(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

Computers & Education 58 (2012) 1296–1307

Contents lists available at SciVerse ScienceDirect

Computers & Education

journal homepage: www.elsevier.com/locate/compedu

Interactive visualization of dependencies

Camilo Arango Moreno [a], Walter F. Bischof [b,*], H. James Hoover [b]

[a] Google Inc., 1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
[b] Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E9, Canada

ARTICLE INFO

Article history:
Received 16 September 2011
Received in revised form
22 December 2011
Accepted 27 December 2011

Keywords:
Dependency visualization
Course structure

ABSTRACT

We present an interactive tool for browsing course requisites as a case study of dependency visualization. This tool uses multiple interactive visualizations to allow the user to explore the dependencies between courses. A usability study revealed that the proposed browser provides significant advantages over traditional methods, in terms of learnability, efficiency and user confidence. The results are discussed within a general framework for interactive visualization of dependencies.

1. Introduction

This paper proposes a new interactive visualization of complex dependencies, i.e., relationships between pairs of objects, in which the first requires the second to accomplish some action. Dependency problems appear in a variety of fields. In software development, for example, a piece of software may depend on many libraries, which in turn may depend on others. In project management, certain tasks of a project depend on other tasks to be completed or on events that must be executed simultaneously. In workflow modeling, some decisions may require the approval of one or more executives in a certain order, followed by approval by an oversight committee. People working on problems that contain dependencies are often given just the relationship information among elements and are then forced to understand complex dependency systems on their own. Using visual representations of dependency graphs can provide a valuable tool to understand these dependencies and to extract additional information.

Simple dependency problems may be modeled by adjacency relationships among elements, and therefore can be represented by a simple directed graph. Others require a more complex model, as they contain different types of relationships or dependencies may have rules associated with them. In the present paper, we focus on complex dependency graphs that can be found in University course requisites: For example, in order to take the course Cmput411 (Introduction to Computer Graphics), students need to take the courses Cmput204 (Algorithms I), Cmput301 (Introduction to Software Engineering) and Math120 (Linear Algebra) in a previous term; for Cmput301, they need to take the courses Cmput201 (Practical Programming Methodology), and Cmput115 (Programming with Data Structures) or Cmput175 (Introduction to the Foundations of Computation II), and so on. In some cases, the requisites courses must be taken in a previous term (prerequisites), in other cases they can be taken at the same time (corequisites). Consequently, the relationships formed by these courses' requisites cannot be modeled by a simple directed graph.

A program of studies plan is a collection of prerequistites that need to be satisfied in order for a student to graduate. That is, it is a scaling-up of the course requisite problem. In highly prescribed programs of study, especially cohort-based, such as engineering and medicine, there is little need by students for dependency visualization tools as they have very little choice in optional courses. For the larger body of students in Arts and Science, the trend is not only toward more flexible programs of study, but for more interdisciplinary programs of study. In this context the prerequisite structure has a major impact on the program planning process. Also increasing numbers of students are transferring into CS after a couple of years of study in another discipline, thus adding a transitional aspect to the planning process.

In flexible programs, students are responsible for mapping out the optional part of their program, which could easily be over half of their courses. They then take that plan to an advisor for approval. In our department, our anecdotal evidence is that about 20% of students make

* Corresponding author. Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E9, Canada.
E-mail address: wfb@ualberta.ca (W.F. Bischof).

0360-1315/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.compedu.2011.12.027

mistakes in their program plan that are then caught by experienced advisors during the program approval process. Many of these mistakes are caused by not understanding the chain of dependencies among courses. For example, the program requirements might state that you must take only three 200-level courses, when a student's intended plan for courses at their 4th year would in fact require them to take all of the core 200-level courses.

The number of students taking flexible programs with a broader range of options is increasing, and advisor capacity is reducing both in number of advisors and their experience. So the need for advising tools that understand and display the dependency structure will continue to increase.

In the work reported here, we use rich-prospect browsing to visualize the dependencies defined by the course requisites. Rich prospect visualizations (Ruecker, 2003) use a browsing interface to support the tasks of understanding, interpreting, or systemizing the material in a domain.

2. Dependency visualization

Choosing the right representation is crucial for creating an effective visualization of object relationships, including dependencies. While an effective representation can greatly simplify a problem, e.g., a Karnaugh diagram, others can make it more confusing. Multiple guides for good design have been published. For example, Tufte (1995) explains correct and incorrect ways of representing information based on examples from graphical design, cartography and art. Likewise, Lidwell, Butler, and Holden (2003) present a set of design principles for multiple disciplines. In the following, we first discuss general principles and guidelines that led to the development of the work reported here, then we present several examples of simple dependency visualization, and finally we proceed to more complex dependency visualizations.

Preece (2002) discusses several usability and user experience goals that are important for the evaluation of interaction products:

- Effectiveness: can a task be performed successfully with the product?
- Efficiency: is the product faster to use? Does the product requires less resources?
- Safety: does use of the product have undesirable side effects or increase the risk of undesirable events? (Note an efficient product could reduce safety).
- Utility: does the product help the user perform the tasks that they need to perform?
- Learnability: is the product easy to learn?
- Memorability: can the user easily remember how to use the product? It is these principles that guided the design of our interactive tool for dependency visualization

For simple dependency visualizations one can use a graph-drawing algorithm that defines the placement of vertices and edges of a particular graph type, following drawing conventions and drawing rules as much as possible. Examples of examples of graph drawing approaches include dots and lines diagrams, circular diagrams, and tree-maps (see Fig. 1). Dots and lines are used to represent general directed or undirected graphs. They represent vertices as dots and edges as lines on free placement coordinate system. Circular diagrams show general graphs using dots and lines in a polar coordinate system. In these diagrams, vertices are arranged around a circumference and edges are drawn using straight or curved lines. Edges may be bundled together to reduce clutter and allow to easy identification of their source and target. Tree-maps were proposed by Shneiderman (1992) as a method for drawing n-ary trees. In these diagrams, edges are represented by inclusion, i.e., nodes are represented as boxes and children nodes are included inside their parent's boxes.

Graph visualization can be enhanced though interactivity. Interactive controls dynamically modify and animate the visual features of the graph (position, color and saturation of nodes and edges) to allow users to easily understand and navigate graphs. Navigation technique include not only simple pan and zoom, but also more sophisticated techniques, including, for example, the fisheye view of Sarkar and Brown (1992) to allow the user to zoom on a particular section of a diagram without losing a global view of the graph.

Kerr (2003)'s Thread Arcs project is a visualization technique for describing the structure of email threads (see Fig. 2). It displays the messages as dots in a straight line, sorted by chronological order and then draws arcs among them to represent the *reply-to* relationship. The visualization offers some interactive features to allow the user to highlight the diagram according to the characteristics of the messages, and to inspect the messages. The advantages of the thread arcs visualization over other tree visualizations is its compactness and its emphasis on chronology, which is fundamental in following an email conversation. The diagram does not contain any text information about the messages, forcing the user to use the interactive features to find this information.

Circos (Krzywinski, Visited June 15, 2009) is a software tool that generates visualizations of genomic data and general 2-dimensional data. Data is displayed in a circular layout and the relationships among elements are drawn as arcs. The project was conceived to visualize relationships between genomes. The use of a circular layout was chosen to minimize the overlap among the lines that represent relationships, making the relationships easier to visualize. Circos provides considerable flexibility for generating graphs in a circular layout.

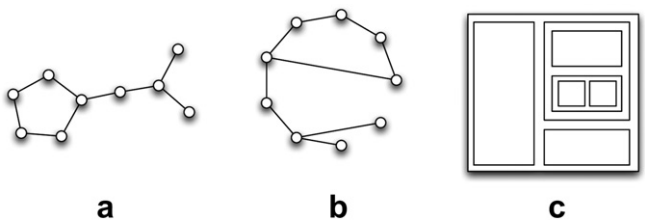


Fig. 1. Graph drawing examples. a) Dot and lines. b) Circular layout using straight lines as edges. c) Tree-map.

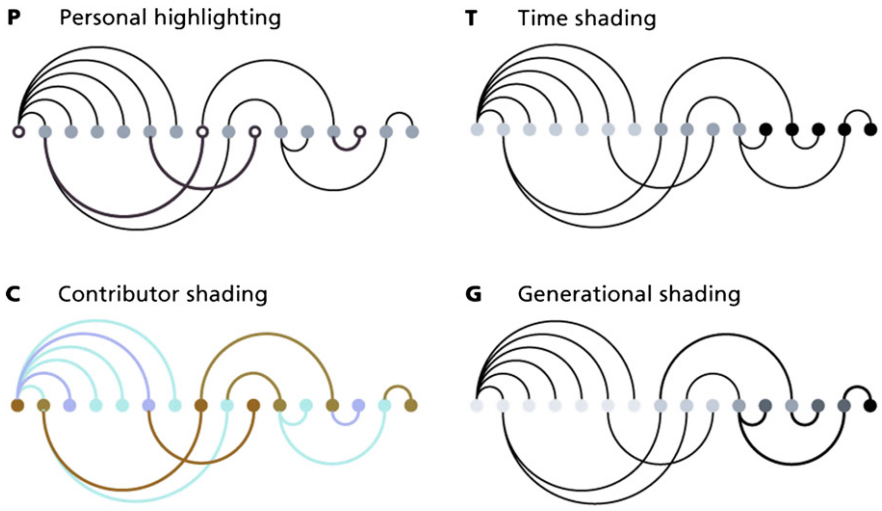


Fig. 2. Email thread visualized using Thread Arcs using different highlight schemes. *From Kerr (2003).*

Several types of plots, including, for example, histograms, scatter plots and text, can be embedded inside the circular graph. One disadvantage of the tool, however, is that the generated images are static.

Flare (Heer, 2009) is a toolkit for creating interactive data visualizations that can be published on the Internet using the Flash plugin. It is based on the Prefuse toolkit, which was developed for the Java platform (Heer, Card, & Landay, 2005). The Flare toolkit includes tools for performing data management, visual encoding, animation, and interaction. The framework is highly extensible and customizable. The visualizations in our system were created using this toolkit.

3. Proposed interactive visualization

We implemented a course browser to test our interactive dependency visualization approach, with the purpose of visualizing the network of courses at our university. For a given course, there may be courses that must be taken before, called *prerequisites*, and courses that may be taken before or simultaneously, called *corequisites*. Each of the prerequisites and corequisites can have associated conditions. For example, it is possible that, in order to take a given course A, a student has to choose between taking courses B and C, or D, E and F. In turn, courses B, C, D, E and F can have their own requisites and conditions. For the purpose of visualization of course dependencies, only the course requisite information is relevant. We thus focused on defining a simple language capable of expressing this information, rather than creating a complex language that tried to express all information, including requisites, equivalent courses and restrictions. The elements of the requisite information are well defined, as they contain just courses and logical operators.

The proposed Course Browser was developed as an interactive tool for exploring a catalog of courses, with an emphasis on easy exploration of the course dependencies through the use of multiple visualizations. A screen shot of the main elements of the course browser user interface is shown in Fig. 3, and an interactive version of the Course Browser can be accessed online (Moreno & Hoover, 2011). The Course Browser groups courses into collections. A *collection* is a group of courses that share common characteristics, for example, belonging to the same department. The prototype includes predefined collections by department, number of credits and number of requisites. Each collection is defined by a logical statement, allowing the introduction of user-defined collections.

Once a collection is selected, the user can explore the associated courses using an overview diagram or a list view. The *collection overview* is an interactive visualization of the courses in a collection (see Fig. 3). This diagram displays how courses inside the collection relate to each other, according to their requisite information. The *course list* uses a more conventional data grid to display the courses in a tabular way. The grid contains columns for the code, title and number of credits of each course, and the columns are sortable, allowing the user to quickly organize and locate courses.

When the user selects a course, either by using the *collection overview diagram* or the *course list*, two more views become available to explore the details of the selection. The *course description* view (see Fig. 4) shows all the details of the course available in the course calendar: course code, department, number of credits, frequency and description. In addition, the requisite information is displayed using a *requisite box diagram*. Furthermore, a *Course Requisite Graph* (see Fig. 5) displays all the requisites of the course, including indirect dependencies.

The Course Browser prototype contains two diagrams designed to display direct and indirect dependencies between courses: the *collection overview* and the *course requisite graph*. Both share the same design principles, so we refer to them as *circular diagrams* (see Figs. 3 and 5). The purpose of these diagrams is not to encode all the complete information about the tree, but to give the user an overview of how courses are related. The arcs hide part of the tree complexity, in exchange for a transitive view of the course requisites.

Circular diagrams position the nodes along the circumference and use curved arrows to express their relationship. No overlapping between the nodes and arrows is possible, as is the case with other graph layouts. For ease of searching, nodes are sorted alphabetically along the circumference. Arrows show a relationship between two courses in the dependency tree; more precisely, each arrow represents a path from the root of the requisite tree to one of its leaves, which must contain a *pre* or a *co* element. This path can traverse multiple nodes of type *allOf, anyOf* or *atLeast*. In the diagram, choices and mandatory requirements are characterized by the saturation of their arrows. This property is determined by a recursive weight distribution algorithm. Weights are assigned as follows. The root of the dependency tree is assigned a weight of 1, and the weight is distributed to each of the children according to the following rules, which are applied recursively:

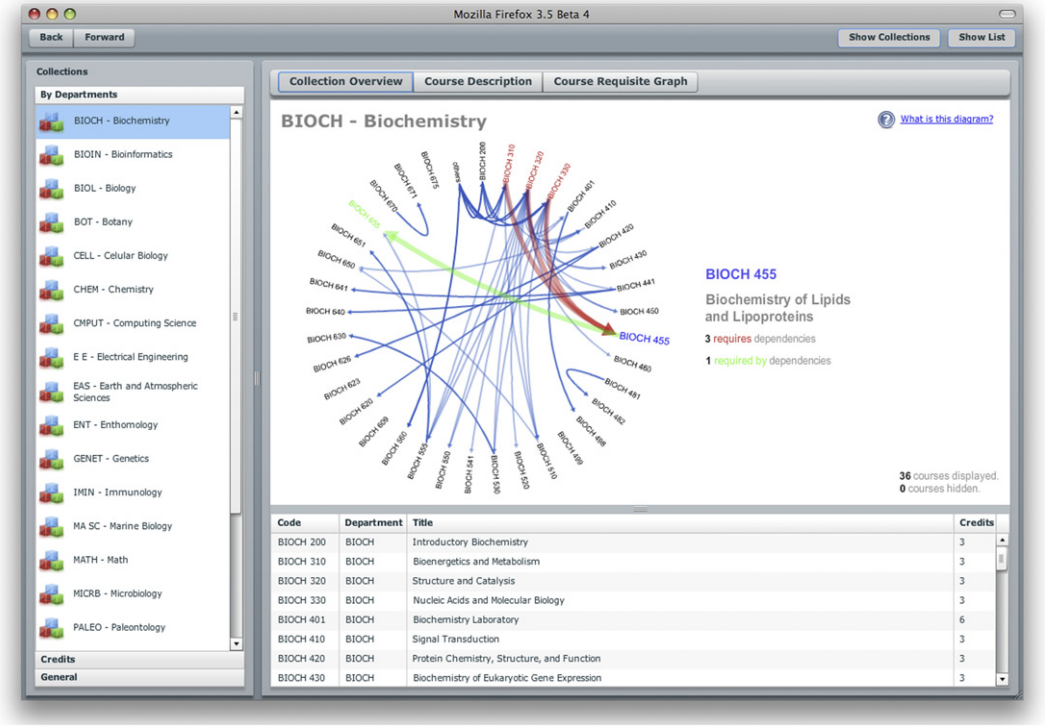


Fig. 3. Screen shot of the user interface of the Course Browser, showing the collection overview of courses.

1. If the node is of type *allOf* and its weight is *w*, each of its immediate children are given the weight *w*.
2. If the node is of type *anyOf* and its weight is *w*, each of its immediate children are given the weight *w*/*c*, where *c* is the number of children of the node.
3. If the node is of type *atLeast* and its weight is *w*, each of its immediate children are given the weight *n***w*/*c*, where *c* is the number of children of the node and *n* is the parameter of the node.

An example of the weight assignment is shown in Fig. 6. The root node is assigned a value of 1. As the root node is of type *allOf*, each of its children is assigned a value of 1 as well. The first child is of type *anyOf*, therefore, its weight is distributed evenly among its children, giving each of them a weight of 1/2. The second child is a leaf, so it has no children to assign weights to. Finally, the third child is an *atLeast* element with a parameter value of 2, and 3 children. Thus, each of its children is given a weight of 2/3.

The circular diagrams meet important design objectives, including stability, compactness, scalability and attribute highlighting. As the course titles are sorted alphabetically along the circumference, the diagram remains stable as new nodes are added, and it is easy to locate

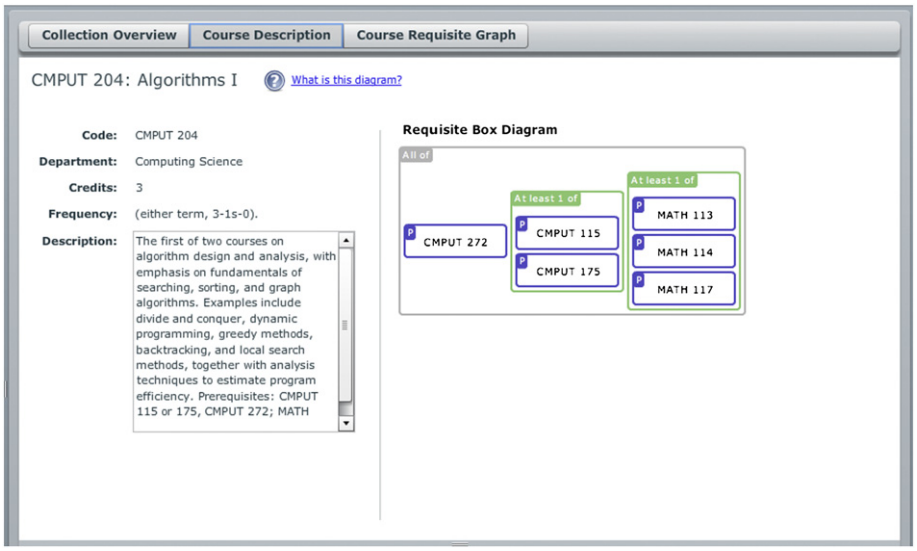


Fig. 4. The Course description view of the Course Browser.

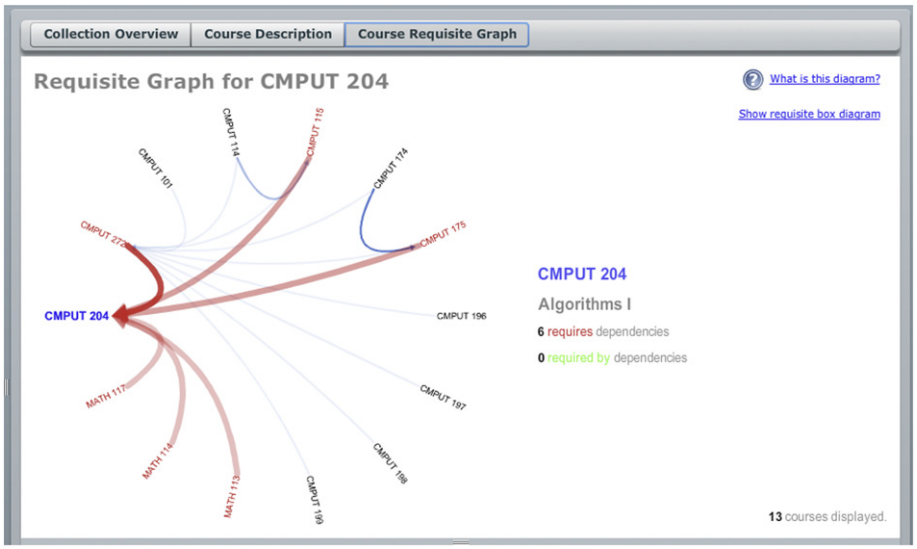


Fig. 5. The Course requisite graph view of the Course Browser.

a particular course. The size of the diagram is independent of the number of nodes and it can be adjusted to fill the size of the screen. Its complexity increases gracefully as more nodes are added. Finally, it allows the user to distinguish courses with different characteristics by scanning the diagram: fundamental courses have many arrows pointing to them; courses with no requisite have arrows originating from them; courses that are not required by other courses have no arrows pointing to them; finally, courses that require many courses have many opaque arrows originating from them while courses that require a few courses but have many choices have many translucent arrows.

As the number of courses increases, the circular diagrams become gradually more complex. For this reason, the number of nodes in each diagram was limited to 150 elements and interactive features were implemented. For example, the user may highlight a course code by positioning the mouse pointer on top of it to consult basic information about it.

3.1. Collection overview

The *Collection Overview* diagram is a circular diagram that displays the courses of a particular collection. Its objective is to show how the courses inside the collection depend on each other. This diagram is useful for identifying the fundamental courses in a collection and courses with no requisites, as well as courses that are not required by other courses.

The diagram has several interactive features. When the user hovers the mouse over a course, a panel appears on the right of the diagram displaying basic information about the course. In addition, the arrows are highlighted using different colors for incoming and outgoing requisites. When the user selects a course, by clicking its code on the diagram or using a different view, the nodes that are related directly or indirectly to the selection are shown and all unrelated ones are hidden. This enhances the readability of the diagram considerably.

3.2. Course requisite graph

The *Course Requisite Graph* is a circular diagram that displays all the direct and indirect requisites of the selected course. This diagram is useful in course planning as it displays all the possible courses that must be taken before a given course. A course must be selected in order to display this diagram. When the mouse hovers over the course code the course information is displayed and the arrows are highlighted.

	Weight
<allOf>	1
<anyOf>	1
<pre>CMPUT 115</pre>	1/2
<pre>CMPUT 175</pre>	1/2
</anyOf>	
<pre>CMPUT 272</pre>	1
<atLeast n="2">	1
<pre>MATH 113</pre>	2/3
<pre>MATH 114</pre>	2/3
<pre>MATH 117</pre>	2/3
</anyOf>	
</allOf>	

Fig. 6. Example of the weight assignment for calculating the saturation of the arrows in the circular diagrams.

Clicking a node changes the selection to its respective course. Furthermore, an optional popup window displays the *Requisite box diagram* of the highlighted course.

4. Usability study

We conducted a usability study to assess the Course Browser according to ease of use, learnability, effectiveness to perform course planning tasks, and user confidence in the results. For comparison, we used the web-based tool that students presently use to browse the course catalog. We asked students to perform equivalent tasks with both tools and measured the completion time and accuracy of their answers. Then we asked the students about their experience and gathered their suggestions on how to improve the tool.

4.1. Traditional method

The current Bear Tracks course browser at our university (called here the Traditional Method) hosts the course catalog along with other information. The navigation scheme for the Traditional Method is shown Fig. 7. Normally, the user starts by searching a course; the system then displays a list of the courses matching the search criteria; the user then accesses detailed course information; and finally, the user may go back to starting a new search.

The *browse course catalog* displays a search form, which displays a term selector (fall/winter or spring/summer). When the term is chosen, the system shows the rest of the search fields (see Fig. 7). The system allows the user to search by simple filters, e.g., department code or course number, but also allows more complex searches, such as "all the courses in the Physics department" or "all third-year courses in the Computing Science department". The results of the query are presented as a list that contains the course's code, title, and description, as shown in Fig. 8. To see the complete information for a course, the student can click on the hyperlink of a course title.

The course view shows the full calendar descriptions of the course (see Fig. 9), including basic course details including units, fee index, approved hours, calendar terms, course components and a general description. The course requisites are contained inside the course description, usually at the end, and are written in natural language. Although some departments follow the same format to express them, there are no consistent rules on how to express the requisites. Only immediate requisites are included in the description of each course, implying that, in order to discover indirect requisites, the user has to start with a search of the basic course, read the requisites of that course, and repeat the process for each of them. It is not uncommon to run five or more searches in order to find all requisites for a given course.

4.2. Methods

4.2.1. Participants

The usability study was conducted with fifteen undergraduate students from the University of Alberta. They were enrolled in a variety of programs, and had different levels of experience with registration and course planning.

4.2.2. First interview

In the first part of the study, participants were asked about their current year of study and the number of courses they had taken; their understanding of the terms *prerequisite* and *corequisite* in the context of course registration; the methods they used for finding information about the courses and their requisites, as well as difficulties with understanding course requisites; and their current practices for planning their course program. Since they were crucial for the task-solving phase, any terms unfamiliar or imprecisely understood by the participant were explained by the interviewer prior to the task.

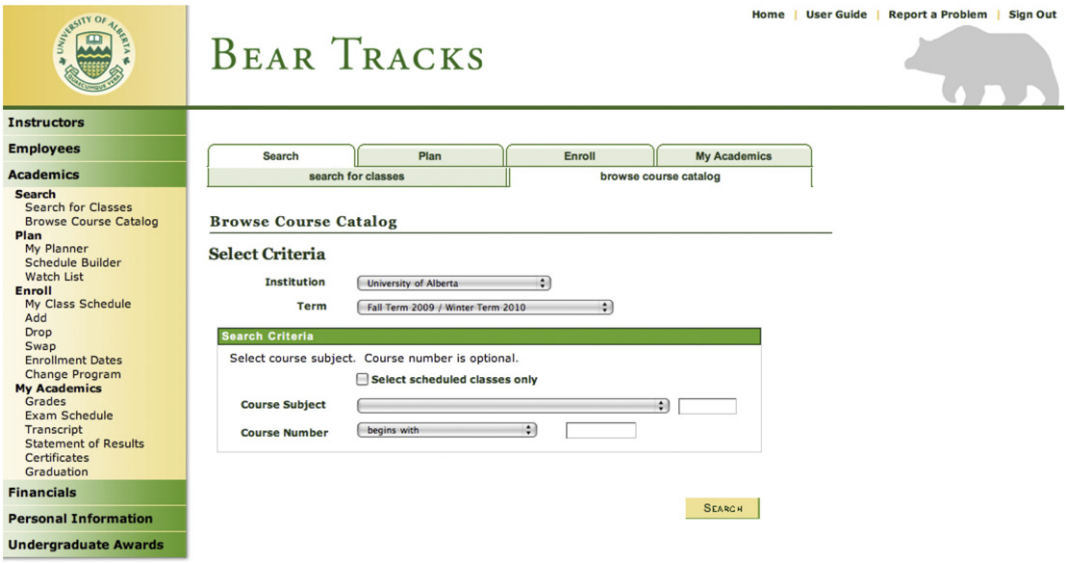
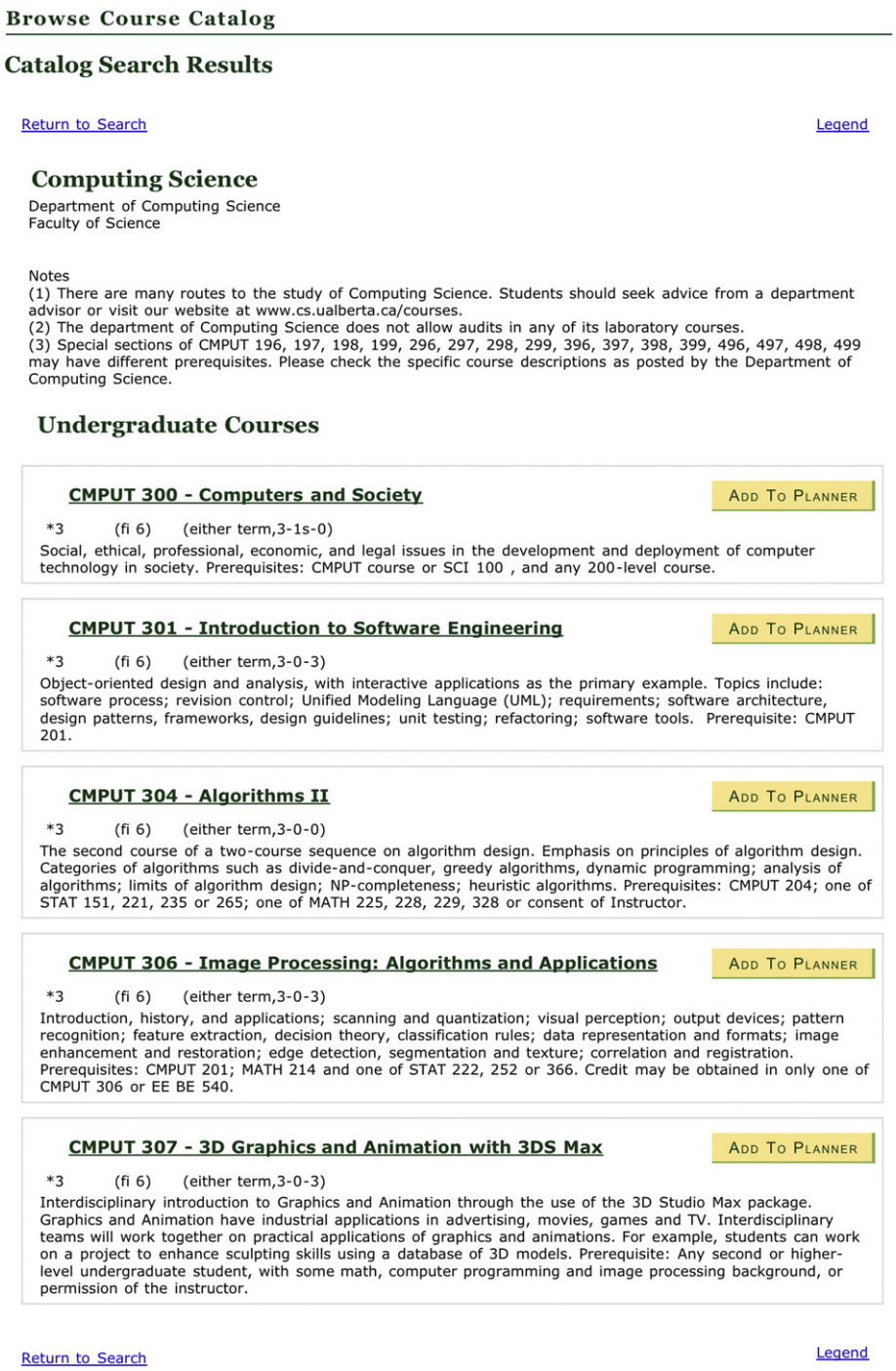


Fig. 7. Course catalog search screen of the Traditional Method.



Browse Course Catalog

Catalog Search Results

Return to Search Legend

Computing Science

Department of Computing Science
Faculty of Science

Notes
(1) There are many routes to the study of Computing Science. Students should seek advice from a department advisor or visit our website at www.cs.ualberta.ca/courses.
(2) The department of Computing Science does not allow audits in any of its laboratory courses.
(3) Special sections of CMPUT 196, 197, 198, 199, 296, 297, 298, 299, 396, 397, 398, 399, 496, 497, 498, 499 may have different prerequisites. Please check the specific course descriptions as posted by the Department of Computing Science.

Undergraduate Courses

CMPUT 300 - Computers and Society — Add To Planner

*3 (fi 6) (either term,3-1s-0)

Social, ethical, professional, economic, and legal issues in the development and deployment of computer technology in society. Prerequisites: CMPUT course or SCI 100 , and any 200-level course.

CMPUT 301 - Introduction to Software Engineering — Add To Planner

*3 (fi 6) (either term,3-0-3)

Object-oriented design and analysis, with interactive applications as the primary example. Topics include: software process; revision control; Unified Modeling Language (UML); requirements; software architecture, design patterns, frameworks, design guidelines; unit testing; refactoring; software tools. Prerequisite: CMPUT 201.

CMPUT 304 - Algorithms II — Add To Planner

*3 (fi 6) (either term,3-0-0)

The second course of a two-course sequence on algorithm design. Emphasis on principles of algorithm design. Categories of algorithms such as divide-and-conquer, greedy algorithms, dynamic programming; analysis of algorithms; limits of algorithm design; NP-completeness; heuristic algorithms. Prerequisites: CMPUT 204; one of STAT 151, 221, 235 or 265; one of MATH 225, 228, 229, 328 or consent of Instructor.

CMPUT 306 - Image Processing: Algorithms and Applications — Add To Planner

*3 (fi 6) (either term,3-0-3)

Introduction, history, and applications; scanning and quantization; visual perception; output devices; pattern recognition; feature extraction, decision theory, classification rules; data representation and formats; image enhancement and restoration; edge detection, segmentation and texture; correlation and registration. Prerequisites: CMPUT 201; MATH 214 and one of STAT 222, 252 or 366. Credit may be obtained in only one of CMPUT 306 or EE BE 540.

CMPUT 307 - 3D Graphics and Animation with 3DS Max — Add To Planner

*3 (fi 6) (either term,3-0-3)

Interdisciplinary introduction to Graphics and Animation through the use of the 3D Studio Max package. Graphics and Animation have industrial applications in advertising, movies, games and TV. Interdisciplinary teams will work together on practical applications of graphics and animations. For example, students can work on a project to enhance sculpting skills using a database of 3D models. Prerequisite: Any second or higher-level undergraduate student, with some math, computer programming and image processing background, or permission of the instructor.

Return to Search Legend

Fig. 8. Course catalog result screen of the Traditional Method. This shows the results for courses in the Computing Science department with codes beginning with "30".

4.2.3. Course planning tasks

In the second part of the study, participants were asked to solve a series of typical course planning tasks, using our Course Browser and the Traditional Method. With each tool, we asked seven questions of increasing difficulty, starting with counting the courses in a certain department and ending with the planning of all the courses required in order to take a fourth year course. The order of the tools was counterbalanced across participants, and with each tool, the participants had to solve similar problems. Before solving the tasks with the Course Browser, each participant was given 5 min for free exploration of the user interface. After that, a short tutorial video was shown and they were given the chance to ask questions about the application.

4.2.4. Second interview

In the third part of the study, an interview was conducted to determine the participants' experience with the tools, ease of use, ease of learning, and their level of confidence in their answers.

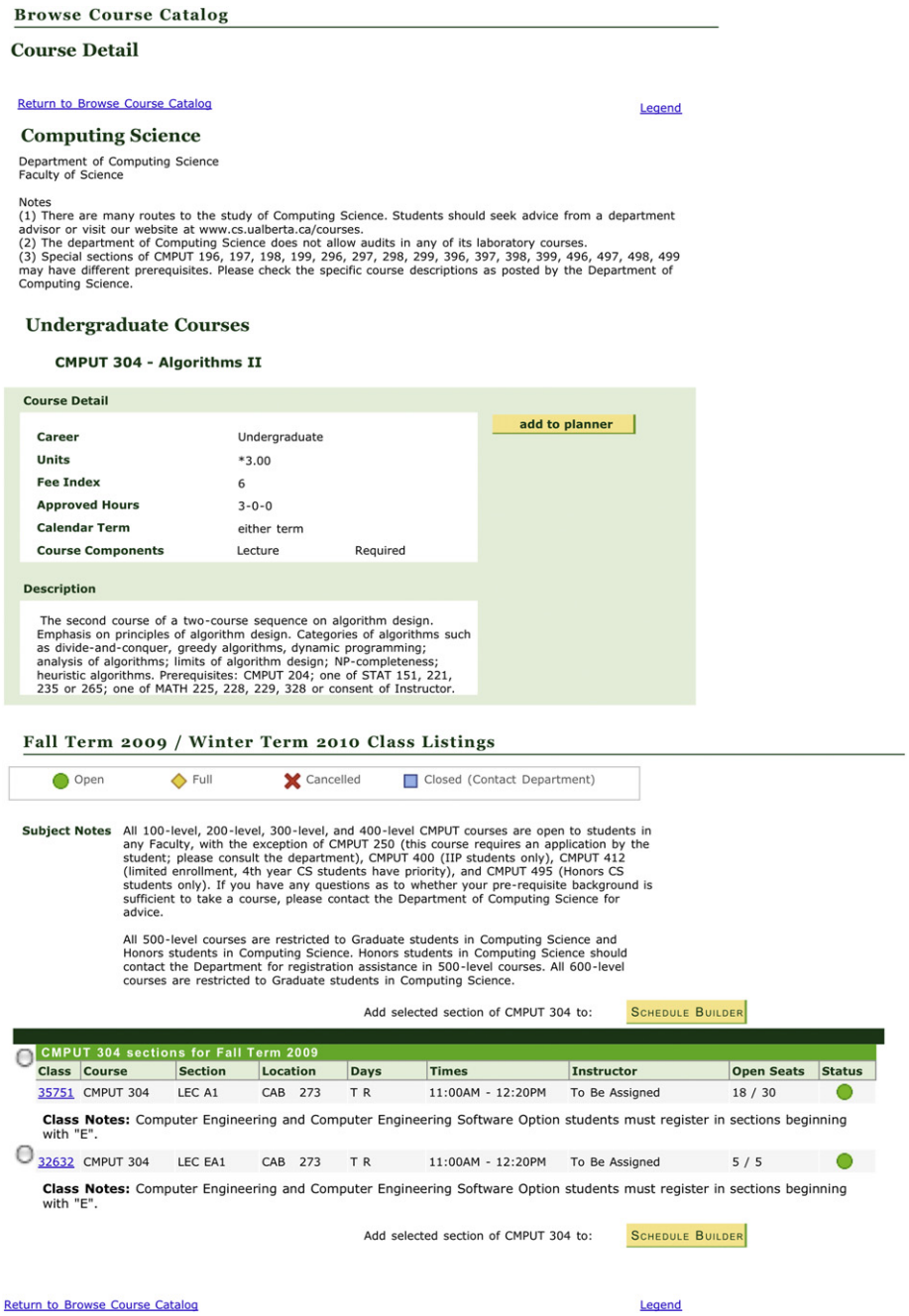


Fig. 9. Detail page for CMPUT 304.

4.3. Performance

In the task-solving phase, participants were asked to perform seven tasks related to course planning. The completion time and the correctness of the answers were recorded for each task, and the results are presented discussed below.

4.3.1. Results

A summary of the completion times for all seven tasks as well as correctness are shown in Table 1. Below, we discuss the results for each task.

Task 1 required the participants to count the number of courses offered by a certain department of the university. The average time taken by the participants to complete this task was significantly lower with the Course Browser than with the Traditional Method [$M_{CB} = 43.6$ s, $M_{TB} = 87.6$ s, $t(14) = 3.03$, $p < .01$, Cohen's D = 0.79] and had a significantly smaller spread [$S_{CB} = 22.8$ s, $S_{TM} = 49.1$ s, $F(14, 14) = 4.62$, $p < .01$]. No significant differences in correctness were observed.

Task 2 asked the participants to identify the courses with no requisites among a list of four. The average completion times were significantly smaller with the Course Browser [$M_{CB} = 91.9$ s, $M_{TM} = 242.3$ s, $t(14) = 9.39$, $p < .001$, Cohen's D = 2.43] and had a significantly

Table 1
Completion times and correctness for task solving.

	Time (s)		Correct (%)	
	TM	CB	TM	CB
Task 1	87.6	43.6	100	86.6
Task 2	242.3	91.9	53.3	93.3
Task 3	96.7	103.3	86.6	86.6
Task 4	224.1	225.1	20.0	46.6
Task 5	129.7	95.8	60.0	46.6
Task 6	137.6	62.5	13.3	86.6
Task 7	510.3	499.6	26.6	73.3

smaller spread [$S_{CB} = 26.9$ s, $S_{TM} = 69.4$ s, $F(14, 14) = 6.68$, $p < .001$]. Correctness was significantly higher with the Course Browser [CB 93.3%, TM 53.3%, $\chi^2(1) = 4.17$, $p < .05$].

Task 3 had the purpose of testing the understanding of the concepts *prerequisite* and *corequisite*. The participants were given a course and a list of courses. First, they were asked to choose the ones that could be taken *before* in order to meet the requirements of the course. Then, they were asked to choose the ones that could be taken *simultaneously* to meet the requirements. Some of the courses in the list were prerequisites, others were corequisites, and some were not related at all. There were no significant differences in completion time or correctness between the two tools.

Task 4 required participants to create two different lists of courses that could be taken in order to meet the requisites of a given course. Neither the completion time nor correctness were significantly different for the two methods.

Task 5 consisted of completing the requisites for a given course. Participants were told to assume that they had taken some courses and were asked which courses are they were missing in order to meet the requisites of a given course. The average completion time was significantly lower with the Course Browser than with the Traditional Method [$M_{CB} = 95.8$ s, $M_{TM} = 129.7$ s, $t(14) = 2.25$, $p < .05$, Cohen's $D = 0.58$]. There were no significant correctness differences between the two tools.

Task 6 required the participants to perform a reverse dependency lookup of the course requisites. For this exercise, they were given a course and were asked to find other courses in the same department that directly required it. The average completion time was significantly lower with the Course Browser than with the Traditional Method [$M_{CB} = 62.5$ s, $M_{TM} = 137.6$ s, $t(10) = 2.80$, $p < .05$, Cohen's $D = 0.84$]. Correctness was significantly higher with the Course Browser than with the Traditional Method [CB 86.6%, TM 13.3%, $\chi^2(1) = 9.10$, $p < .01$].

In Task 7, the participants were given the description of a program as shown in the university calendar, and were asked to make a full plan, term by term, of the courses they would take in order to register for a fourth year level of course as soon as possible. The completion times were not significantly different among the two tools, but correctness was significantly higher for the Course Browser [CB 73.3%, TM 26.6%, $\chi^2(1) = 5.10$, $p < .05$].

As indicated in Section 4.2.3, we counterbalanced the order in which participants used the Course Browser and the Traditional Method. Furthermore, we checked whether order affected the results using a Split-Plot Factorial Anova with Order as a between-subjects and browser as a within-subjects factor. For none of the seven tasks was the effect of Order significant (all p's > 0.2).

4.3.2. Discussion

The Course Browser proved to be superior to the Traditional Method for tasks that required looking up indirect requisites and doing reverse requisite lookups, and it also proved to be significantly faster to navigate. For simple tasks that did not require students to look for more than one course, performance was similar with both tools.

The Course Browser was faster than the Traditional Method for navigating and finding courses. We attribute this to the fact that the representation of requisites was easier to understand in the Course Browser than the Traditional Method. In Task 2, for example, the participants had to identify the courses with no requisites among a list of four. With the Traditional Method, the participants had to start a new search for each of the courses, and many of the participants did not know where to find the information about the requisites. In contrast, the Course Browser offers participants multiple ways to find courses. Most participants started by choosing the department from the *collections view*. From there, they could either use the *Collection Overview diagram* and see if the course had any connections or consult the *Course description view* and take a look at the requisite diagram, which will inform them right away if the course had no requisites.

For reverse dependency lookup tasks, the Course Browser was better than the Traditional Method with respect to both, speed and correctness. This is illustrated with Task 6, "*If I have taken this course, what courses can I take next?*". Only 13.3% participants found the correct answer with the Traditional Method, but 86.6% did so with the Course Browser. This is so because the Traditional Method does not offer any means to perform reverse requirement searches. In fact, four of the participants stated that it was not possible to solve the task; the others tried to look for the answers manually, missing some of the courses. In contrast, the Course Browser was designed to simplify this type of task. All of the participants identified that the *Collection Overview diagram* allowed them to solve the question easily.

Finally, in the most difficult task, Task 7, the participants were given the description of a program as shown in the university calendar, and were asked to make a full plan, term by term, of the courses they would take in order to register for a fourth year course as soon as possible. We did not observe significant differences in the completion time, but a significant improvement in correctness with the Course Browser. In this task, the participants readily took advantage of the information provided by our tool. Some of the participants chose to use the box diagrams and the back and forward navigation, while others felt more comfortable with the *course requisite graph*. Still, this task involved considerable effort to complete and shows the need for a more complete tool for planning.

4.4. Second interview

The second interview captured the subjective impressions of each participant after completing the tasks. Some questions were answered on a five-level Likert scale (strongly agree, agree, neutral, disagree, strongly disagree) and some were open-ended. Some of the questions were aimed at a comparison of the Course Browser and the Traditional Method. The others were aimed at evaluating the Course Browser.

4.4.1. Comparison questions

For the following questions, recall that Bear Tracks is the public name students use for the Traditional Method. The question "*It is easy to find a course on [Bear Tracks/Course Browser*]" gave a significantly higher score, i.e., a higher number of "agree" and "strongly agree" answers, for the Course Browser [$W = 60.5$, $p < .05$]. The question, "*The [course catalog on the Bear Tracks/Course Description view on the Course Browser] displays enough information about the course for planning your program*." gave a significantly higher score for the Course Browser [$W = 40$, $p < .01$]. The question "*It was easy to identify the requisites of a course using [Bear Tracks/Course Browser]*" gave a significantly higher score for the Course Browser [$W = 9.5$, $p < .001$]. The question "*The requisites of courses are presented in a clear way on [Bear Tracks/Course Browser]*" gave a significantly higher score for the Course Browser [$W = 16$, $p < .001$]. The question, "*I felt very confident with the answers I got from [Bear Tracks/Course Browser]*" showed that participants felt significantly more confident with the information obtained with the Course Browser [$W = 31$, $p < .001$]. The next question compared the adequacy of the systems for performing course-planning tasks. For the Traditional Method, we asked "*The course catalog on Bear Tracks is adequate for performing course planning*", and for the Course Browser, we asked: *"I would like to use a tool similar to the Course Browser for my course planning*". We observed a significantly higher score for the Course Browser [$W = 6$, $p < .001$]. Finally, a significant number of participants agreed with the sentence "*The Course Browser is better than Bear Tracks for performing course-planning tasks*" [$V = 91$, $p < .001$].

4.4.2. Questions on the course browser

Course Browser's user interface In order to explain the user interface of the Course Browser to the participants, we recorded a three-and-a-half-minute screencast. It guided the user through the most important features of the application. A significant number of the participants agreed with agreed with the statement "*My understanding of the Course Browser improved significantly after watching the Quick Tour video*" [$V = 80$, $p < .01$].

Requisite Box diagrams A significant number of participants agreed or strongly agreed with the statement "*I understood the meaning of the Requisite Box Diagra*m" [$V = 105$, $p < .001$]. Similarly, significant number of participants agreed or strongly agreed with the statement "*The Requisite Box Diagram is an effective way to visualize course requisites*." [$V = 120$, $p < .001$].

Navigation Features A significant number of participants agreed or strongly agreed with the statement "*The Collections offer an effective way to group the courses in the course catalog*" [$V = 115.5$, $p < .001$]. A significant number of participants agreed or strongly agreed with the statement "*The Course List offers an effective way to browse the courses*" [$V = 94.5$, $p < .01$].

Course Requisite Graph A significant number of participants agreed or strongly agreed with the statement "*I understood the meaning of the Course Requisite Graph*" [$V = 105$, $p < .001$]. A significant number of participants agreed or strongly agreed with the statement "*The Course Requisite Graph offers an effective way to visualize the relationships between courses*" [$V = 120$, $p < .001$].

Collection overview Diagram A significant number of participants agreed or strongly agreed with the statement "*I understood the meaning of the Collection Overview diagram*" [$V = 120$, $p < .001$]. A significant number of participants agreed or strongly agreed with the statement "*The Collection Overview diagram offers an effective way to visualize the relationships between courses*" [$V = 105$, $p < .001$].

4.4.3. Discussion

Participants expressed a strong acceptance of the Course Browser and considered it superior to the Traditional Method for performing course-planning tasks. We obtained significant results in favor of the Course Browser for all questions. We also identified problems with the current Course Browser prototype and gathered important suggestions on how to improve it.

Most participants agree it was easy to find courses with both the Course Browser and the Traditional Method. However, they showed a stronger preference for the Course Browser as it allowed them to search for courses faster. Some participants suggested incorporating a search feature into the Course Browser. During the development of the prototype we were aware of the value of this feature, but could not implement it due to time constraints. A search could further improve the navigability of the Course Browser tool.

Participants also found the Course Browser adequate for performing course planning and consulting course requisites. The requisite information was easier to understand using the Course Browser. They also agreed that the visualization and navigation tools provided in our system simplified the interpretation of results.

Participants showed a higher level of confidence in the Course Browser, and they agreed on the usefulness and relevance of the different elements of the Course Browser's user interface.

Finally, participants expressed approval of the circular diagrams: a significant number of them agreed when asked if they understood their meaning of both the *collection overview* and the *course requisite graph*. They also agreed that these visualizations offered an effective way for visualizing relationships.

5. General discussion

Every dependency visualization domain has multiple ways for visualizing particular kinds of dependencies. Since each software tool in these domains has to deal with dependencies, one can define a common unified visualization framework. A dependency visualization framework should provide the ability to model and visualize a wide range of relationships between elements. For example, in an organizational chart, it is often useful to document the work relationships among coworkers in teams. These relationships crosscut the tree hierarchy and, therefore, do not fit into the simple hierarchy model. Furthermore, a person may participate in a variety of projects, and relationship visualization should depend on the desired view, for example, for project versus management reporting.

At this stage we do not fully understand how to best visualize dependencies, but the work presented here lays a basis for developing a framework that can support both static and interactive diagrams and enables us to test different visualization ideas. In general, dependencies and even simple hierarchies can be challenging for automatic layout. For this reason, the visualization framework must allow interactive manipulation. This interaction can be implemented using, for example, a rich-prospect browsing interface.

The creation of visualization diagrams requires a layout engine capable of rendering different styles of diagram. Since a purely algorithmic layout approach rarely works to complete satisfaction, it is important to permit users to specify hints and constraints. As this increases the complexity of the user interaction with the visualization, the system must have the ability to remember past hints and layouts from other users, possibly experts.

5.1. Insights on dependency visualization

The creation of dependency visualizations can be a powerful tool for understanding problems, if they are well defined. In our experience with the Course Browser, we learned valuable lessons on how to create useful visualization of dependency graphs.

Problem-driven visualization: A good starting point for creating a dependency visualization, and any generation in general, is to define a set of questions about the domain that the visualization must help answer. These questions can serve as evaluation criteria for comparing different prototypes.

Taking advantage of the particular characteristics of the problem domain: If the dependencies form a tree, one can use specific algorithms for trees. If the dependencies form a graph, it is important to check the particular graph characteristics, e.g., if they are acyclic, connected, biconnected or bipartite, and choose algorithms optimal for these types of graphs. It is also important to consider the typical number of elements that need to be visualized. Dealing with hundreds of elements requires different strategies than dealing with thousands or hundreds of thousands. If the exact number can not be defined, at least the order of magnitude should be useful.

Let the user decide what view is the best: Different views will appeal to different users. In the Course Browser study, we observed that some users preferred to use the overview diagram while others preferred to navigate the requirements using the hyperlinks in the requisite box diagram. Providing different ways to visualize information and different affordances to manipulate the views will maximize the utility of the user interface.

Preserve history: Dependency visualization is a means to an end. In the case of planning courses, the user must be able to explore "what-if" scenarios, and then make a choice. In the event that the choice is unsatisfiable, perhaps do to course timetable constraints, then they need to revisit previous scenarios, and possibly expand on them. Thus some mechanism for preserving the queries that resulted in prior dependency visualizations should be supported.

Making good use of different perceptual channels, e.g., shape, color or movement. If possible, it is advisable to use more than one of these channels to encode the same property. This allows to preserve the semantics of a diagram even if it is transferred to a different medium, e.g., from screen to paper. It may also be helpful for people with perceptual disabilities to completely understand the diagram.

Taking advantage of the media for displaying a visualization: Each medium has its advantages and limitations. For example, when using a computer screen, one might include interactive features to help the user with navigation. When using paper, the resolution can be higher than on a screen, but the content must be static and the use of color may not be feasible.

Avoiding over-simplification of information: Removing some of the complexity to provide an overview of the problem can help the users to better understand the information, as long as they are aware of the simplification. Therefore, it is important to make the simplifications as explicit as possible; otherwise, the users might take the simplified model for the real model. The fact that study subjects correctly answered realistic questions indicated that the various diagrams did not remove key items of information.

Making diagrams self explanatory: Users are unenthusiastic about reading documentation. Therefore, any diagram should give the user clues on how to interpret and manipulate it. One way to do this is to provide tooltips or rollover messages. In addition, choosing a familiar metaphor will help users to guess the possible actions.

With hindsight, we should have also tracked how often the various kinds of visualizations (e.g., collection overview, course requisite graph, and requisite box diagram) were used. Our informal observation is that it varied significantly among users, with them preferring one over the other, or using multiple views. Such a data collection feature should be a part of any production system in order to acquire usability data.

5.2. Future work

The proposed Course browser can be improved in a number of way. These improvements speak to the design principles of dependency visualization in general.

Search: One of the features that many of the participants in the user study suggested is a search box. This box would instantly filter the course information by code and title.

Custom collections: The current prototype only supports predefined collections. With custom collections, the user would be capable of defining new collections based on their needs. A custom collection would be defined by a query string, for example, users could look for courses in Physics and Math with three credits and no requisites.

Reverse dependency lookup: The current prototype of the Course Browser only allows the user to see reverse dependencies, i.e., courses that require a given course, within a collection. Sometimes it is useful to see the reverse dependencies across different collections. This can be done by extending the Course Requisite graph to explore the dependency graph in reverse.

Improve readability of circular diagrams: Some of the participants were confused by the use of different saturation levels to represent the strength of course dependency in circular diagrams. The difference between mandatory requisites and choices can be made more explicit by using different colors or combining dashed and continuous lines.

Improve scalability of circular diagrams: Currently, the circular diagrams are limited to 150 elements or less. Beyond that point, the diagrams become too cluttered to be useful. New techniques must be explored to allow the visualization of up to thousands of elements. This may be done by dynamically collapsing groups of courses into single nodes and expanding them as necessary.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council of Canada through the Discovery Grant Program to Bischof and Hoover.

References

Moreno, C. A., & Hoover, H. J. (2011). *Interactive course browser*. http://www.ualberta.ca/~jhoover/CB1/main.html Accessed 13.09.11.
Heer, J. (2009). *Flare: Data visualization for the web*. http://flare.prefuse.org/ Accessed 13.09.11.
Heer, J., Card, S. K., & Landay, J. A. (2005). Prefuse: a toolkit for interactive information visualization. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 421–430). New York, NY, USA: ACM.
Kerr, B. (2003). Thread arcs: an email thread visualization. In *Proceedings of the 2003 IEEE Symposium on Information Visualization* (pp. 27).
Krzywinski, M. (June 15, 2009). *Circos: Circularly composited genomic data and annotation imager*. http://mkweb.bcgsc.ca/circos/ Accessed 13.09.11.
Lidwell, W., Butler, J., & Holden, K. (2003). *Universal principles of design: A cross disciplinary reference*. Rockport Publishers.
Preece, J. (2002). *Interaction design*. New York: J. Wiley & Sons.
Ruecker, S. (2003). *Affordances of prospect for academic users of interpretively-tagged text collections*. Ph.D. Thesis University of Alberta.
Sarkar, M., & Brown, M. H. (1992). Graphical fisheye views of graphs. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 83–91). New York, NY, USA: ACM.
Shneiderman, B. (1992). Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics, 11*, 92–99.
Tufte, E. R. (1995). *Envisioning information* (5th ed.). Graphics Press.