

Logically Optimal Curriculum Sequences for Adaptive Hypermedia Systems

Roland Hübscher

107 Dunstan Hall, Department of Computer Science and Software Engineering
Auburn University, Auburn, AL 36849-5347, U.S.A.
roland@eng.auburn.edu

Abstract. Curriculum sequencing is an important technique used in many adaptive hypermedia systems. When following one of the possible page sequences, visiting some pages may become redundant, because its content has been covered already by another page. Using disjunctive and conjunctive prerequisites instead of partial orders to describe the many possible sequences, *logical redundancy* between pages can be computed on the fly without burdening the teaching model with that task [1]. Although the general case of finding all redundant pages is NP-Complete [2] and thus, intractable unless $P = NP$, a large subset can be located efficiently in realtime. The advantage of separating out logical redundancy, the advantage of using conjunctive and disjunctive prerequisites, and the algorithms to find redundant pages are discussed. An interesting characteristic of the presented approach is that it can be used together with a wide variety of user and teaching models.

1 Introduction

The goal of curriculum sequencing is to provide a student with an optimal path through the material to be learned and tasks to be executed, e.g., practicing a skill or answering questions [3]. Since every student has different prior knowledge, preferences, and often different learning goals, providing individualized curriculum sequences for each student using adaptive hypermedia is a promising approach.

But what is an optimal path? In an educational adaptive hypermedia system, an optimal path maximizes a combination of the student's understanding of the material and the efficiency of learning the material. However, it makes more sense to attempt to suggest a few good paths instead of an optimal one, because the latter does not exist. It should then be left to the learner to select whichever one of the, possibly many, paths that are expected to lead to effective learning for this student. Otherwise, the hypermedia loses its big advantage of being freely explorable, providing at least somewhat of an authentic learning experience to the student [4]. However, providing a hypermedia system with a link structure that allows the user to choose between good paths could result in redundant visits to certain pages and thus, in a suboptimal curriculum sequence. This situation cannot be avoided if a wide variety of learners need to be supported. Different examples, different kinds of representations, different modalities, etc. can be used for explaining the same concepts in slightly different ways which means that not everybody ought to have to look at everything everybody else does.

For instance, assume that in one sequence concept A needs to be studied before concept X . In another sequence, we want the student, again for pedagogical reasons, to read about concept B before X . Looking at the bigger picture, we realize that either A or B ought to be studied before X . Thus, once the learner understands A , he or she can ignore B , as it has become redundant with respect to the goal of understanding X . However, as we will see later, the general case is not as simple as this trivial example may imply.

A guiding principle for our approach is that the space the user may explore is maximized, i.e., we want to constrain the user's choice as little as possible. However, navigation support should scaffold [5] the user to take the best paths with respect to the user's interest and current understanding of the domain, the domain itself, and the underlying pedagogical framework. In short, we are interested in effectively supporting scaffolded exploration.

In this paper, we study how we can find logically optimal paths in hypermedia systems that adaptively provide many good curriculum sequences. Curriculum sequencing is used as a technical term, although most of the time this sequencing, or adaptive ordering [6], is applied to smaller parts, like courses. Logically

optimal paths do not contain pages that are redundant based on logical, and not on pedagogical, grounds. Sometimes, one explicitly wants to be redundant to some degree, e.g., an issue should be covered by at least two examples. Such situations can also be described by the presented formalism. We show that generally all redundancy cannot be detected in realtime, however, a large subset can be found efficiently by the presented algorithms.

2 Sequencing with Prerequisites

The material to be learned and tasks to be executed, from now on simply called “units,” need to be ordered for each learner dependent on the teaching method employed and the model the system has of the student. The resulting structure describing all the sequences can be represented as a directed acyclic and-or graph where the vertices are the units and the edges are prerequisites between the units. This graph is called a *prerequisite graph*. Note that we use the term “prerequisite” not in a strictly pedagogical sense, i.e., we do not claim that organizing a course simply using prerequisites is appropriate. We simply call the temporal constraints between the units prerequisites. However, we claim that most pedagogically interesting organizations of the units can formally be reduced to a prerequisite graph.

One might argue that a simple partial-order representation of the units is just as expressive. Although this is true in principle, the size of the partial-order graph would be exponentially larger than the prerequisite graph as will become clear below.

A prerequisite is either a conjunctive or a disjunctive. A conjunctive prerequisite $u_1 \wedge u_2 \wedge \dots \wedge u_n \Rightarrow u$ asserts that all of the units u_1, u_2, \dots, u_n need to be “visited” by the student before unit u may be visited. Instead of “visited” we could also say “learned,” “seen,” “understood,” etc., depending on the specific design of the adaptive hypermedia system and the teaching method adopted. This is independent of our concept of logically optimal paths. Similarly, a disjunctive prerequisite $u_1 \vee u_2 \vee \dots \vee u_n \Rightarrow u$ asserts that at least one of the units u_1, u_2, \dots, u_n needs to be visited by the student before unit u .

We are currently designing a language that allows course designers to describe teaching methods that can be translated into the prerequisite formalism introduced here. The following few simple examples will provide an idea of where this language is going.

Assume that x and y are some knowledge units, for instance, a concept or a topic consisting of several knowledge units. Sometimes, for instance in Problem-Based Learning [7, 8], it is preferable to have the students find out that they have a need to learn certain prerequisites. So, one would like to talk first about y and only then about x . The rule

if x is required to understand y **then** $y \Rightarrow x$

says that prerequisites (in the pedagogical, not formal, sense) should be visited *after* what they are prerequisite for. This example makes it explicit that we use the prerequisites in the formal sense as a way to order the units. However, how prerequisites in the pedagogical sense are used depends on the teaching method used.

The following rule results in a top-down organization of part-whole hierarchies since it says that one should learn about the whole before learning about its parts.

if y is part of x **then** $x \Rightarrow y$

And finally, the last rule says that related issues should be visited in parallel to the main topic: If x is strongly related to y , then x is visited while visiting y . This rule only makes sense if x and y are more complex hypermedia structures than a single page. Then we can refer to the start and the beginning units of those structures and use our prerequisites again. Virtual units introduced below can be used as special start and end units if necessary.

if x is strongly related to y **then** $\text{start}(y) \Rightarrow \text{start}(x)$
if x is strongly related to y **then** $\text{end}(x) \Rightarrow \text{end}(y)$

In order to make the simple formalism of conjunctive and disjunctive prerequisites a bit more expressive, we introduce the notion of *virtual units*. A virtual unit does not exist for the student, however it is an element

in the prerequisite graph. This allows us to have prerequisites with nested conjunctions and disjunctions like the following one requiring the learner to visit exactly two of the three units u_1 , u_2 , u_3 before visiting u :

$$(u_1 \wedge u_2) \vee (u_1 \wedge u_3) \vee (u_2 \wedge u_3) \Rightarrow u$$

This constraint can be implemented with the following four prerequisites introducing three virtual units v_1 , v_2 , and v_3 :

$$\begin{aligned} u_1 \wedge u_2 &\Rightarrow v_1 \\ u_1 \wedge u_3 &\Rightarrow v_2 \\ u_2 \wedge u_3 &\Rightarrow v_3 \\ v_1 \vee v_2 \vee v_3 &\Rightarrow u \end{aligned}$$

At each moment, a set of units is called the *goal*. The goal is what the user tries to learn. For instance, for class notes the goal is quite constant for a semester, namely covering a certain minimal set of topics. For a manual, though, the goal may change whenever the user is accessing the manual.

We do not plan to add negations to the formalism because that would result in non-monotonic behavior, i.e., units that used to be accessible to the user are suddenly not accessible anymore. This would violate our guiding principle that the space the user may explore is maximized, i.e., that the user's choices are constrained as little as possible. Furthermore, we believe that this would cause even more potential usability problems than the appearance of new links as is common in adaptive hypermedia [9]. However, we do not have any empirical evidence for the negative effect of non-monotonic adaptive hypermedia as of yet.

2.1 Disjunctions Add Complexity

As will be shown below, disjunctive prerequisites come in handy in some situations. However, adding disjunctions causes problems, because they add degrees of freedom. If both, u_1 and u_2 are required for u , then the user has no choice, and adding more conditions u_3, \dots, u_k does not add any new choices. There is one way to get to u : Visit all units u_1, \dots, u_k in the prerequisite. However, if we have a disjunctive constraint $u_1 \vee u_2 \vee \dots \vee u_n \Rightarrow u$ there are $2^n - 1$ subsets of u_1, u_2, \dots, u_n that can be visited before visiting u . So disjunctions add an exponential amount of choice. Thus, if the representation with disjunctive and conjunctive prerequisites would be replaced by a partial-order representation as in other adaptive hypermedia systems[3, 10], the size of the partial-order graph would increase exponentially.

That does not seem to be a serious problem, though, because as soon as a disjunct u_i in the above prerequisite gets visited, all the other $u_j, j \neq i$, are redundant because u may be visited now. However, some of the units u_j may be involved in some other prerequisites and have to be visited for other reasons than just enabling u to be visited. And what's worse, this is not a local phenomenon: These other units may be far away in the graph. The algorithm described later will solve this problem.

2.2 Using Prerequisites to Order Units

We claim that the use of conjunctive and disjunctive prerequisites with virtual units is powerful enough to describe a large class of dependencies in adaptive hypermedia systems. An interesting characteristic of this approach is that it is relatively independent of the underlying teaching methods and user model. Thus, it can be used with a wide variety of user models and teaching methods [1]. All the model needs to output is the set of units that can be considered visited (or learned, seen, etc.).

Providing just one of many effective sequences to the user would be the wrong approach, since that would get rid of most of the benefits of having *hypermedia* in the first place. Providing conjunctive prerequisites is very close to providing one sequence only. Furthermore, separating the pedagogical and the logical characteristics of the dependencies makes it more clear what the semantics of the user model and teaching method are. Finally, adding disjunctions allows one to nicely describe certain dependencies that are quite important in an educational setting.

Disjunctive prerequisites are useful if there is redundant material in the hypermedia system supporting diverse set of learners. For instance, the same concept can and often should be described in different ways,

using different representations, different modes of presentation, etc., [11, 12]. However, the student should have the freedom to select the most appropriate one (from the set of “good” ones). This relationship can then be described with a disjunction. Of course, one might argue that the adaptive system should provide the appropriate representation and mode. However, that is pushing adaptiveness too far. Sometimes a system can safely make a good choice, but taking all the decisions away from the learner is a misguided approach.

Another example use of disjunctions is if one wants to make sure that a student sees the use of a certain concept or feature, say of a programming language, in at least two examples. This constraint can be easily expressed as a disjunction of conjunctions. Letting e_i be the examples and t be the topic to cover, we can express this constraint as follows.

$$(e_1 \wedge e_2) \vee (e_1 \wedge e_3) \vee \cdots \vee (e_1 \wedge e_n) \vee \cdots \vee (e_{n-1} \wedge e_n) \Rightarrow t$$

A similar example where disjunctions come in handy is where the instructor has several illustrating examples to explain an issue in the lecture notes. He wants to select just one example and make sure he is not going over other, redundant examples. Conjunctions alone cannot express such a relationship easily. Again, let e_i be the examples and t be the topic to cover. Then, we can express the instructor’s approach in subtly different ways. If the example needs to be covered *before* the topic then this is expressed as follows.

$$e_1 \vee e_2 \vee \cdots \vee e_n \Rightarrow t$$

If the instructor want to make sure at least one example is used to illustrate the topic *at any time*, then we introduce two virtual units v_1 and v_2 and describe the organization as follows.

$$\begin{array}{l} e_1 \vee e_2 \vee \cdots \vee e_n \Rightarrow v_1 \\ t \wedge v_1 \qquad \qquad \Rightarrow v_2 \end{array}$$

Of course, one could implement this prerequisite in some ad-hoc fashion, but that tends to be a bad idea in the long run and complexity will show its ugly face in some form sooner or later.

3 Redundant Units

As mentioned earlier, without disjunctions, there is no need to deal with redundant units, because, sooner or later, all units will be necessary. However, as soon as disjunctions are added, things are getting more complicated. Let’s look at a few small examples.

Example 1: There is one disjunctive prerequisite, such that at least one of A or B must be visited before X is visited.

$$A \vee B \Rightarrow X$$

Assume we want to visit X . As soon as A is visited, X may be visited next. Since X is now enabled, i.e., can be visited, there is no need to visit B , since it does not enable something of any value. We say that visiting A makes B *redundant* (via X). Similarly, visiting B first makes A redundant.

Example 2: We extend the first example by adding B as prerequisite for a new unit Y and our new goal is Z .

$$\begin{array}{l} A \vee B \Rightarrow X \\ B \qquad \Rightarrow Y \\ X \wedge Y \Rightarrow Z \end{array}$$

In this case, visiting A does not make B redundant, because B is still needed to enable Y to be visited. It also becomes clear that visiting A is actually inefficient. B needs to be visited independently of whether A is visited or not. Thus, A is redundant before any of the units has been visited, because B is *required* and, sooner or later, will be visited.

Example 3: A slightly different situation arises in the final example.

$$\begin{aligned} A \wedge B &\Rightarrow X \\ B \wedge C &\Rightarrow Y \\ X \vee Y &\Rightarrow Z \end{aligned}$$

Here, we have the case where B is required, although there is only a disjunctive prerequisite for the goal Z .

These examples show that some units may become redundant due to other units being visited and some units are redundant simply due to the structure of the prerequisite graph and the goal unit. They also show that these decisions cannot be made locally.

3.1 Formalizing Redundancy

Before redundancy of units can be formalized, a few terms need to be defined.

unit A unit is the smallest description of some concept, method, etc. Without loss of generality for our discussion, it can be thought of as a web page.

goal A goal is a set of units that the user wants to visit, but might not be able to right now, because it is not enabled yet.

virtual unit A virtual unit is a unit that exists in the prerequisite graph in order to express the necessary prerequisites (e.g., a disjunction of conjunctions), however, it is invisible to the user, i.e., it does not represent an actual page.

visited unit A unit is visited if the concept the unit describes is assumed to be known by the user. Only enabled units may be visited. When and why a concept is assumed to be known depends on the user model. For simplicity reasons, without sacrificing any generality, we assume that the user visiting a page means he or she has actually learned its content.

enabled unit A unit is enabled if all of its prerequisites are satisfied. If the prerequisite is a conjunction, then all of its prerequisite units need to be visited. If it is a disjunctive prerequisite, then at least one of the units needs to be visited.

redundant unit A unit is redundant if it has not been visited, and visiting it does not enable any unit at any time in the future. A more formal description follows below.

In a simple web-based adaptive hypermedia system (WHAM, see section 3.4) where we employed the described adaptive algorithm, we used three types of hyperlinks: recommended links displayed in bold blue; redundant links displayed in purple; and forbidden links which were plain text, i.e., not links. Recommended links pointed to enabled, non-redundant units. No links (or forbidden links) were made to units that were not enabled. Redundant links pointed to enabled, redundant units.

Redundant units can then be defined as follows. Let $P = \{u_1, u_2, \dots, u_k\}$ be the set of units representing the prerequisite units of $P \Rightarrow u$. A *future-enabled* unit is a unit that is enabled assuming that all required units have been visited in addition to those units the user actually has visited. Then we can recursively define a redundant unit u as follows, where “ \rightarrow ” stands for the material implication:

$$\text{redundant}(u) \equiv \forall P'. (u \in P') \wedge (P' \Rightarrow u') \rightarrow \text{redundant}(u') \vee \text{futureEnabled}(u')$$

That is, u is redundant if all units u' for which u is a prerequisite are either redundant, or enabled or will be enabled for sure in the future. An efficient algorithm to find most, but not necessarily all, of these redundant units will be presented below.

3.2 Complexity of Finding Redundant Units

Finding enabled units is easy, and therefore, finding disabled units is easy. The real difficulty in a computational sense lies in finding the redundant units as we will see in a moment.

The input to the problem of finding redundant units are the prerequisite graph $G = (V, E)$, the goal unit g , and the set of visited units U . We assume that there is only one goal unit. If there are more than one, say

u_1, \dots, u_n then we simply introduce a new, virtual unit g and a conjunctive prerequisite $u_1 \wedge \dots \wedge u_n \Rightarrow g$. The graph's vertices V is the set of units in the graph, and the directed edges E are between the prerequisites of a unit and the unit.

Unfortunately, finding out whether a unit is required is NP-complete as can be seen as follows. Let $v(u)$ stand for “unit u has been visited.” Let F be a set of formulas in propositional logic as follows. For each conjunctive prerequisite $u_1 \wedge \dots \wedge u_k \Rightarrow u$ add $v(u) \rightarrow v(u_1) \wedge \dots \wedge v(u_k)$ to F because u can only have been visited if all u_1, \dots, u_k have been visited. Similarly, for each disjunctive prerequisite $u_1 \vee \dots \vee u_k \Rightarrow u$ add $v(u) \rightarrow v(u_1) \vee \dots \vee v(u_k)$ to F . Then, to show that u is required one has to show that $F \cup \{g\} \vdash u$. This is equivalent to the satisfiability problem which is known to be NP-complete [2]. Thus, we cannot expect to find an efficient algorithm that finds all required units.

Although this is bad news, all hope is not lost. If the graph and the goal are constant, then finding the required units can be done off-line. For instance, for class notes the goal can be assumed to be constant for the whole semester, namely cover a certain minimal set of topics. However, for a manual where the goal changes very often, solving an NP-complete problem in realtime is not feasible. Although finding required units is hard, finding only those redundant units as a function of enabled, but not future-enabled, units can be done efficiently.

3.3 Finding Most Redundant Units

Now that it is clear that we cannot find all redundant units on the fly, we look at a simpler version of the problem that will result in an algorithm that is correct but not complete, i.e., it will find only redundant units but possibly not all. However, the same algorithm can be used to find all redundant units if all required units are known.

Finding Redundancy-Propagating Paths The prerequisite graph G is initialized independently of the user's actions and goals, i.e., the initialization is only dependent on the prerequisites. Once the graph is initialized, redundant units can be efficiently computed while the user is visiting pages.

A unit u was earlier defined to be redundant if $\forall P'. u \in P' \wedge P' \Rightarrow u' \rightarrow \text{redundant}(u') \vee \text{futureEnabled}(u')$. Note that “futureEnabled” may have to be replaced with “enabled” depending on whether the required units have been computed or not. Checking each enabled unit recursively to find out whether it is redundant or not would be prohibitively expensive.

Therefore, we devised an algorithm that allows redundancy to be propagated from visited towards non-visited units, which is computationally much cheaper. This requires the system to find the so-called *redundancy-propagating paths* in the prerequisite graph $G = (V, E)$ as follows. (We say that a vertex “represents a conjunct/disjunct” if it is part of a conjunctive/disjunctive prerequisite.)

```

for each vertex  $v$  in reverse topological order do
  // i.e., ordered such that if there is a path from  $v'$  to  $v''$ ,
  // then  $v''$  must appear before  $v'$  in the topological order
   $c(v) \leftarrow |\text{tokens}(v)|$ 
  for each  $v_i$ , where  $(v_i, v) \in E$  do
     $\text{tokens}(v_i) \leftarrow \text{tokens}(v_i) \cup \text{tokens}(v)$ 
    if  $(v_i, v)$  represents a conjunct then
       $t \leftarrow$  new unique token
       $\text{tokens}(v_i) \leftarrow \text{tokens}(v_i) \cup \{t\}$ 
    end if
  end for each
  // the set  $\text{tokens}(v)$  can now safely be deleted
end for each

```

The algorithm generates a unique token for each conjunct in a prerequisite and propagates it away from the goal. Each vertex v then counts how many different tokens arrive which is the number, denoted as $c(v)$, of different conjuncts it is involved in. Then, an edge (v_i, v_j) is redundancy propagating (rp) from v_j to v_i , if the following condition holds:

$$\text{rp}(v_i, v_j) = \begin{cases} \mathbf{true} & \text{if } (v_i, v_j) \text{ represents a conjunct } \wedge c(v_i) = c(v_j) + 1 \\ \mathbf{true} & \text{if } (v_i, v_j) \text{ represents a disjunct } \wedge c(v_i) = c(v_j) \\ \mathbf{false} & \text{otherwise} \end{cases}$$

The time complexity of this algorithm is $O(|V|^2)$ which is fast enough since it has to be run only once for a given set of prerequisites and is independent of the goals. Note that it is also independent of knowing the required units.

Propagating Redundancy Once the user starts using the hypermedia system, all new redundant units need to be found each time a unit is visited. Thus, this algorithm needs to be very efficient to be useful.

We define a procedure *visit*(v) that is called for a vertex when it is visited. Of course, only enabled vertices can be visited. Procedure *visit*(v) checks whether visiting v resulted in another unit becoming enabled. If so, procedure *propagate* defined below will find the redundant units.

```

procedure visit( $v$ ):
  visited( $v$ )  $\leftarrow$  true
  for each  $v_i$ , where  $(v, v_i) \in E$  do
    // one step forward
    if enabled( $v_i$ ) then
      // start from  $v_i$  and propagate redundancy backwards
      propagate( $v_i$ )
    end if
  end for each
end procedure

```

Procedure *propagate* finds redundant units by searching backwards along the redundancy-propagating paths only. Note that we only have to call *propagate* if visiting v caused v_i to become enabled, but not if it was already enabled before v was visited. We omit this code here for simplicity.

```

procedure propagate( $v$ ):
  if not redundant( $v$ ) and not visited( $v$ ) then
    for each  $v_i$ , where  $(v_i, v) \in V$  do
      if rp( $v_i, v$ ) then
        // we only propagate along the redundancy-propagating paths
        propagate( $v_i$ )
        redundant( $v_i$ )  $\leftarrow$  true
      end if
    end for each
  end if
end procedure

```

Worst time complexity is $O(|V|)$. However, this is irrelevant for any practical system. Since each vertex can become redundant only once and the user will need to visit $O(|V|)$ vertices, the expected time complexity is roughly constant as informal testing has shown.

3.4 Implementation for the World Wide Web

We have implemented these algorithms in WHAM (Web-based Hypermedia for Adaptive Manuals) using a simple user model, that stores which pages the user believes to understand and which not. Initially, we want to use this prototype to study the following issues: does the distinction between recommended, redundant, and forbidden links support scaffolded exploration effectively; is the continuously changing interface (links may come and go) a source of serious usability problems or not; are very simple user models good enough for many situations, or do we need indeed advanced cognitive models to provide useful navigation support.

In WHAM, the above algorithms are implemented using Java applets accessing a relational database storing the prerequisite graph including the count c for each node used to compute whether a link is redundancy propagating. The algorithms needed to be modified to work for large web sites. Computing the redundancy-propagating paths requires all nodes to be visited in topological order. Since there may be very many nodes to be visited when traversing the prerequisite graph, an iterative version of topological sort is used. The **visit** and the **propagate** functions are currently implemented recursively since they are not expected to traverse many edges of the prerequisite graph. All the data structures are implemented as tables in a relational data base (we use MySQL as relational database together with Apache as web server). The transformation of above rather simple looking algorithms into an efficient iterative, database-oriented version was not as straight forward as we expected.

The result is an efficient and scalable set of Java applets that can be used in various ways by adaptive hypermedia systems that implement different kinds of pedagogical models and different kinds of user models.

4 Conclusions

A large class of curriculum sequences can be described with conjunctive and disjunctive prerequisites. Using disjunctions implies that some of the content is redundant. By separating out logical redundancy from the pedagogical rules, we can simplify those rules and focus on the pedagogical issues more. However, using disjunctions comes, as so often, with a price to pay. Finding redundant units, or pages, is not easy, and to find all redundant units may be prohibitively expensive.

We have presented an algorithm that initializes the prerequisite graph, so that all redundant units that are only dependent on visited, but not on required, units can be found efficiently. If there is enough time to compute all the required units, for instance, if the goal is not going to change while the user is using the system, then the algorithm is complete, i.e., it will find all redundant units.

References

1. Paul De Bra, Geert-Jan Houben, and Hongjing Wu. Aham: a dexter-based reference model for adaptive hypermedia. In *Proceedings of the 10th ACM Conference on Hypertext and Hypermedia*, pages 147–156, 1999.
2. Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, NY, 1979.
3. Peter Brusilovsky. Adaptive educational systems on the world-wide-web: A review of available technologies. In *4th International Conference in Intelligent Tutoring Systems*, San Antonio, TX, 1998.
4. Joseph Petraglia. *Reality by Design: The Rhetoric and Technology of Authenticity in Education*. Lawrence Erlbaum Associates, Mahwah, NJ, 1998.
5. Roland Hübscher, Sadhana Puntambekar, and Mark Guzdial. A scaffolded learning environment supporting learning and design activities. In *AERA*, Chicago, 1997.
6. Peter Brusilovsky. *Methods and Techniques of Adaptive Hypermedia*, volume 6, pages 87–129. Kluwer Academic Publishers, 1996.
7. Howard S. Barrows. *How to Design a Problem Based Curriculum for the Preclinical Years*. Springer Verlag, New York, NY, 1985.
8. Roland Hübscher, Cindy E. Hmelo, N. Hari Narayanan, Mark Guzdial, and Janet L. Kolodner. McBAGEL: A shared and structured electronic workspace for problem-based learning. In *Second International Conference on the Learning Sciences*, Evanston, IL, 1996.
9. Kristina Höök. Evaluating the utility and usability of an adaptive hypermedia system. In *Proceedings of 1997 International Conference on Intelligent User Interfaces*, Orlando, FL, 1997. ACM.
10. Nicola Henze and Wolfgang Nejdl. Adaptivity in the kbs hyperbook system. In *Second Workshop on Adaptive Systems and User Modeling on the World Wide Web*, 1999.
11. Juan E. Gilbert and C. Y. Han. Adapting instruction in search of ‘a significant difference’. *Journal of Network and Computer Applications*, 22, 1999.
12. Rand J. Spiro, Paul J. Feltovich, Michael J. Jacobson, and Richard L. Coulson. Cognitive flexibility, constructivism, and hypertext: Random access instruction for advanced knowledge acquisition in ill-structured domains. *Educational Technology*, May 1991:24–33, 1991.