A Dialogue System to Teach Database Concepts

R. W. A. Hudson and J. A. Self
Department of Computer Studies, University of Lancaster, Lancaster LA1 4YN, UK

This paper describes a program written to investigate the derivation of teaching strategies in computer-assisted learning dialogue systems. The program endeavours to teach database concepts and is based upon a relational database management system. The program's design, however, is intended to be independent of the particular subject matter taught and is influenced by the recent emphasis on knowledge representation in artificial intelligence.

INTRODUCTION

Computer-assisted learning systems need both to understand the knowledge to be taught and to have some means of determining the sequence with which it should be dispensed. Relating the two should be a student model, that is, a representation of the knowledge attainment of a student user. The program we will describe derives its teaching strategy from such a student model.

The program aims to help a student to become proficient in the use of the commands necessary to operate a database system. The principles demonstrated, however, are domain independent and the system could be modified to teach other subjects.

Research into the design of computer-assisted learning systems has been submerged under the wave of microcomputer euphoria. The large-scale demonstration projects of the mid-1970s—NDPCAL,[1] PLATO[2] and TICCIT[3]—did not receive particularly favourable evaluations,[4–6] but these seem now to be considered irrelevant as the projects themselves are technologically obsolete. With the advent of microcomputers, there has been a reversion to techniques discarded two decades ago—simple drills, games, simulations and programmed learning.[7]

This project aims not to mimic the formal lecture or tutorial nor to provide undirected practice and experience but to provide a flexible environment in which (a) the student or the system can initiate dialogue on any area within the subject domain; (b) the student takes the major decisions on whether to be instructed or not and in which area he wants information; (c) the student can test his skills using examples provided by the system; (d) the student is instructed on concepts or skills that he desires or that the system decides that he needs.

In this paper we first describe related work on computer-assisted learning dialogue systems and then summarize the database system which forms the domain of expertise of our teaching program. The bulk of the paper gives details of the components of the teaching program itself, with an illustrative dialogue.

DIALOGUE SYSTEMS

Almost all the tutorial programs which are generally available were written in author languages, i.e. programming languages which at their simplest only provide routines for displaying text, comparing a short student response with a set of anticipated responses, and transferring control to a specified 'frame' of material. Since author languages are designed to be used by teachers without programming experience, it is only to be expected that most programs lack sophistication. While a skilful programmer can lessen the regimentation of the resultant dialogues by following convoluted paths through the material, it is usually the case that 'the teaching is primitive in accommodating individual differences since the decision rules are largely based only on the last response of the student' and 'there is little attempt to build up a representation of (the student's) knowledge and skills'.[8]

Disagreement with the underlying educational philosophy of such programs and an appreciation of their computational naivety led several researchers to try to develop learning environments in which the student had more control over what was discussed and in which more meaningful dialogues could be supported. Many of the problems which this task imposes lie within the realm of artificial intelligence. It would, however, be something of a digression to argue the case for artificial intelligence in computer-assisted learning (but see Refs. 9 and 10) and so we will merely give brief descriptions of a few projects.

The SCHOLAR system was the first attempt to build a 'mixed-initiative' dialogue system,[11] i.e. one in which both the student and the computer can ask questions. The subject domain was the geography of South America, knowledge of which was held in a static graph structure. This structure was not, however, used to generate a teaching strategy, topics for discussion being selected at random. As a result, dialogues with SCHOLAR lack the purposeful nature of good tutorials.

Most subsequent dialogue systems have gained purposivity by adopting game-playing or problem-solving environments. For example, WEST[12] attempts to monitor a student's attempt to play a simple board game in order to coach the student by interrupting, when appropriate, to give advice on the arithmetic and strategic skills involved. The program compares the student's moves with those that ought to be made (as determined by its own 'expert procedure') to determine the student's weaknesses. The student is modelled by a vector of numbers, each number representing his mastery of some skill.

With SOPHIE,[13] the student has to try to isolate the fault in an electronic circuit by asking for measurements

CCC–0010–4620/82/0025–0135 $02.50

at different points in the circuit. In addition to answering the student's questions, SOPHIE comments on the student's hypothesis, but does not provide assistance unless requested to do so. Like SOPHIE, GUIDON[14] is built around an 'expert system',[15] in this case, MYCIN[16] which acts as a consultant on infectious diseases. GUIDON monitors a student's diagnosis by comparing his questions with those that would be asked by MYCIN in the same situation.

Such systems emphasize the use of 'mixed-initiative', natural language dialogues in discovery learning situations but tend to ignore the primary teaching process. They assume that the student has already been taught the necessary material and merely needs practice in applying it. However, in most tutorials basic material may have to be offered, and our system attempts to do this within a problem-solving environment.

EDBMS

The artificial intelligence-based dialogue systems emphasize the fact, ignored by author language dialogue systems, that to sustain a conversation one must understand the topic of conversation. Thus, programs should themselves be able to solve problems posed to students. In our system the role of 'expert program' is played by EDBMS, an educational database management system based on the relational model of data storage and designed to provide a student with practical experience in database techniques.[17] (Commercially available database systems would not be appropriate since they could not be modified or used flexibly enough.)

EDBMS provides a query language to enable a student to manipulate, reorganize and retrieve data held in relational form using high-level algebraic operations (selection, projection, join, division, union, difference, intersection, cartesian product, etc.) defined in Ref. 18. This language is designed for students familiar with the function call notation of most programming languages. Hence, each command consists of a command-name followed by none or more parameters in brackets, e.g.

```
PROJECT(PARTS, PNAME, COLOUR).
```

Some commands, e.g. PROJECT, have a variable number of parameters. Each relation held in the database has a name (e.g. PARTS) and each relational or set operation returns as its result a new relation which may be retained in the database by assigning it to a relation name:

```
NEW := PROJECT(PARTS, PNAME, COLOUR).
```

If the computed relation does not need to be retained it may simply be printed:

```
PRINT(PROJECT(PARTS, PNAME, COLOUR)).
```

Commands may be nested, that is, wherever a relation is required as a parameter a command returning a relation may be used, e.g.

```
PRINT(PROJECT(SELECT(UNION(PARTS,
            SUPPLIERS), WEIGHT, >, 15), PNAME)).
```

The language also provides commands for sorting a relation on a specified column, for listing the names of relations in the database, and for creating and deleting relations by means of an interactive session with the user.

Thus EDBMS is a small-scale database system designed to enable a student to understand relational operations. EDBMS ignores many of the problems of practical systems, such as security and integrity. More germane for our purposes, EDBMS, as it stands, provides no direct teaching, by means of explanation or advice, during a student session.

QUADBASE

The dialogue tutor system, QUADBASE, is intended to provide both primary teaching material and a problem solving environment. Our main concern has been with the construction of a student model adequate for the generation of a teaching strategy. It may be helpful first to look at the illustrative dialogue given in Fig. 1.

QUADBASE consists of a number of modules which interact according to the diagram shown in Fig. 2, and we will now consider these modules in turn.

Student model

The student model is the heart of the system. It holds information on the student's hypothesized knowledge state and is constructed from the student's performance with the system and from assumptions made during the operation of the explanation package.

The student model is in the form of a single entry point flow graph, the nodes of which represent concepts to be learned and the arcs of which determine the prerequisite structure of the concepts. The particular graph used for the dialogue of Fig. 1 is shown in Fig. 3: the details, however, are not too important as the graph may be changed simply by changing a data file. In contrast to the 'vector student model' of most dialogue systems, the structure of our model enables a teaching strategy to be generated.

The student's knowledge state is represented by superimposing on the graph, for each node, a measure of the student's understanding of the concept represented by that node. These measures are held as numbers in the range 0 to 100 and are modified throughout the teaching session. Measures that fall below some threshold may initiate some explanatory dialogue from QUADBASE.

While the flowgraph reflects the structure of the knowledge to be taught, so that a sensible teaching sequence could be generated by, for example, a depth-first or breadth-first traversal, it does not describe a completely predetermined sequence as is characteristic of author language dialogue systems.

Input analyser

QUADBASE is always in one of two modes: 'command mode', in which it will be expecting the student to be inputting a database command, or 'tuition mode', in which the student is undergoing instruction. The student may change mode at any time. The input analyser, therefore, is prepared for either kind of response: an

(We join the dialogue after some tuition has been given and the student has been trying unsuccessfully to use PROJECT. The student's input is underlined.)

No comma in PROJECT.
You are clearly have trouble with this command.
Would you like some help?

Yes

I will first of all give you the definition of project and then an example.

Project is an operator for constructing a 'vertical' subset of a relation—i.e. a subset obtained by selecting specified attributes and eliminating others (and also eliminating duplicate tuples within the attributes selected).

(Two lengthy examples follow.)

The commands for the above are:—
PROJECT(COUNTRY, CAPITAL) and
PROJECT(COUNTRY, CAPITAL, NAME).

Would you like to try an example?

Yes

Below are the relations held in the system
SUPPLIERS
PARTS
SP
Which relation would you like to project?

SUPPLIERS

OK—now project SUPPLIERS over CITY.

PROJECT(SUPPLIERS, CITY)

Well done—you have got it.
Your result is the relation:—

```
CITY
________
ATHENS
LONDON
PARIS
________
```

From your performance I can see that you need more practice using PROJECT and SELECT. I will now give you an example in which both these commands will have to be used.

Please would you retrieve and print the partnames of parts supplied in London using a single command.

PROJECT(SELECT(PARTS, CITY, EQ, LONDON), PNAME)

Great stuff—middle stump!
The result of your command is:—

```
PNAME
________
SCREW
NUT
COG
________
```

Now type in a command.

Explain intersect.

I will explain intersect.
We define intersect as follows:—
The intersection of two union-compatible relations A and B is the set of all tuples T belonging to both A and B.

Shall I continue with the explanation?

Explain union first.

I will explain union.
We define union in the following way:—
The union of two union-compatible relations A and B is the set of all tuples T belonging to either A or B (or both).

Shall I continue with the explanation?

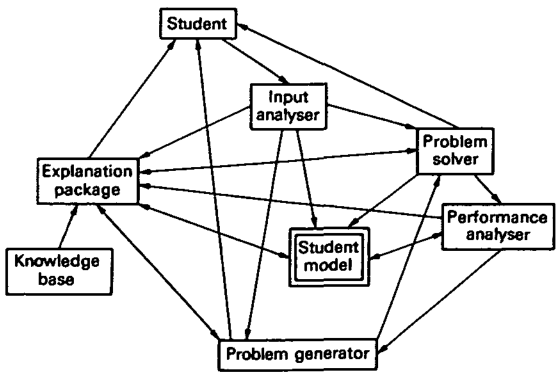
No

You interrupted my explanation of intersect.
Would you now like that to continue?

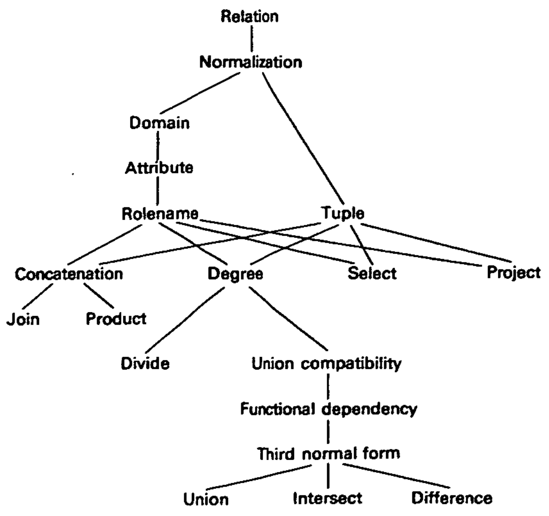
Yes

(and so on)

Figure 1. A dialogue with QUADBASE.



16 **Figure 2.** QUADBASE modules.



18 **Figure 3.** Structure of the student model.

input statement is first parsed to see if it corresponds to a valid command format and, if this is not successful, is assumed to be a natural language statement, e.g. 'Please explain union first.' Keyword analysis is used on natural language sentences, and this is adequate for our purposes though, of course, limited compared to artificial intelligence parsing techniques.

Performance analyser

Every successful and unsuccessful use of a database command alters the student model and if QUADBASE decides that the student is finding difficulty with some concept, it will interrupt and offer assistance. Coaching will only be given if the student agrees. During the coaching session the student is given explanations and practice examples on the troublesome concept.

The modification of the student model takes into account the relationship between the nodes of the graph. For example, an error with the PROJECT command reduces the measure associated with the PROJECT node and also, to a lesser extent, the prerequisite concepts TUPLE, ROLENAME, etc. Consequently, a subsequent error with, for example, SELECT may lead QUADBASE to the conclusion that the source of the difficulty is really a common prerequisite concept, e.g. TUPLE. Similarly, QUADBASE increases the measures associated with the prerequisite concepts of a successfully used command.

Explanation package

When the student requests an explanation or agrees to one suggested by QUADBASE, the system traverses the graph from the entry point to the node representing the concept to be explained. If the measure associated with a node is below the threshold (or if the student has particularly requested an explanation) then QUADBASE gives a definition of the concept, followed by optional examples and practice questions, in a manner similar to that of the TICCIT system.[3] The teaching sequence, once under way, can be altered by student interrupts, for example, to ask for a reminder of the definition of some concept or to return to the command mode.

Knowledge base

The text for the definitions and examples given by the explanation package is stored in a random file which can be modified independently of QUADBASE. Each entry in this knowledge base is identified by a number and the numbers associated with the entries for a particular concept are stored at the corresponding node in the student model. These numbers, like the rest of the student model, are stored in a data file. In this way the size of QUADBASE is reduced and the system does not need recompiling to change text output.

Problem generator

The practice problems are not stored in a text file but are generated as and when needed by the explanation package. First, problems are generated which involve only the operation under discussion and then ones which involve all the operations which the student needs to practise. The problem generator uses a set of problem formats.[19] For example, for a simple projection request, the format is PROJECT(*rel*, *attr*), where *rel* is selected from the store of relations by the student and *attr* is selected by a random number generator.

Problem solver

A dialogue system which cannot answer questions, either posed by the student or by itself (during a coaching session) is of little use. Consequently, as explained above, the artificial intelligence-based dialogue systems are built upon 'expert problem-solving programs'. With QUADBASE, the expert program is EDBMS, described above.

Operation of QUADBASE

When the student enters QUADBASE he is asked whether he is familiar with the database concepts and manipulations. If he considers that he is then he may proceed to use EDBMS as he wishes (or to complete accompanying worksheets). If the student is not confident enough to use the system immediately he is asked which concepts he would like explained. He may, if he wishes, ask for a full tutorial.

As the student works through problems and receives explanations, QUADBASE updates the student model to reflect the student's current state of knowledge. As long as the student is considered to be making satisfactory progress QUADBASE does not intervene. If, however, QUADBASE feels that the student is having difficulty with a particular concept an explanation will be offered. In addition, the student may at any time request an explanation or make a general plea for 'help'. The explanation given by QUADBASE is adapted to the perceived student needs by making it depend upon the measure associated with the corresponding node. If, for example, the student requests tuition on a concept for which the measure is high, QUADBASE will give only a short explanation to act as a reminder and then ask the student if he requires more information.

DISCUSSION

QUADBASE is a demonstration computer-assisted learning system which provides basic teaching material, a strategy for dispensing it, and a problem-solving environment. It aims to teach a student relational database concepts but its design is intended to be appropriate for any subject domain in which the concepts can be hierarchically organized. QUADBASE incorporates a student model from which a teaching strategy is generated suited to the individual needs of the student.

The student model is the key component of a dialogue system for it provides the means by which the knowledge to be taught can be presented taking into account the student's strengths and weaknesses. The student model must have a structure which reflects the knowledge structure of the subject domain. In QUADBASE, the student model is a flow graph, nodes represent concepts and arcs the relationships between them. The model assumes that teaching material has to be sequenced for the student to be able to learn it effectively.

Recent dialogue systems have evolved towards game-playing environments where the student is encouraged to discover knowledge and exercise skills. While this accords with the Piagetian school of educational thinking, it is an approach which separates the presentation of teaching material from the provision of game-playing environments. It is our view that primary teaching material needs to be available for presentation to a student during a tutorial session.

With QUADBASE the emphasis has been on the role of the student model and its relationship with the other components. Some of these components (e.g. the input analyser and problem generator) are merely adequate to complete a working system but they can be improved independently of other components. Some components were left minimally developed in order to be able to continue to run QUADBASE on the equipment available. QUADBASE is written in Pascal and runs on a 64K Eclipse with only 32K available to a user program.

REFERENCES

1. R. Hooper, *The National Development Programme in Computer Assisted Learning: Final Report of the Director*, Council for Educational Technology, London (1977).
2. D. L. Bitzer, The wide world of computer-based education, in *Advances in Computers 15*, ed. by M. Rubinoff and M. Yovits, Academic Press, New York (1976).
3. M. D. Merrill, Learner control in computer based learning. *Computers and Education* **4**, 77–95 (1980).
4. J. Fielden and P. Pearson, *The Cost of Learning with Computers*, Council for Educational Technology, London (1978).
5. R. T. Murphy and L. R. Appel, *Evaluation of the PLATO IV computer-based education system in the community college*, NSF Contract No. NSF-C731. Educational Testing Service, Princeton, New Jersey (1977).
6. D. L. Alderman, Evaluation of the TICCIT computer-assisted instructional system in the community college. *Special Interest Group on Computer Uses in Education (SIGCUE) Bulletin* **13** (No. 3), 5–17 (1979).
7. J. A. M. Howe and B. du Boulay, Microprocessor assisted learning: turning the clock back. *Programmed Learning and Educational Technology* **16**, 240–246 (1979).
8. J. R. Hartley, An appraisal of computer-assisted learning in the United Kingdom. *Programmed Learning and Educational Technology* **15**, 136–151 (1978).
9. G. P. Kearsley, The relevance of AI research to CAI. *Journal of Educational Technology Systems* **6**, 229–250 (1978).
10. A. Gable and C. V. Page, The use of artificial intelligence techniques in computer-assisted instruction: an overview. *International Journal of Man-Machine Studies* **12**, 259–282 (1980).
11. J. R. Carbonell, AI in CAI: an artificial intelligence approach to computer-assisted instruction. *Institute of Electrical and Electronic Engineers Transactions on Man-Machine Systems* **11**, 190–202 (1970).
12. R. R. Burton and J. S. Brown, An investigation of computer coaching for informal learning activities. *International Journal of Man-Machine Studies* **11**, 5–24 (1979).
13. J. S. Brown, R. R. Burton and A. G. Bell, SOPHIE: a step toward creating a reactive learning environment. *International Journal of Man-Machine Studies* **7**, 675–696 (1975).
14. W. J. Clancey, Tutoring rules for guiding a case method dialogue. *International Journal of Man-Machine Studies* **11**, 25–49 (1979).
15. D. Michie, Expert systems. *Computer Journal* **23**, 369–376 (1980).
16. E. H. Shortliffe, *Computer-based Medical Consultations: MYCIN*, Elsevier, New York (1976).
17. R. W. A. Hudson, The development of an educational database management system using the relational model of data storage. MA Thesis, University of Lancaster (1978).
18. C. J. Date, *An Introduction to Database Systems*, Addison-Wesley, Reading, Massachusetts (1977).
19. B. G. Palmer and A. E. Oldehoeft, The design of an instructional system based on problem-generators. *International Journal of Man-Machine Studies* **7**, 249–271 (1975).

Received May 1981