Advice with Part-Whole and Precedence Relations in Task Graphs for Intelligent Tutoring Systems

Jean-Yves Djamen
Diantre R&D inc.
djamen@cam.org

Francois Pachet
LAFORIA
pachet@laforia.fr

Abstract

This article shows how advice can be enhanced in Intelligent Tutoring Systems (ITSs). Our approach is based on the systematic exploitation of two relations in task graphs : part-whole and temporal precedence. The first relation describes the decomposition of tasks into sub-tasks, and distinguishes concrete actions from abstract tasks. The second relation describes temporal constraints between tasks. The underlying reasoning mechanism can easily communicate any given task to the human learner. Moreover it performs the analysis of his/her actions during the solution of a problem. This analysis is used to generate relevant advice in a tutoring context. The mechanism is general and can be applied to arbitrary task graphs, thereby endowing the ITS with a non trivial advice-generating capacity.

1 INTRODUCTION

Cognitive task analysis is an important phase in the development of an Intelligent Tutoring System (ITS). Such an analysis (see for example [5, 4]) typically results in the identification of one or several graphs representing, among others, teacher's (and/or learner's) knowledge (or responses) in a given domain (or problem solving). Those graphs, also known as *task graphs*, can describe specific tasks to be done during the tutoring session and thereby help in several tutoring activities such as assessment.

Although the analysis phase is known as important and incorporated in the design of several ITS projects, exploiting their results is yet another problem to solve during the development of ITSs.

In fact, representing analyzed tasks in an effective intelligent program can be very complex, indeed if prominent features of tutoring (such as presenting the material at a good level of detail with regard to learner's actions, or recognizing alternative solutions to the same problem) are to be taken into account. To illustrate some of the key points to be addressed during the representation of analyzed tasks, let us consider a problem P, whose completion requires the execution of a task [T] made up of three sub-tasks [A], [B] and [C]. The Part-Whole relation of [T] can be represented as shown in figure 1, i.e. sub-tasks of a given task are described as child nodes.

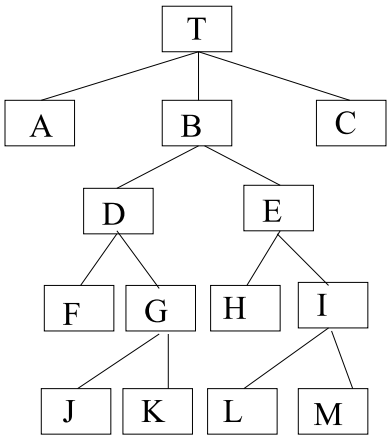

Figure 1: A typical task graph.

We will later refer to a task made up of sub-tasks as an *abstract task* or simply a *task*. A task with no child is a *concrete task* or simply an *action*. We further suppose that some precedence relations have been defined between sub-tasks of [T] as follows:

- Sub-tasks of [T], [D], [E], [G] and [I] must be done in the specified order;

- Sub-Tasks of [B] must be done regardless of any execution order.

Therefore, resolving the problem P will lead to one of the following sets of actions:

1. (A), (F), (J), (K), (H), (L), (M), (C)
2. (A), (H), (L), (M), (F), (J), (K), (C)

Such a representation can easily be handled in most artificial intelligence (AI) programs. Particularly in ITS, teaching may be faithfully represented as a top down traversal of the graph.

However, the top-down representation is not suitable for assessment. In fact, the underlying reasoning process leads most of the time to huge computational complexity [1].

Moreover, an accurate cognitive task analysis will always result to much more complex relations between tasks. For example cognitive task analysis of P may also incorporate the following constraints:

- Resolving [T] can be reduced to resolving (A) and (C) in some context (especially when the task [B] has been executed previously);
- [D] can be interrupted by a partial execution of [E] and vice-versa.

With regard to the latter constraint, there exists an infinite number of sets of actions that can be given by a trainee (or any appropriate automated process) in order to solve P. For example the following set, that may incorporate an endless loop on (F) (H), is a valid solution that solves P:

- (A), (F), (H), ... (F), (H) ... (F), (H), (L), (M), (J), (K), (C)

In this consideration, the following sets will not solve P if an interruption on either [D] or [E] is likely to force the re-initialization of their execution, or more generally if the execution of [D] can influence in some way the execution of [E]:

- (A), (F), (H), (L), (M), (J), (K), (C)
- (A), (H), (F), (J), (K), (L), (M), (C)

Therefore, changing the execution order of certain tasks can result to a non execution of some of their sub-tasks.

In real applications a given task can influence other tasks in several ways, thereby leading to multiple alternative solutions with regard to a single problem. As a result, interpreting learner's actions is still problematic in ITS developed so far.

2 Task graphs in ITSs

Taking into account several temporal constraints is not strait-forward in the development of an ITS. In fact this problem is already known in AI, where the misrepresentation of a piece of knowledge can lead to a combinatorial explosion during the reasoning process [7, 8].

Since a reasoning mechanism is tightly related to the representation [3], the problem can also be viewed as the one of chosing between a *detailed representation* (Cf. figure 2) and a *concise representation* (Cf. figure 3) of a task graph. A detailed representation can be difficult to handle in a huge graph. It may also incorporate some wrong copies of a given task [1]. On the other end a concise representation incorporates unexpected situations such as loops.

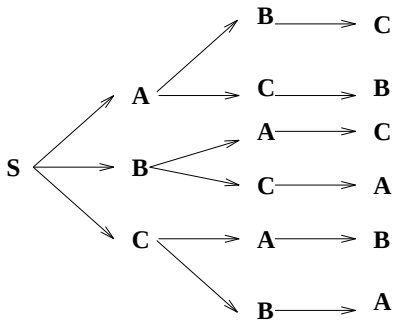

Figure 2: Detailed representation.

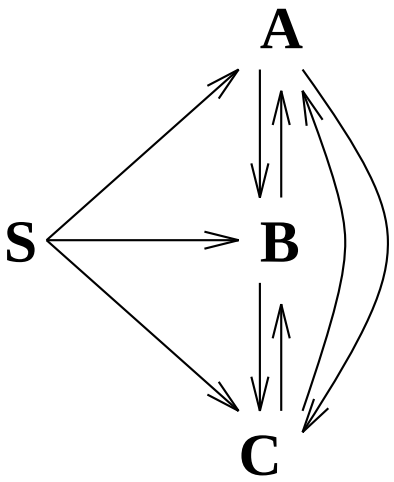

Figure 3: Concise representation.

More precisely, problems to be addressed in ITS task graphs include the following:

- Level of abstraction of tasks;
- Relation between tasks and their sub-tasks;
- Computational complexity (reasoning part of the graph);
- Synthesis of results;
- Integration of idiosyncratic advice;
- Structural identity and physical identity of tasks;

In some well known ITSs, such as Sherlock [6], some of these problems (also described as knowledge representation problems) have been solved by hard-coding some solution paths.

3 Sub-tasks behavior class approach

Our approach to solving the knowledge representation problems listed above consists in assigning a so-called *behavior class* to abstract tasks. A behavior class enables the specification of constraints that can hold only on direct sub-tasks of a given task. For instance, the behavior class <STRICT-AND> will be associated to [T] to specify that all its sub-tasks, i.e. (A) [B] (C), must be done with regard to the specified order. Figure 4 shows the representation of solving the problem P using our approach. For illustration purposes, nodes in figure 1 have been instanciated as follows:

- [T] = "Making an infusion with the following rates: volume V = 25 ml and rate R = 35 ml/h"
- (A) = "Press the On-Off/Charge button"
- [B] = "Enter rate and Volume"
- (C) = "Press Start"
- [D] = "Enter Rate 35"
- (F) = "Press Rate Key"
- [G] = "Enter fraction 35"
- (J) = "Press key pad 3"
- (K) = "Press key pad 5"
- [E] = "Enter Volume 25"
- (H) = "Press Volume Key"
- [I] = "Enter fraction 25"

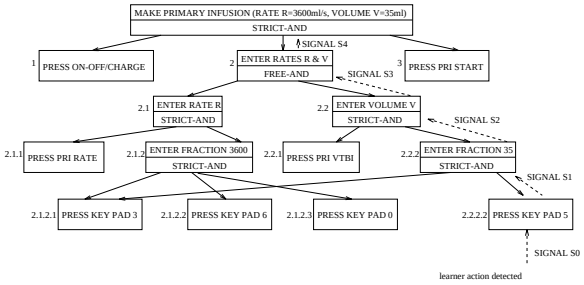

Figure 4: Representing a task using behavior class approach.

- (L) = "Press key pad 2"
- (M) = "Press key pad 5"

A task is therefore simply represented using the grammar below (the syntax is following the Bacchus Naur form), where a list of basic classes can be defined and upgraded with minimum changes in the program:

BEHAVIOR-CLASS ::= '(', TASKS, ')', BASIC-CLASS
TASKS ::= TASK-LIST; TASK-LIST, BEHAVIOR-CLASS
BASIC-CLASS ::= 'Free-And' | 'Strict-And' | 'Free-Or' | 'Exclusive-Or' | Optional-N | Strict-N-Over-M ...
TASK-LIST ::= TASK, TASK-LIST | TASK
TASK ::= 'Task1' | 'Task2' | ...

We have built an algorithm (dubbed QCOM.I) on top of behavior class representation [9]. In fact every time the learner's action is detected, QCOM.I sends a signal to the corresponding node in the graph. Several cases are then possible:

- The action is legal (i.e. it belongs to the list of next possible actions obtained by a top-down traversal of the selected task graph);
- The action is not legal (no corresponding node can be found in the selected graph);
- The action is ambiguous (multiple corresponding nodes are present in the graph);
- The action is not ambiguous (only one corresponding node is present in the graph).

By receiving a signal each node:

1. produces comments with regard to the received signal;
2. changes its state if the action is legal (a concrete task in the graph can either be inactive or terminated while an abstract task can be inactive, terminated or pending);

3. Ascends the signal to the parent node (up to the root).

For example, an action on (F) will create the following trace:

- **(F):** Terminated;
- **[D]:** Pending;
- **[B]:** Pending;
- **[T]:** [B] is pending prematurely, execute (A) first.

The overall advice will be summarized by *"Should not execute task [B] since action (A) is not yet done"*. As can be noted here, this advice is at a good level of granularity. In fact, QCOM.I has detected a pending task after the execution of an action and hence has produced an advice using the appropriate abstract tast (i.e. [B] and not [D]).

A failure within the trace, i.e. a violation of a constraint defined in the behavior class will enforce the restoration of previous states (i.e. before the given action). For instance an action on (A) will generate the following changes:

- **(A):** Terminated;
- **[T]:** Pending.

This action will not produce an advice. However a verbose mode of QCOM.I will generate the following text: "action (A) has been done successfully".

Comments are treated in context by QCOM.I in order to produce a meaningful advice during a given tutoring session. Some comments are qualitatively described as follows:

- Already (action or task already done)
- Pending (Task is pending)
- PendingAlready (task is already pending)
- PendingNotTerminated (Task is pending and is not terminated yet)
- PendingPrematurely (Task is pending prematurely)
- Premature (action or task done prematurely)
- Terminated (action or task terminated)

4 CONCLUSION

We have shown a way to provide advice in an ITS using part-whole and precedence relations defined in a task graph. Our approach enables the enlargement of temporal constraints that can be defined into (and treated by) a task graph. More generally the behavior class approach is usefull in numerous ways:

- It provides a means of decomposition of tasks in a task graph. With this regard, abstract tasks can be viewed as placeholders for general comments to be given to the trainee.
- It provides a way to insert constraints between sub-task with regard to defined goals. With this regard, constraints are described within a given abstract task, therefore preventing a proliferation of constraints (links) in task graphs.
- It provides a simple way to compute (expected or past) actions/tasks vis-à-vis expected solutions. With this regard, t
- It provides an easy way to incorporate comments with regard to expected activities. With this regard, the same ITS can serve for advice, critic, etc.

However, our approach can not support all precedences between tasks. In fact, it has shown not to be effective when dealing with some conditions; such as those enabling both concurrent and interruptible sub-tasks of a given task. However we have addressed some of these problems elsewhere [2].

References

[1] Jean-Yves Djamen. *Architecture de système tutoriel intelligent pour l'analyse du raisonnement de l'apprenant.* PhD thesis, Université de Montréal, Département Informatique et Recherche Opérationnelles, 1995.

[2] Jean-Yves Djamen, Marc Kaltenbach, and Claude Frasson. Qualitative comments with physical systems for intelligent tutoring systems. In *Proc. of the Eighth Florida Artificial Intelligence Research Symposium*, pages 304–308, Melbourne Beach, FL., April 1995.

[3] Kenneth D. Forbus. Intelligent computer-aided engineering. *AI Magazine*, Fall:23–36, 1988.

[4] N. Frederiksen, R. Glaser, A. M. Lesgold, and M. Shafto. *Diagnostic monitoring of skill and knowledge acquisition*. Lawrence Erlbaum Associates, Hillsdale, 1990. edited book.

[5] Suzanne P. Lajoie and Sharon J. Derry. *Computers as cognitive tools*. Lawrence Erlbaum Associates Publishers, 1993. edited book.

[6] Alan Lesgold, Suzanne P. Lajoie, M. Bunzo, and G. Eggan. SHERLOCK: A coached practice environment for an electronic troubleshooting job. In Jill H. Larkin and Ruth Chabay, editors, *Computer Assisted Instruction and Intelligent Tutoring Systems: Shared goals and complementary approaches*, pages 201–238. Lawrence Erlbaum Associates Publishers, Hillsdale, New Jersey, 1992.

[7] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 6, pages 463–502. American Elsevier Pub. Co., 1969.

[8] Hyacinth S. Nwana and Ray C. Paton. Drawing from the shortcomings of artificial intelligence: Some critical issues connectionism must address. In F. Attia, S. Flory, S. Hashemi, G. Gouardères, and J. P. Marciano, editors, *EXPERSYS*, pages 37–43. I.I.T.T. International, Paris, 1992.

[9] F. Pachet, J. Y. Djamen, Claude Frasson, and Marc Kaltenbach. Production de conseils pertinents exploitant les relations de composition et de précédence dans un arbre de tâches. *Techniques des Sciences Éducatives*, 3(1), 1996.