds.), Intelligent Tutoring Systems: Lessons Learned, pp. 229-256, Hillsdale, NJ: Lawrence Erlbaum Associates Publishers, 1988
9. W. R. Murray: A Blackboard-based Dynamic Instructional Planner. The Eighth National Conference on Artificial Intelligence, AAAI Press/The MIT Press, pp.434-441, 1990
10. O. Park, R. S. Perez, & R. J. Seidel: Intelligent CAI: Old wine in new bottles, or a new vintage? In G. Kearsley (Ed.), Artificial Intelligence and Instruction: applications and methods, Reading, MA: Addison-Wesley, 1987
11. D. R. Peachey, G. I. McCalla: Using planning techniques in intelligent tutoring systems. International Journal of Man Machine Studies, Vol. 24, pp. 77-98, 1986
12. D. M. Russell, T. P. Moran, & D. S. Jordan: The Instructional Design Environment. In Intelligent Tutoring Systems: Lessons Learned, Lawrence Erlbaum Associates, pp.203-228, 1988
13. B. Wasson: Content Planning for Intelligent Tutoring, World Conference on Artificial Intelligence in Education, 1989
14. P. H. Winne, L. L. Kramer: Representing Knowledge about Teaching: DOCENT -- An AI Planning System for Teaching and Learning. Intelligent Tutoring Systems: At the Crossroads of Artificial Intelligence and Education, Chapter 8, New Jersey: Ablex Publishing Corporation, pp.162-187, 1990
15. B. Woolf, T. Murray, D. Suthers, K. Schultz, Knowledge Primitives for Tutoring Systems, International Conference of Intelligent Tutoring Systems, Montreal, Canada, pp.194-200, 1988