Appeared in: P. Brna, S. Ohlsson & H. Pain, editors. World Conference on AI and Education, pp. 330–337. AACE, Charlottesville, VA, 1993.

Automatic Generation of Optimal Learning Routes

RADBOUD WINKELS and JOOST BREUKER
Dept. of Computer Science & Law, University of Amsterdam
Kloveniersburgwal 72, 1012 CZ Amsterdam, The Netherlands
E-Mail: winkels@lri.jur.uva.nl

Abstract: In this paper we will discuss the nature of a curriculum in Help Systems, how it can be used to guide users in opportunistic coaching, and, most importantly, how it can be generated automatically for EUROHELP domain models that do not provide such a didactic view. In EUROHELP, a shell for building Intelligent Help Systems for computer applications, curricula take the form of a network of Didactic Goals. These Didactic Goals form the basis for expanding the current knowledge of the user of the system. Their major function is the specification of knowledge which can more easily be acquired by a user by applying some learning principle. The learning principles are specified in the form of types of relations between concepts. Five types of relations are distinguished. For a given domain representation a Didactic Goal Generator automatically deduces all didactic relations between concepts off-line. The mechanisms described are domain independent. The feasibility of the approach was tested in an implementation for part of the Unix Vi editor, and evaluated by comparing the results to those of human tutors.
Keywords: Help Systems, curriculum generation, Didactic Goals

Introduction

When teaching a substantial and complex domain, teachers have to organize the material in semesters, lessons, topics, etc., and to sequence these according to inherent properties of the domain and their tutorial goals. This is usually termed: The curriculum. Computers teaching students will have to take the same kind of decisions, but so far most research in this field did not address the issue (cf. (McCalla, 1990)). In traditional CAI and most existing Intelligent Tutoring Systems (ITS), the curriculum is fixed, if present at all. People were interested in other issues, like knowledge representation, communication, and student modelling. There is however, a recent interest in the dynamic planning and adjusting of the curriculum on the basis of a student's performance (e.g. (Brecht, 1990; Murray, 1989)).

The little ITS research that deals with curricula, is about representation (e.g. (Goldstein, 1979; Lesgold, 1988)), and planning (Brecht, 1990; Murray, 1989). These researchers describe how the curriculum is represented, what kind of links between domain concepts are used, and how the resulting structure is traversed for a particular student.

In the EUROHELP project, aimed at developing a shell for building Intelligent Help Systems (IHS) for computer applications, we were faced by somewhat different curriculum issues (Breuker, 1990; Winkels, 1992). Users of computer applications bring their own tasks and goals to the system, so the Help System can only teach within the context of these tasks. It will have to wait for *occasions* to expand the users' knowledge or remediate misconceptions (opportunistic coaching). Therefore, the curriculum cannot take the traditional prescriptive form of a decomposition of instructional goals, all the way down to tasks and exercises at the domain–level (cf. (Lesgold, 1988), Fig. 1). At best it could suggest what topics to introduce given a user's state of knowledge and current task.

EuroHelp is a *shell* for building IHS; it contains all domain–independent functionality of Help Systems plus a formalism and tools to fill the shell for a particular application. A developer of an IHS has only to fill one central knowledge base – nevertheless a substantial task –, that will be interpreted by specific modules to generate different views on the domain (Winkels *et al.*, 1991). One of these views will have to be an "optimal" learning point of view, e.g. what topics should preferably be introduced before others. Therefore our research focussed on specifying domain independent relations between domain concepts that reflect such an optimal learning point of view, and how to automatically generate these on the basis of a domain representation that does not explicitly contain such information.

The results of this research will be discussed in this paper. We tested our approach in an implementation for part of the Unix Vi editor, and evaluated it by comparing the results to those of human tutors. We will start with a short description of what EuroHelp domain models look like.

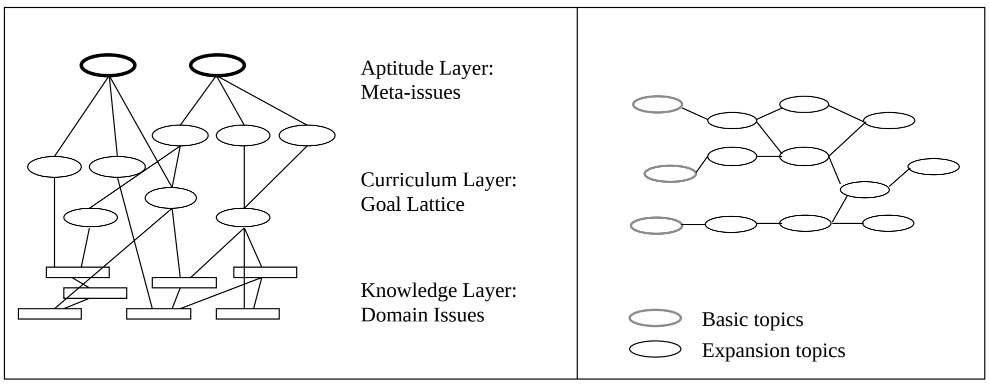

Figure 1: A true decomposition (left; Lesgold, 1988) vs learning trajectories (right)

Domain Representation

Users of computer applications have to learn how to use the application (skill), and how the application works (insight). EuroHelp Domain Models reflect these different types of knowledge, by representing them separately as *operational* and *support* knowledge resp. (Clancey, 1983). The working of applications has been modeled by representing both the objects available within an application and the commands that act upon them. Objects are characterized by their attributes and their relations to other objects. There are two kinds of relations between objects: The "is–a" relation for typing objects, and set relations (e.g. "consist–of" for unordered sets; spatial relations like "before" for ordered sets). The commands consist of one or more "primitive actions" that operate on (sets of) objects (Winkels *et al.*, 1991). Examples are: `create(Object), add(Set, Object), show(Object)`.

The representation of the operational knowledge consists of tasks that represent user intentions. How these tasks can actually be performed is represented by plans containing sub–tasks in a multi level model. Basically the representation formalism is an AND/OR tree (see: (Breuker, 1990) for more details).

Didactic Goals

A EuroHelp curriculum takes the form of a network of Didactic Goals. They form the basis for expanding the current knowledge of the user of the system. Their major function is the specification of knowledge which can more easily be acquired by a user by applying some learning principle. The learning principles are specified in the form of types of relations between concepts. In this sense, Didactic

Goals have a similar representation format as *genetic graphs* (Goldstein, 1979). For a particular user the Didactic Goals consist of relations between knowledge the user already has, as reflected by the User Model and new topics in the domain. For a particular application, the Didactic Goals do provide a partial ordering of topics, but the relations are heterogeneous (see below) and often a concept will be linked to more than one other concept by more than one type of relation. The Didactic Goals do not specify conditions for following one link or another (e.g. like in more traditional CAI), they only specify the "learning principle" that will facilitate the acquisition of a new concept given a known one. Other information, i.e. the current task of the user and the state of the User Model, will cause one or another link to be preferred. Usually this means the link to topics that facilitate performance will be chosen.

The Didactic Goals serve two purposes: Facilitate learning and facilitate performance. These two goals are closely related. Performance will promote learning, learning will enhance performance. Learning should be facilitated because specifying the (didactic) relation between domain concepts aids the structuring and mental representation of the application by the user. The didactic goals facilitate performance, because in general they lead from more generally applicable concepts to more specific ones.

We distinguish five types of didactic relations:

Generalization – Specification: Some concepts are more general than others. They share attributes, but the more specific ones have extra attributes. All the shared attributes can be transferred from the general concept to the more specific one if the Help System explicitly mentions this specification relation. This facilitates acquisition. Another way in which commands can be more general or specific than others, is by the objects they act upon. Commands that act upon objects higher in the object hierarchy are more specific than commands that affect objects lower in the hierarchy. E.g. "`delete word`" is more specific than "`delete character`", at least from a performance point of view: Deleting a word can be achieved by deleting all the characters, but there is no way you can delete a character with the "`delete word`" command (unless it is a one character word).

In general the direction of teaching will be from general to specific, both from a learning point of view and a performance point of view.

Inversion: Instead of transferring shared attributes, inversion allows one to transfer the inverse of attributes, e.g. "X is the opposite of Y". We can distinguish two kinds of inversion:

General inversion: A general way of *undoing* a previous action, e.g. the "`u(ndo)`" command in Unix–Vi, or "`%Z`" for the Macintosh, but also global "undo" commands like the "`q!`" in Vi that quits the editor without saving all the changes that have been made since the last "write".

Specific inversion: One action is the inverse of another action, e.g. the cursor–left and cursor–right commands.

This relation is a good illustration of the difference between the EuroHelp approach and that of (Goldstein, 1979). Since there is no apparent evolutionary link between two "inverse" concepts, Goldstein does not use this link. In a Genetic Graph it would either show up as an analogy, or not at all. Because we are looking for links that may improve or facilitate learning, the inversion relation is a powerful one.

Analogy: Explaining domain concepts by using metaphors. The idea is to have the user take a known structure from one (part of a) domain to the new area. For Help Systems we can distinguish three forms of analogy. First of all there is analogy within the application domain. In this view "specific inversion" is a form of analogy. Secondly there is analogy within the general EuroHelp domain of computer applications, e.g. comparing the "`ZZ`" command to leave the Unix Vi text editor to the "`q`" command to leave Unix mail. Finally there is analogy across very different domains, e.g. real world metaphors like comparing an electronic mail system to writing and sending a letter in the "real world". The first two forms of analogy we can handle, the third introduced too many problems to solve in any generic way within the EuroHelp project.

Similarity: A peculiar version of "analogy" or the "generalization – specification" relation. Two concepts are exactly the same, apart from their name. Or rather, two names refer to the same concept.

Abstraction – Concretion: Finally, because EuroHelp domain models contain two kinds of knowledge, support and operational (see above), there have to be links between these types of knowledge for the same concept. One of the "principles" of coaching in Help Systems is exactly to alternate these two kinds of knowledge (Winkels *et al.*, 1988). This is achieved through the abstraction – concretion relation. When certain procedures are reasonably well practiced by the user (*skill*), it is time to explain some underlying, hidden objects and models, i.e. to introduce some support knowledge (*insight*): abstraction. Also, it may be desirable to illustrate an explanation of how part of a system works (support knowledge) by telling the user how he can use it (operational knowledge), thereby specifying a concretion relation. An example may clarify this. Suppose a Unix Vi user has been deleting text (characters, words, lines) quite a lot and has also moved it by "putting" it elsewhere, but does not know how this works, i.e. she does not know anything about buffers in general or the specific workings of the buffers in Vi. Then the concept of a "buffer" will become a Didactic Goal. It is the common (hidden) object in the effect descriptions of these commands (deleting text results in moving it to the delete buffer; putting text copies the text from the delete buffer). The "abstraction" will be linked to all move plans the user knows. As soon as she uses one of these plans, the Help System will spot this occasion for an expansion. Telling the user about buffers may in turn trigger telling about how to address these buffers (Vi has ten addressable delete buffers that operate in a stack like fashion), so giving some new operational knowledge (concretion).

Generating Didactic Goals

As soon as a domain model exists, the didactic view on the representation can be generated. This is done by the Didactic Goal Generator (DGG), i.e. off–line. The DGG can be considered a tool for building IHS, i.e. it is part of the shell, but not of a stand–alone Help System. The method for generating them is straightforward: The Didactic Goal Generator compares all pairs of concepts in the domain model and tries to find a didactic relation between them.

Generalization – Specification: There are several ways this relation can be deduced from the domain model:

1. variabilisation vs. instantiation, e.g. "move to the right" versus "move 1 position to the right". This can be deduced from the "value restrictions" in the representation, i.e. more restricted concepts are more specific than less restricted ones. This is the way (Goldstein, 1979) operationalises his "generalization – specialization" relation.
2. super–concepts vs. subconcepts in the *isa* hierarchy, e.g. "delete command" versus "delete–word command" through: `isa(delete-command, delete-word-command)`.
3. subconcepts vs. super–concepts in the *consist-of* hierarchy, e.g. "delete a character" versus "delete a word" through: `consist-of(word, character)`. As explained above, concepts lower in the consist-of hierarchy are more general than the ones higher up.
4. high level vs. low level in the task–plan hierarchy that reflects the operational knowledge in EuroHelp domain models. So for instance, first tell about the idea of "moving" an object, then about the "delete and put" operations.
5. few goals, or effects, and conditions vs. many goals, or effects, and conditions, e.g. "delete line" versus "delete line from position X", because the second one has the extra condition that the cursor be at the specific position X.

Application of these methods may lead to many possible relations between domain concepts, not all of which are particularly interesting or useful. One way to constrain this is to add priorities to the different methods. E.g. we are first interested in similarities between commands concerning the deletion and creation of objects, or a change of modes, only later in cursor movements.

Inversion: Some specific inversions can immediately be found in the effect descriptions of commands:

create(Object)	$\Leftrightarrow$	delete(Object)
add(Set, Object)	$\Leftrightarrow$	remove(Set, Object)
open	$\Leftrightarrow$	close

Others will have to be "planned" by the EuroHelp Backward–Chaining Planner (Breuker, 1990) that also has to plan "undo–plans" during sessions with the Help System when the user has made a so–called "executable error" and undesired effects have occurred.

Analogy: The very restricted form of analogy, i.e. within the particular application, is partly dealt with by the specific inversions above. In this way there is a mapping between "move to the end of the line" and "move to the beginning of the line", but specifying this through the inversion link is stronger than just stating the analogy. A second type of intra–application analogy can be used to link two system procedures that achieve the same effects in different modes, e.g. deleting a character in "insert mode" and in "command mode" in Vi. Analogies across application domains could be detected at higher operational levels in the task–plan hierarchies, e.g. general "leave mode" procedures, but whether these analogies will be helpful or only confusing is another matter. Therefore, these analogies will have to be explicitly represented by the Help System developers.

Similarity: In order for two concepts to be "identical", all attributes should be the same, i.e. for commands, all effect descriptions and conditions should be identical.

Abstraction – Concretion: The need for abstraction will have to be inferred from the operational knowledge the user is supposed to have acquired, as reflected in the User Model. In the Domain Model two support views of the domain are given: One consists of the effect descriptions of commands, the other of some partial models of the specific application, e.g. a buffer–model. This last kind of support knowledge will have to be added to the domain model by hand, there is no general mechanism for describing models in EuroHelp. The didactic relations between operational knowledge and these pieces of support knowledge will have to be added by hand as well, they cannot be derived automatically.

For the effect descriptions, linking support to operational knowledge can happen via the common (hidden) objects, e.g. like the "delete" and "put" example given earlier that share the object "delete–buffer". This will at least be necessary for all objects in the domain. Once the implicit use of such an object exceeds a given threshold, abstraction is appropriate. This "counting" of implicit use is *inherited* through the is-a hierarchy, e.g. implicit manipulation of the "delete flag" and the "read flag" in Unix mail, will also increment the use of the more general concept of "flag", so the abstraction might be towards this more general concept instead of to one or both of the more specific ones. For models, the threshold is higher than for single objects, because after all, more concepts will have to be explained.

For a specific user for a specific session, the subset of Didactic Goals that are one or two links away from the user's knowledge state as reflected by the User Model, are selected (the "Knowledge frontier", (Goldstein, 1979)). As soon as the user could apply the new knowledge, the Help System will seize the occasion and supply the information. Suppose, for instance for the Vi–domain, that the User Model states that "`cursor-right`" is well known (operationally) within the context (plan) "move cursor one word to the right". The Didactic Goals specify relations between "`cursor-right`" and "`w`" (jump to next word) – specification –, and between "`cursor-right`" and "`cursor-left`" – inversion –, but "`cursor-left`" is also well known. Then "`w`" becomes an occasion for expansion, and the performance trigger is "`cursor-right`" as part of a plan to move a word to the right. As soon as the user uses the "`cursor right`" command in this context, the Help System will use the opportunity to tell the user about the "`w`" command.

If the user's performance (or question) gives an opportunity for several expansions, i.e. there is more than one didactic link to new domain topics, the most readily applicable, performance oriented one is chosen. This means for instance that generalizations and specifications will be preferred over abstractions or analogies. As always, the IHS is performance, rather than learning biased.

Evaluation

To see if the approach to curriculum generation described is feasible, a simple version of the Didactic Goal Generator (DGG) was implemented and tested on part of the domain of the Unix Vi editor (van derLaan, 1989). Because the implemented DGG only worked on effect descriptions of commands (support knowledge), not on task–plan hierarchies, not all of the methods for generating didactic goals from a domain model (AM), that were described in the previous section, were used. For generating "generalization – specification" relations, only the third and fifth method are used: Subconcepts versus superconcepts in the consist–of hierarchy, and few effects and conditions versus many effects and conditions. Analogies are only generated for commands that achieve the same effects in different modes. Similarity relations were not yet distinguished at that time, but are easy to generate. For forty–five commands of Vi, the DGG generated eighty–six didactic relations between them. Table 1 gives the distribution over the four types of didactic relations. Clearly, the generalization – specification relation is generated most. Figure 2 gives a graphical representation of most of the generated trajectories for three types of relations. The commands are ordered from most "general" to most "specific" from left to right, on the basis of the "specification" relation. For clarity reasons the abstraction – concretion relations are not presented in the figure. Nearly all of these link "delete" and "put" commands to support descriptions of the buffers in Vi.

Didactic relation	**Nr.**
generalization – specification	46
abstraction – concretion	17
inversion	18
analogy	5
Total	86

Table 1: Distribution of generated didactic relations

Clusters of topics, linked by didactic relations, can easily be distinguished in Fig. 2. The two left clusters consist of commands for moving around in Vi, the top–right cluster consists of delete, put, and change commands, and the bottom–right cluster of insert–mode commands. It is also clear that not all didactic relations were generated that would have been possible, partly because not all methods were implemented. E.g. a specification relation between "`a`" (append text after cursor) and "`A`" (append text at end of line), is missing, because the DGG could not relate "move cursor one position to the right" to "jump to end of line" given the domain representation used. Another aspect Fig. 2 shows, is the commands that should be part of a basic instruction before users are confronted with the Help System, and that will provide the starting points for opportunistic expansion of the user's knowledge. They are those to the left of the different clusters, e.g. the "`h`, `l`, `j`, and `k`" cursor–commands. They are most general, and from there all other movement commands can be reached. From an efficiency point of view, some commands to the very right of larger clusters could be introduced as well, e.g. the search command "/".

The learning trajectories the DGG generated were compared to orderings of the same domain topics by four human experts on Vi. There were no significant differences. We also compared the generated

Didactic Goals with those used by a human tutor in a long–term empirical study with students learning Vi (Winkels, 1992). Again, there were no significant differences.

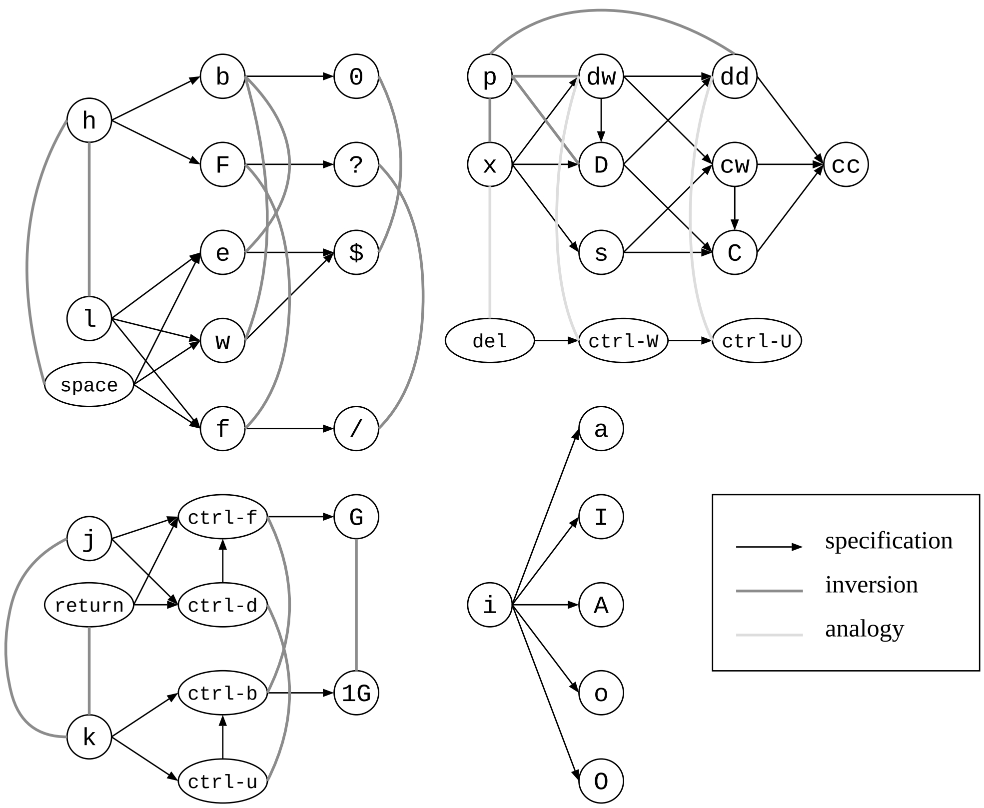

Figure 2: The generated didactic goals for Vi

Conclusion

We have presented generic mechanisms for automatically generating a didactic, or "optimal learning" point of view given a domain representation that does not explicitly contain such information. The resulting network of Didactic Goals can be used to guide users or students in an opportunistic coaching paradigm through the domain. It does not constitute a tutoring plan, but indicates possible (partial) routes through the domain. With a simple extension it can be used as the basis for a true curriculum, e.g. first travel all specification links, then do abstractions, etc. The resulting "curriculum" will not be as strict as specified by for instance (Lesgold, 1988), nor as sophisticated and tuned to the specific domain (unless the Help System Builder incorporates this detailed information by hand), but that is the price we have to pay for the generality.

Identification of the abstraction–concretion relation as a link between operational and support knowledge from a didactic point of view, is an interesting elaboration of Clancey's distinction of these two types of knowledge.

An implemented simple version of the DGG for part of the domain of Vi showed that the approach to generating didactic relations presented in this chapter is feasible, and results in as good curricula as human tutors use.

References

B.J. Brecht. *Determining the Focus of Instruction: Content Planning for Intelligent Tutoring Systems.* PhD thesis, University of Saskatchewan, Saskatoon, Canada, 1990.

J.A. Breuker, editor. *EUROHELP: Developing Intelligent Help Systems.* EC, Copenhagen, 1990.

W.J. Clancey. The epistemology of a rule based system–a framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

I.P. Goldstein. The genetic graph: a representation for the evolution of procedural knowledge. *International Journal of Man–Machine Studies*, 11:51–77, 1979.

A. Lesgold. Toward a theory of curriculum for use in designing intelligent instructional systems. In H. Mandl and A. Lesgols, editors, *Learning Issues for Intelligent Tutoring Systems*, pages 114–137. Springer-Verlag, New York, 1988.

G. McCalla. The Search for Adaptability, Flexibility, and Individualization: Approaches to Curriculum in Intelligent Tutoring Systems. Technical Report 90–6, University of Saskatchewan, Saskatoon, CAN, 1990.

W. Murray. Control for intelligent tutoring systems: A blackboard-based dynamic instructional planner. In D. Bierman, J. Breuker, and J. Sandberg, editors, *Proceedings of the 4th International Conference on AI & Education*, pages 150–168, Amsterdam, 1989. IOS.

J. van der Laan. Didactic goal generator. Master's thesis, Social Science Informatics, University of Amsterdam, 1989.

R.G.F. Winkels. *Explorations in Intelligent Tutoring and Help.* IOS Press, Amsterdam, 1992.

R.G.F. Winkels, J.A. Breuker, and J.A.C. Sandberg. Didactic Discourse in Intelligent Help Systems. In *Proceedings of the International conference on Intelligent Tutoring Systems, Montreal 1988*, pages 279–285, 1988.

R.G.F. Winkels, J.A. Breuker, and N. den Haan. Principles and Practice of Knowledge Representation in EUROHELP. In L. Birnbaum, editor, *The International Conference on the Learning Sciences*, pages 442–448, Charlottesville, Virginia, 1991. AACE.

Acknowledgements

The EUROHELP project was partially funded by the European Community as an ESPRIT project (contract P280). Partners were: ICL, University of Leeds (UK), Axion (Denmark), Courseware Europe, and the University of Amsterdam (the Netherlands). For the work described here we would like to thank our student Jan van der Laan for implementing a prototype of the DGG for Vi.