

# Finding Redundant Paths in Hypermedia

Roland Hübscher  
Computer Science & Engineering  
Auburn University  
Auburn, AL 36849-5347  
1 334 844 6343  
roland@eng.auburn.edu

## ABSTRACT

Web page prerequisites can be used to constrain how a user can explore a web site. This can be used in a number of ways, but plays an especially important role in educational sites. If disjunctive and conjunctive constraints are used to describe the preferences, some pages may become redundant given the user's previous path and the user may not want to visit them. The redundancy of a page is not a local property and may depend on many preferences. This paper describes an efficient algorithm that finds these redundant pages and discusses some applications of the approach.

## Keywords

Adaptive hypermedia, algorithms, prerequisites.

## 1. INTRODUCTION

Most web sites can be freely explored along any path the user chooses. However, in many educational applications or in a user manual the user ought to be guided along certain paths. If this is done in web site with static navigation-support, the amount of possible exploration needs to be reduced to a minimum [5] because otherwise the number of required pages to support all the possible paths would grow exponentially. Adaptive hypermedia systems can solve this problem by describing prerequisites between pages or concepts described on one or more pages and making pages only accessible to the user after if he has fulfilled the necessary prerequisites, e.g., by visiting certain other pages first [1, 4]. Such systems can allow the user to maximally explore the site along educationally effective paths only.

In addition to making only appropriate navigation paths available, it would be good if the user could be provided with the necessary information so that he can avoid redundant paths. A redundant path is not instrumental in accomplishing a certain goal given what parts of the site the user has already visited. Of course, a visited page would be redundant in this sense. However, there are additional pages that may have

become redundant as a side effect of the user's visiting a page.

We have developed an approach to solve these problems. We use disjunctive and conjunctive constraints to describe the prerequisites between the pages of a web site, and then an algorithm computes which pages can be accessed next and which pages are redundant to visit for a given goal concept. We have implemented a web-based prototype using the W3-mSQL scripting language [7]. This algorithm can also be used as a basis to adaptively annotate links to help the user navigate efficiently through the hyperspace [2].

## 2. DESCRIBING PREREQUISITES

To simplify the discussion, we assume that every concept is described on one page and that the user understands that concept after visiting the page. Prerequisites are described in form of disjunctive and conjunctive constraints. A disjunctive constraint  $p_1 \vee p_2 \vee \dots \vee p_n \rightarrow p$  asserts that before page  $p$  may be visited, at least one of the pages  $p_1, p_2, \dots, p_n$  must be visited, whereas a conjunctive constraint  $p_1 \wedge p_2 \wedge \dots \wedge p_n \rightarrow p$  requires that all pages  $p_1, p_2, \dots, p_n$  must be visited before  $p$  is. Figure 1 shows a preference graph with the prerequisites  $a \vee b \rightarrow c$ ,  $a \rightarrow d$ ,  $c \wedge d \rightarrow e$ , and  $c \rightarrow f$ .

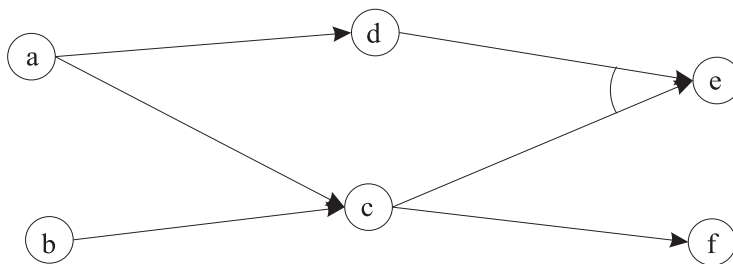


Figure 1: A preference graph with one conjunction  $e$ .

For the sake of the following discussion, assume that all pages are hyperlinked to all other pages forming a fully connected graph. Then, any page can be accessed from any other page if its necessary prerequisites are satisfied. If they are satisfied, the page is *enabled* and can be visited at any time. Of course, it would be redundant to visit a visited page again. However, there may be pages that are redundant although they have not been visited at all.

This redundancy can only be caused by disjunctive constraints. Consider the preference graph in Figure 1 and our goal is to learn more about the concept on page  $f$ . Assume none of the pages  $a$  to  $e$  has been visited and we now visit  $b$ . This enables  $c$  because it is constrained by a disjunction. Since  $c$  has been enabled, visiting  $a$  is completely redundant in reaching  $f$ . However, if we make  $e$  our

goal,  $a$  cannot be made redundant after visiting  $b$ , since it still is needed for visiting  $e$  via  $d$ . This shows that finding redundant pages is not trivial, because the redundancy of a page is not a local property but depends on the interaction of conjunctive constraints that may be far apart in the dependency graph.

### 3. REDUNDANCY PROPAGATING PATHS

The goal is to find a way to compute pages that have become redundant after another page has been visited. Furthermore, the algorithm should be fast and as local as possible, i.e., should not have to traverse large parts of the dependency graph each time a page is visited.

As mentioned earlier, only disjunctive constraints cause redundancies. Conjunctions don't because all of their prerequisites have to be visited no matter what happens in other parts of the graph. Note in Figure 1 that a page is not redundant if it still can cause a page that may be required to become enabled. This is indeed the case with page  $a$  even after  $b$  has been visited and  $e$  is the goal concept. This characteristic can actually be computed and stored locally in each node of the dependency graph. This initialization is done before the user traverses the site. While the user is navigating through the pages, visited pages may cause some pages to become redundant which then can be appropriately reflected in the user interface.

#### 3.1 Initializing the Preference Graph

The initialization of the algorithm computes local information stored as a number in each node of the dependency graph. This number  $\alpha$  is equal to how many conjuncts can be reached from this node, where  $p_1, \dots, p_n$  are the conjuncts of the conjunctive constraint  $p_1 \wedge \dots \wedge p_n \rightarrow p$ .

This number can be computed for the whole dependency graph in  $O(|N|)$  where  $N$  is the set of all nodes in the dependency graph. (The terms node and page will be used interchangeably in the rest of this paper.) Note that this computation does not depend on the number of pages in the site but on the number of pages mentioned by the prerequisites.

The algorithm traverses the graph in topological order and propagates for each conjunct a distinct number back to the root of the graph. Then, each node counts the number of distinct numbers that have arrived at the node, which is equal to the number  $\alpha$  of conjuncts that can be reached from this node. The  $\alpha$ 's for the dependency graph of Figure 1 are displayed in Figure 2.

#### 3.2 Updating the Preference Graph

Whenever a page is visited, the pages that have become redundant must be marked as such. This can be done based on  $\alpha$  alone.

First we need to introduce the concept of a redundancy-propagating path, or RP path for short. A vertex  $(p, q)$  in a preference graph propagates redundancy if  $q$  is a disjunctive node and  $\alpha(q) = \alpha(p)$  or  $q$  is a conjunctive node and  $\alpha(q) = \alpha(p) + 1$  (see the dark arrows in Figure 3). If  $(p, q)$  propagates redundancy then  $p$  and  $q$  support the same set of conjuncts and in this case if  $q$  is redundant, then so must be  $p$ . An RP path is therefore a path that consists only of vertices propagating redundancy. The path's direction is "backwards," i.e., in the reverse direction the vertices indicate. For instance, in Figure 3  $i-h-f-e$  is an RP path.

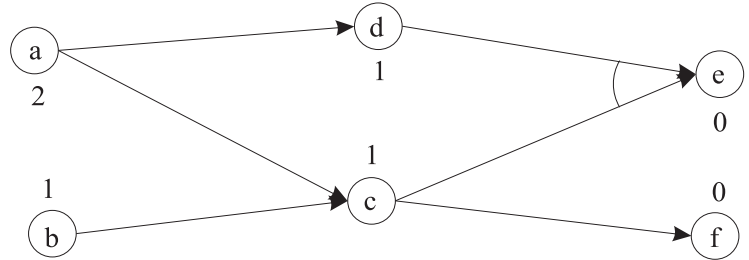


Figure 2: Preference graph annotated with the number of conjuncts.

With this observation, we can state the algorithm must mark the pages redundant as a result of visiting a page. When a disjunctive node is enabled due to a visit of one of its disjuncts, this node propagates back on all RP paths the fact that it has also become redundant. Each node along this path is set to redundant as well. If a node is already redundant, the propagation can stop.

The update algorithm has a constant expected complexity because the propagating subgraph has fewer nodes than the dependency graph and each node becomes redundant at most once.

### 4. APPLICATIONS AND EXTENSIONS

The algorithm was described in the context of its simplest use, where a node in the preference graph is a page which describes one concept, and visiting this page means understanding the described concept. Our first web-based prototype does exactly this and informal testing was rather positive. Links to pages are indeed hyperlinks whereas disabled links are plain text. Enabled non-redundant links were shown in bold and colored blue whereas enabled redundant links were colored and displayed in parentheses. This provides simple but useful feedback.

This simple use of the algorithm can be applied to manuals or web sites providing class material. Assume a user is trying to read about a certain concept  $x$  and is first referred to concepts  $a$ ,  $b$  and  $c$ . These concepts may require him to read about a further concept. After a short while, he has completely forgotten that he wanted to know about  $x$  and is trapped in one of those never-ending explorations. Another problem is that some of the concepts require understanding only one of several possible prerequisites. For instance, to understand what a primitive data type is, it's good enough to read either about characters, integers or floats. There is no need to read about them all.

We are currently working on a generalized version of our

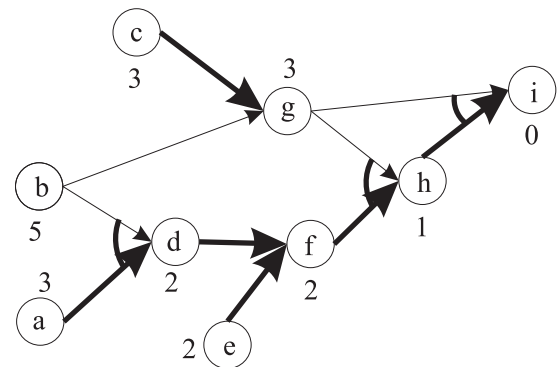


Figure 3: A preference graph with RP paths.

approach. The nodes need not directly refer to pages, at least not all nodes. This allows expressing a wider range of constraints, e.g., as a conjunction of disjunctions. Furthermore, it is completely up to the application designer to decide what is done with the different kinds of nodes characterized by the properties redundant, enabled and visited. Adding negation to the constraint language is not necessarily a good thing, because then the set of enabled pages becomes non-monotonic which may very well be confusing to the users.

Describing the preferences between the pages may be tedious and also at the wrong level for an end user, e.g., an instructor creating web-based course material. For this purpose, we are developing a language that allows the user to describe the organization of the material at a much higher level directly relating to the course and not to the implementation of the course materials. This description will then be translated by the system into the disjunctive and conjunctive constraints and into preference ordering of the enabled, non-redundant pages [6].

## 5. CONCLUSIONS

We have presented an algorithm that can be used to detect redundant pages in hypermedia systems. The algorithm is efficient and can be easily implemented on the web. It can be used in its basic form or as a part of a system that uses more detailed information about the user.

We have implemented a prototype and are currently working on a more integrated system to be used by educators to build web sites supporting their courses. We are planning to field-test whether our approach is indeed more than just an interesting algorithm, which is something too often ignored in the field of adaptive hypermedia systems [3].

## 6. REFERENCES

[1] Brusilovsky, P. Efficient techniques for adaptive hypermedia. In *Intelligent hypertext: Advanced techniques for the World Wide Web*, C. Nicholas and J.

Mayfield, Eds., Lecture Notes in Computer Science, Vol. 1326, Berlin: Springer-Verlag, 1997, pp. 12-30.

- [2] Brusilovsky, P., Pesin, L. and Zyryanov, M. Towards an adaptive hypermedia component for an intelligent learning environment. In *Human-Computer Interaction*, L. J. Bass, J. Gornostaev, and C., Unger Eds., Springer-Verlag, Berlin, 1993, pp. 348-358.
- [3] Eklund, J. and Brusilovsky, P. The value of adaptivity in hypermedia learning environments: A short review of empirical evidence. In *Proceedings of Second Adaptive Hypertext and Hypermedia Workshop at the Ninth ACM International Hypertext Conference Hypertext'98*, : P. Brusilovsky and P. De Bra, Eds., Pittsburgh, Computing Science Reports, Report No. 98/12, Eindhoven University of Technology, Eindhoven, 1998, pp. 13-19.
- [4] Hauck, F. J. Supporting Hierarchical Guided Tours in the World Wide Web. In *Fifth International World Wide Web Conference*, May 6-10, 1996, Paris, France.
- [5] Hegarty, M., Quilici, J., Narayanan, N. H., Holmquist, S., & Moreno, R. Multimedia instruction: Lessons from evaluation of a theory-based design. To appear in the *Journal of Educational Multimedia and Hypermedia*, Association for the Advancement of Computing in Education., 1999.
- [6] Hübscher, R. Reusable Web Sites for Content Delivery. In *ED-MEDIA 2000—World Conference on Educational Multimedia, Hypermedia and Telecommunications*, Seattle, 1999.
- [7] Jepson, B. and Hughes, D. J. Official Guide to Mini Sql 2.0. New York, NY: John Wiley & Sons, 1998.