CourseNavigator: Interactive Learning Path Exploration

Zhan Li∗, Olga Papaemmanouil† and Georgia Koutrika†
∗ Brandeis University, Waltham, MA, USA, † HP Labs, Palo Alto, USA
∗{zhanli,olga}@cs.brandeis.edu, †koutrika@hp.com

ABSTRACT
Course selection decision making is an extremely tedious task that needs to consider course prerequisites, degree requirements, class schedules, as well as the student’s preferences and constraints. As a result, students often make short term decisions based on imprecise information without deep understanding of the longer-term impact on their education goal and in most cases without good understanding of the alternative options. In this paper, we introduce CourseNavigator, a new course exploration service that attempts to address the course exploration challenge. Our service identifies all possible course selection options for a given academic period, referred to as learning paths, that can meet the student’s customized goals and constraints. CourseNavigator offers a suite of learning path generation algorithms designed to meet a range of course exploration end-goals such as learning paths for a given period and desired degree as well as the highest ranked paths based on user-defined ranking functions. Our techniques rely on a graph-search algorithm for enumerating candidate learning paths and employ a number of strategies (i.e., early detection of dead-end paths, limiting the exploration to strategic course selections) for improving the exploration efficiency.

1. INTRODUCTION
In a university, students are typically expected to explore and process information from numerous data sources (e.g., course catalog/schedule, course/student services web sites, etc) in order to make course registration decisions for each semester of their studies. Course selection is a complex decision making problem bound to a number of factors. Student personal educational goals involve the completion of a certain degree by graduation year (e.g., B.Sc. in CS) or strengthening their expertise in certain areas (e.g., programming). Course prerequisites dictate which courses are eligible for selection at a given point based on whether their prerequisite courses have been completed. Finally, courses are not offered always. Class schedules determine which courses are offered at certain periods. They are usually available only for two or three semesters ahead from current semester; future class schedules are not known.

Given all these factors, learning path planning, i.e., planning which courses to take in what order so as to complete an educational goal in time, becomes an extremely complex problem for students. The plethora of courses generates an exponential number of possible learning paths. Course prerequisites, educational goal requirements, and scheduling constraints designate only a subset of those as eligible or desired paths. Hand picking them is tedious and risky. As a result, students often make short-term decisions based on imprecise information without deep understanding of the longer-term impact of their decisions on their goal and in most cases without good understanding of the alternative options.

Ideally, students would like to explore various scenarios and see different possible learning paths, ranked in some meaningful way. For example, they might want to ask: “which are the shortest learning paths to a CS major if I take up to 4 courses per semester?”; “which course selections increase my future course options and number of possible paths to a CS major?”; “given my past selections, are there paths that lead to a major in the next 4 semesters?”. Hence, the learning path planning problem naturally lends itself to an interactive data exploration solution.

Unfortunately, existing tools (e.g., Rutgers Degree Navigator [2], Michigan State University Degree Navigator [1]) can only recommend courses to take next. They do not show entire learning paths and they are not suitable for planning ahead. Learn2learn [7] looks into students’ past choices and visualizes popular paths; it is not intended for interactive exploration of all possible paths based on a student’s personalized goals. Finally, personalized curriculum sequencing systems are also limited to next-step course/topic suggestions based on student performance [4, 5, 6].

In this paper, we present CourseNavigator, a new course exploration system that attempts to address these challenges. Our system identifies all possible course selection options, referred to as learning paths, that can meet the student’s customized educational goals taking into account the course prerequisites, the goal requirements, and the class scheduling constraints.

To handle the multiple constraint types of our planning problem (both time-based and course-based ones), we introduce the learning graph, a directed graph that simultaneously encodes course prerequisites and class scheduling constraints in its structure and contains a set of overlapping learning paths for a given student and enrollment status. To handle the plethora of possible paths and recommend the best paths, we design a suite of algorithms to address a number of end-goals. Specifically, CourseNavigator contains algorithms that generate all learning paths for a given academic period and educational goal as well as algorithms that generate the best plans based on a user-defined ranking function (e.g., shortest, most reliable, etc). Our techniques rely on graph search for enumerating candidate learning paths and employ a number of strategies (i.e.,
early detection of dead-end paths, limiting the exploration to stra-
gic course selections) for improving the exploration efficiency.
Our contributions are summarized as follows:

1. We formally define the learning path planning problem and
we introduce the learning graph as a conceptual structure that
embodies all the possible learning paths for a set of prereq-
isites and the class schedule.

2. We describe three algorithms for generating the learning
paths that satisfy different user goals.

3. We propose three methods for ranking learning paths, based
on time, workload and reliability of a path.

The remaining of the paper is organized as follows. Section 2
defines the learning paths exploration problem formally. Section 3
outlines the system model and main functionality of our system.
Section 4 discusses the three learning paths generation algorithms.
Section 5 presents preliminary experimental results and we con-
clude in Section 6.

2. PROBLEM DEFINITION

Next, we formally define our learning path exploration problem.

Course Information Let us define as $C$ the set of courses of-
fered to students. Each course $c_i \in C$ is described by its prereq-
isites condition $Q_i$, and the set $S_i$ of semesters it is offered. This
information is provided by the university’s registrar. The prerequi-
site condition $Q_i$ refers to the courses a student needs to complete
before enrolling to course $c_i$. Let us assume a variable $x_j$ that is
true if the course $c_j$ has been completed by the given student and
false otherwise. Then, $Q_i$ is defined as a boolean expression on
these variables, i.e., $Q_i = (x_j \land \ldots \land x_k) \lor \ldots \lor (x_m \land \ldots \land x_n)$
for any $x_j$ with $j \neq i$.

Enrollment Status At any time point a given student has a
certain enrollment status, which is described by: (a) the current
semester $s$ (b) the set of courses the student has completed by the
semester $s$ and (c) the course options that are available for the
student in that semester. This last set include courses offered in
semester $s$ whose prerequisites are satisfied by the student’s com-
mpleted courses.

Learning Path/Graph In this work, we express our various ex-
ploration problems as graph construction problems. Specifically,
we capture course selections over a number of semesters as trans-
sitions from one enrollment status to another. We represent each
enrollment status as a node in a graph and therefore course selec-
tions over a period of time are modeled as paths between nodes.
We refer to these as learning paths. Our system allows users to express
their exploration goals and uses a suite of algorithms to generate the
set of learning paths that satisfy these goals. The output learning
paths (which might be overlapping) define the learning graph.

Formally, a learning graph $G(E, V)$ is a directed graph that con-
tains a set of overlapping learning paths where a learning path
$p_i \in G$ is a set of nodes ordered by time. Each path node $n_i \in V$
represents a unique enrollment status for a semester $s_i$. For the en-
rollment status of node $n_i$ we represent the courses the student has
already completed as $X_i = \{c_j \in C\}$ and the course options as
$Y_i$. These are the courses offered in semester $s_i$ and for which the
student’s enrollment status at $n_i$ satisfies their prerequisites, i.e.,

$$Y_i = \{c_j \in C - X_i | Q_j == \text{true}, s_i \in S_j\}$$

Each path edge of the exploration graph $e(n_i, n_{i+1}) \in E$ repre-
sents a transition from enrollment status of $n_i$ to the status of $n_{i+1}$. This transition happens within a semester (i.e., $s_{i+1} = s_i + 1$)
assuming that the student enrolled to a set of courses out of the
$Y_i$ course available to him. We represents these elected courses as
$W_{i+1}$ and based on our definition it follows that $W_{i+1} \subseteq Y_i$.
Furthermore, $X_{i+1} = X_i \cup W_{i+1}$, i.e., in a given semester’s
student’s completed courses are augmented by the course selections
of the previous semester.

An example learning graph with two overlapping learning paths are
shown in Figure 1 for a period of three semesters (Fall 2011, Spring 12, Fall 2012). Here, the student starts in Fall 2011 (node $n_1$).
Both paths include the election of two courses [11A, 29A] for Fall 2011 leading to node $n_2$. For Spring 2012, one path elects courses [12B, 21B, 65A] leading to node $n_3$, while the other path elects [12B, 21B, 2A] leading to node $n_4$.

Exploration Tasks Let us assume a student who uses our system
for a course exploration task. Exploration tasks are expressed as
conditions on some future enrollment status (i.e., graph nodes). For
example, a student can request the generation of all possible course
selections given his current enrollment status and for $d$ semesters
ahead. This is equivalent to searching for the learning paths starting
from a node $n_1$ representing the student’s current enrollment status
and leading to some node(s) $n_d$ where $s_d = s_i + d$. In another
scenario a student might want to generate all possible learning paths
that guarantee the completion of a given set of interesting courses.
These are the paths starting from the current enrollment status node
$n_1$ and leading to some node(s) $n_d$ where the completed courses $X_d$
include the desired courses.

Finally our users are able to express ranking criteria on the gen-
erated learning paths. We discuss ranking functions in more detail
in Section 4.3 and examples of ranking functions include returning
the top-$k$ shortest in time paths, paths with least workload or even
the top most “reliable” paths. Here, the reliability of a path is mea-
sured based on its probability to be materialized in the future given
the uncertainty of certain course offerings.

Learning Path Generation Given the above definitions we now
define our exploration problem as a graph construction problem.
Let us assume a student with a given enrollment status at a semester
$s$ and an exploration task expressed as a condition on some future
enrollment status. Our algorithms define a start node $n_a$ as the node
that captures the student’s enrollment status at semester $s$. Given
the set of courses $c_i \in C$, the schedule for each course $S_i$ and the
prerequisites condition for each course $Q_i$, each algorithm gener-
ates all learning paths starting from $n_a$ and leading to all possible
goal nodes $n_b$ that satisfy the student’s exploration task.

3. SYSTEM OVERVIEW

Figure 2 offers an overview of our system model. Initially, the
student provides the exploration parameters through the front-end
interface. These parameters include the student’s enrollment status
and his desired exploration goal (e.g., graduation semester, a set of
desired courses), constraints (e.g., maximum number of courses to
take per semester, courses to avoid), and preferred ranking for the
output learning paths (e.g., shortest).

The dataset to be explored is generated by the back-end. Here,
the registrar provides all class and degree information which in-
cludes the class schedules, course descriptions, and degree require-
ments. Specifically, the Prerequisite Parser parses the course descriptions and generates the prerequisites condition \( Q_i \) for each course \( c_i \in C \). The Schedule Parser parses the class scheduling information and generates the schedule \( S_i \) for each course \( c_i \). Given the input from the student and the registrar the Learning Path Generator generates the learning paths. These paths are presented to the user by the Learning Path Visualizer.

4. LEARNING PATH GENERATION

Our system includes a suite of algorithms for generating graphs of three types of learning paths: (a) deadline-driven learning paths, which are all the learning paths until a given end semester, (b) goal-driven learning paths, which are learning paths that can meet the student-defined goal requirement by a given end semester and (c) ranked learning paths, which are the highest ranked goal-driven paths based on the student’s ranking preferences. In this section we discuss the three learning path generation algorithms.

4.1 Deadline-driven Learning Paths

A very common exploration task for students involves understanding their course selection options up to a certain semester. This would ideally be their graduation semester but more often than not students are interested in exploring their options only for a couple of semesters ahead. This is due to the fact that class schedules are released for only one or two semesters forward as well as that students’ goals and interests often change between semesters.

Given a student’s enrollment status in the current semester \( s \) and a target semester \( d \), our system generates all deadline-driven learning paths that the student can follow starting from semester \( s \) and ending to semester \( d \). Starting from the current semester our algorithm identifies all possible course selection options the student has in this semester. Given a selection of courses in the current semester it then generates the student’s new enrollment status for the next semester. It iteratively explores all possible course selection options for the upcoming semesters and generates all possible future enrollment status up to semester \( d \). Each new enrollment status is a new graph node and hence the algorithm generates the final learning graph. The pseudocode is shown in Algorithm 1.

Formally, our algorithm generates a learning graph that includes all paths with a start node \( n_1 \), with \( s_1 = s \) and a set of goal vertices \( N = \{n_i | s_i = d \} \). Let us assume the student has completed \( X \) courses by the current semester \( s \). Algorithm 1 initiates the exploration by creating the start node \( n_1 \) with \( X_1 = X \) and \( s_1 = s \) (line 1). We note also that this node’s status includes the set of courses \( Y_1 \) that the student can enroll to in semester \( s_1 \). This set is calculated in a straightforward way from the set of courses the students has completed so far \( X_1 \), the courses that require them as prerequisites and the schedule of these courses (line 2).

For any node \( n_i \) with out-degree equal to 0 (i.e., with unexplored course selection choices), we calculate all possible course combinations the student can enroll to out of the course options set \( Y_i \) (line 7). For each course combination \( W_{i,i+1} \subseteq Y_i \), a new edge \( e(n_i, n_{i+1}) \) is added that represents the decision to take the courses in \( W_{i,i+1} \) in semester \( s_i \). We also add the new node \( n_{i+1} \) which refers to one semester ahead of \( n_i \) (line 10) and update \( X_{i+1} \) to be the union of \( X_i \) and \( W_{i,i+1} \) (line 11). We then update \( Y_{i+1} \) to include the courses that are offered in \( s_i \) and those whose prerequisites are satisfied (line 13).

Students can optionally specify the maximum number of courses they wish to take per semester \( m \) in which case \( W_{i,i+1} \) will have a size of \( m \) or less (line 8). We repeat the exploration process for each new generated node and stop exploring further than a node \( n_i \) if \( n_i \) is in the end semester, i.e., \( s_i = d \) (line 5). We return the learning graph at line 15.

Algorithm 1 Deadline-driven Learning Paths

Input: offered courses \( C \), current semester \( s \), end semester \( d \), student’s completed courses \( X \) by \( s \)

Output: learning graph \( G(E, V) \)

1: create start node \( n_1 \) with \( s_1 = s \), \( X_1 = X \)
2: \( Y_1 = \{c_j \in C - X | Q_j = true, s \in S_j \} \)
3: \( V = V \cup \{n_1 \} \)
4: for each node \( n_i \) with out-degree = 0
5: \( s_i = d \)
6: continue
7: for each course combination \( W_{i,i+1} \) from \( Y_i 
8: if |W_{i,i+1}| > m
9: continue
10: create a node \( n_{i+1} \) and an edge \( e(n_i, n_{i+1}) \)
11: \( X_{i+1} = X_i \cup W_{i,i+1} \)
12: \( s_{i+1} = s_i + 1 \)
13: \( Y_{i+1} = \{c_j \in C - X_{i+1} | Q_j = true, s_{i+1} \in S_j \} \)
14: \( V = V \cup \{n_{i+1} \}, E = E \cup e(n_i, n_{i+1}) \)
15: return \( G(E, V) \)

Figure 3 shows an example of deadline-driven learning paths. Here we assume the offered courses \( C = \{11A, 29A, 21A\} \), courses 11A and 29A do not have any prerequisites and 21A has 11A as prerequisite. We also assume the following schedules: \( S_{11A} = S_{29A} = \{Fall’11, Fall’12\}, S_{21A} = \{Spring’12\} \). The learning graph is built with the assumption that the student wants to find all learning paths from Fall ‘11 to Spring ‘13. At start node \( n_1 \) we have two course options, 11A and 29A, because they are offered in Fall ‘11 and they don’t have prerequisites. So we generate edges to select course 11A, or 29A, or both 11A and 29A, leading to nodes \( n_2, n_3 \) and \( n_4 \). At node \( n_4 \) there are no courses option because 11A is not offered in Spring ‘12 and the prerequisite of 21A has not been satisfied. At node \( n_5 \) 21A is eligible to take because it is offered in Spring ‘12 and its prerequisite 11A is completed. At node \( n_5 \) we generate an edge that selects course 29A which leads to node \( n_6 \). At node \( n_7 \) we generate an edge that selects course 11A which leads to node \( n_8 \). We stop at node \( n_8 \) because there is no more course to take and we stop at node \( n_4 \) and \( n_9 \) because both nodes are in the end semester, i.e., \( s_8 = s_9 = \text{Spring ‘13} \).
4.2  Goal-driven Learning Paths

Apart from exploring all possible course options for upcoming semesters, students are often interested in exploring learning pathways that lead to a course-related goal. This goal could be completing the course requirements of a degree, completing a set of interesting courses, etc. For example, students might be interested in exploring course selections that will allow them to complete the requirements of a CS major or course selection that allows them to complete a given set of programming courses. We refer to this as goal-driven course exploration and we propose an algorithm that identifies these goal-driven learning paths. In our approach, the user specifies (a) his desired goal requirement as a boolean expression on the student’s enrollment status and (b) the end semester (deadline). Our approach generates a learning graph that includes paths to goal nodes that meet the exploration goal by the end semester.

Our algorithm relies on the following observation. All goal-driven paths will be a subset of the deadline-based learning paths generated by Algorithm 1 (for the same deadline). Specifically, the output set will include all the deadline-based paths that satisfy the student’s goal requirement. Hence, our approach generates the learning paths similarly to the Algorithm 1, but strives to detect as early as possible the paths that will not meet this goal. In order to detect and prune these paths we employ two path pruning strategies: the time-based strategy and the course-availability based strategy. Both approaches guarantee to identify all paths that meet the user’s exploration goal.

4.2.1  Time-based Pruning Strategy

Given the student’s current enrollment status, the student’s desired goal requirement and an end semester, this strategy tests whether there is enough time to complete goal requirement by the end semester. If there is no time, we stop exploring further.

To check this condition, we calculate the minimum number of courses, \( m_{\text{in}} \), a student needs to take in semester \( s_i \) to complete his goal requirement by end semester \( d \). Let us assume the student wants to take up to \( m \) courses per semester and a best-case scenario, where the student takes exactly \( m \) courses in each of the following semesters (except for the current semester \( s_i \)). Then, when in semester \( s_i \) there are \((d - s_i - 1)\) semesters left until the end semester. We define \( \text{left}_{s_i} \) to be the number of remaining courses needed to reach the student’s desired goal, which can be calculated using Ford-Fulkerson max-flow algorithm and was introduced in[3]. Then, the minimum number of courses that the student needs to take in semester \( s_i \) to complete goal requirement by end semester \( d \) is

\[
\text{min}_{s_i} = \text{left}_{s_i} - m(s - s_i - 1) \quad (1)
\]

If \( \text{min}_{s_i} \) is greater than the maximum number of courses the student can take per semester, \( \text{min}_{s_i} > m \), then we stop exploring the node \( n_{s_i} \). Otherwise, it’s possible to reach a goal from \( n_{s_i} \), but the student has to take at least \( \text{min}_{s_i} \) courses in semester \( s_i \).

Lemma 1. The time-based pruning strategy doesn’t prune any paths that meet the student’s goal requirement.

Proof. Let’s assume there is a path \( p_i \) that can lead to a node that satisfies the goal requirement and is eliminated by the time-based pruning strategy. We also assume that \( p_i \) is eliminated at node \( n_{s_i} \) and that the goal vertex on \( p_i \) is \( n_g \). Given that we prune at \( n_{s_i} \), it holds that \( \text{min}_{s_i} > m \) and then based on Equation 1 we have the following inequality:

\[
\text{left}_{s_i} > d - s_i \quad (2)
\]

Let’s assume the average number of courses the student takes per semester from \( n_i \) to \( n_g \) is \( n_{\text{num}} \). Given that the student can take no more than \( m \) courses per semester, we have \( n_{\text{num}} < m \). And since \( n_g \) is a goal vertex, the student must take at least \( \text{left}_{s_i} \) courses from \( n_i \) to \( n_g \), i.e., \( \text{left}_{s_i} = \lceil \frac{g - X_i}{n_{\text{num}}} \rceil \). Thus, we have

\[
\frac{\text{left}_{s_i}}{m} = \frac{|g - X_i|}{n_{\text{num}}} = s_g - s_i \quad (3)
\]

However, since \( n_g \) is a goal node it represents an enrollment status on or before the deadline, i.e., \( s_g < d \). Thus \( s_g - s_i < d - s_i \). Therefore, we can get

\[
\frac{\text{left}_{s_i}}{m} < d - s_i \quad (4)
\]

which contradicts with inequality 2. Therefore, there couldn’t exist such a path \( p_i \), that leads to a goal and is eliminated by time-based pruning strategy.

4.2.2  Course-availability based Pruning Strategy

The above strategy assumes the best case scenario for all future course enrollments. Specifically, it assumes that the student takes the highest number of courses he can/wishes to take each semester while being agnostic of the course schedule (i.e., whether or not those courses are offered on these semesters). In reality these courses might be offered. So in this strategy, we address this limitation by checking the course availability.

Given the student’s current enrollment status, the goal requirement and an end semester, this strategy checks whether there will be enough courses offered in the remaining semesters to complete the goal requirement. At the current enrollment status \( n_{s_i} \), we assume the student takes all courses offered in the remaining semesters (defined as \( C_{\text{offered}} \)) and that the enrollment status in the end semester is modeled by the node \( n_e \). If the goal requirement is not satisfied based on the completed courses \( X_e \) at node \( n_e \), we can safely conclude that even if the student takes all the remaining offered courses, he still cannot complete the goal requirement. In that case we don’t explore any paths out from \( n_{s_i} \). It is is easy to demonstrate that the course-availability based strategy does not prune any paths that meet the student’s goal requirement.

4.2.3  Goal-driven Learning Paths Algorithm

To collect the goal-driven learning paths we modify Algorithm 1 in two directions. First, when we explore a node \( n_{s_i} \) and decide whether to create new edge/nodes out from \( n_{s_i} \), apart from checking if \( n_{s_i} \) is in the end semester (line 5 in Algorithm 1), we also check if the completed courses \( X_e \) satisfy the goal requirement. If either case is true, \( n_{s_i} \) is an end node and we can stop exploring further.

Second, we detect as early as possible deadline-driven paths that cannot meet the goal requirements and stop their exploration. In Algorithm 1, before creating new edges and nodes at node \( n_{s_i} \) (line 10), we use our time-based and course-availability based pruning strategies to check if \( n_{s_i} \) could lead to any node that satisfies goal requirement. If it could, we create new edges and nodes as Algorithm 1. Otherwise, we stop expanding \( n_{s_i} \) further. This approach allows us to generate paths that meet the student’s goals while improving the efficiency of the path generation process.

Let’s take the learning paths in Figure 3 as an example, here we assume the goal requirement is to take all three courses \{11A, 21A, 29A\} and the end semester is Fall 12. At node \( n_{4} \), there is only 1 course 21A offered in the remaining semester (Spring 12). Based on course-availability pruning strategy, even if the student takes 21A, all completed courses \{29A, 21A\} in the end semester Fall 12 still cannot satisfy the goal requirement, so we prune at node
4.3 Ranked Learning Paths

In many cases, the number of generated goal-driven learning paths can be too high for the system to visualize and for the users to observe effectively. For a node \( n_i \) on the learning graph, the number of selection options is \( \sum_{i=1}^{m} \binom{|Y_i|}{k} \), where \( m \) is the maximum number of courses the student wishes to take per semester and \( Y_i \) is the set of available courses for node \( n_i \). When the user’s exploration spans long academic periods, the user’s course options increase substantially, i.e., \( |Y_i| \) becomes high, and so does the number of output learning paths.

To address this challenge, our service allows users to specify a ranking function for the goal-driven paths and generate the top-\( k \) paths based on the user’s preferred ordering. Path ranking is customizable. Our algorithm is agnostic to the specific ranking function and relies on prioritizing its exploration and focusing primarily on the most “promising” paths. Specifically, our approach assigns a cost value on each edge depending on the ranking function and based on that calculates the cost on each path. It then employs a best-first search strategy to identify the next path to explore based on the cost of each path.

4.3.1 Ranking Functions

Next we describe the three types of rankings we support.

**Time-based ranking.** Here exploration paths are ranked based on goal completion time (i.e., the length of the learning path). This allows students to find the paths that can satisfy their goal requirement within as few semesters as possible. Here we defined the cost of edge and path as follows. Each edge has a cost value of one, since each edge represents the transition from one semester to the next. Furthermore, the cost of path with end node \( n_i \) is the number of semesters from start node to node \( n_i \) (i.e., the length of the path from start node to \( n_i \)).

**Workload-based ranking.** Another supported ranking function is based on the path’s workload, i.e., the degree of difficulty of the path. Students often define difficulty based on an estimated study hours per week required by courses. This ranking function allows students to search for the “easiest” paths that can meet their goal requirement or paths whose workload does not exceed a given threshold. For this ranking function we first define \( w(c_i) \) as the workload of the course \( c_i \in C \), e.g., the number of hours students need to spend on course \( c_i \) per week (this number of often provided by student that have taken the course in the past). Then we define the cost of each edge as the sum of the workload of each course in the courses selection represented by the edge. Finally we define the cost of a path as the sum of the cost of each edge on the path.

**Reliability-based ranking.** Another useful ranking function is reliability. We define the reliability of a course in a given semester to be the probability of that course being offered that semester. Since most universities release the final schedules for only 1-2 semesters ahead, courses offered within these semesters have probability of 1.0 while for future semesters the probability is calculated based on historical schedule. A more reliable a path is, the bigger the probability that the courses on the path are offered in that semester. For this ranking function, we define the reliability of a course \( prob(c_i, s) \) as the probability of course \( c_i \in C \) being offered in semester \( s \). Then the cost of each edge can be defined as the product of the \( prob(c_i, s) \) for each course \( c_i \) in the courses selection represented by the edge. Finally we define the cost of each path as the product of the cost of each edge on the path.

4.3.2 Ranked Learning Paths Algorithm

To generate efficiently the top-\( k \) learning paths we employ a best first search approach. Specifically, we modified Algorithm 1 as follows. First, each time we generate a new node and new edge (line 10 in Algorithm 1), we calculate the cost of the new path based on user-defined ranking function. Then, for each new node (line 4), we explore first its outgoing edge with the lowest cost. If the edge ends with a goal node, we store the path from the root to that goal node as one of the top-\( k \) paths. We keep track of the number of top-\( k \) paths generated and we stop the exploration when \( k \) paths have been generated. The algorithm also uses the pruning strategies discussed in Section 4.2.1 and 4.2.2.

**Lemma 2.** For the three ranking functions defined in Section 4.3.1, our ranked learning paths algorithm outputs the top-\( k \) paths.

**Proof.** Let us assume \( p_k \) is the \( k^{th} \) ranked path included in the learning graph and \( p_m \) is the path that ranks higher than \( p_k \) based on the same ranking function but it is not included in the top-\( k \) output paths. Independently on the three ranking functions Course-Navigator supports, the subpaths of \( p_m \) must rank lower than \( p_m \) and therefore must rank higher than \( p_k \). Because each time we select the highest rank node to explore next, subpaths of \( p_m \) and hence \( p_m \) too must be generated and explored before \( p_k \). Therefore, if \( p_k \) is the \( k^{th} \) ranked path in the graph, \( p_m \) must also be included in the graph, which contradicts the assumption that \( p_m \) is not included in the top-\( k \) output paths. Hence the generated learning graph will not miss any of the top-\( k \) paths. \( \square \)

Let’s take the learning path in Figure 3 as an example. Here we assume the goal is to complete all three courses and we want to find the top-1 shortest path based on goal completion time. After creating node \( n_2, n_3, n_4 \) from start node \( n_1 \), all three paths have the same cost because the length of all paths equals to one. So we can select anyone of them. Let’s assume we expand node \( n_3 \) next. From node \( n_3 \) we create node \( n_6 \) and because \( n_6 \) is a goal node, we have found one shortest path from \( n_1 \) to \( n_6 \) and we stop the exploration process without creating the whole graph.

5. PRELIMINARY EVALUATION

In this section we present experimental results from our implementation of the three learning paths algorithms.

5.1 Experimental Setup

We implement our system in Java and experiment with an exploration data set drawn from 38 Computer Science courses offered at Brandeis University and the class schedules of the academic period ending in Fall ’15 with various start semesters. For all three path generation algorithms, we assume the student hasn’t taken any computer science courses at his initial enrollment status and the maximum number of courses he can take per semester is three. For goal-driven learning paths, we assume the goal is to get a computer science major which requires students to complete 7 core courses
and 5 elective courses and for ranked learning paths, we use time-based ranking function. All experiments were run on an Intel PowerEdge R320 server with 32GB RAM.

5.2 Experimental Results

Table 2 compares the number of output paths and the runtime of the deadline-driven and the goal-driven learning paths algorithms. The comparison is done for different academic periods starting from 4 and up until 7 semesters. We can observe that the goal-driven algorithm generates a smaller set of learning paths than the deadline-driven learning paths algorithm for the same academic period. When the exploration spans a long academic period (6 semester or higher), both algorithms’ scalability suffers. For deadline-driven algorithm, we cannot generate the learning graph for deadline-driven algorithm for more than 5 semesters because the graph is huge and we were not able to store it in memory. For goal-driven algorithm, there are more than 40 million learning paths when the exploration spans 6 semesters, which is impossible for the system to visualize effectively and for the student to understand. So both algorithms are suitable for short-term exploration.

Figure 4 shows the runtime of generating different number of shortest paths using the time-based ranking function for different academic periods. We can observe even when the exploration spans a long academic period (e.g., 8 semesters), generating 1,000 shortest paths still takes no more than 25 seconds, which provides interactive performance to the students.

Effectiveness of Pruning Strategies We also ran experiments that evaluate the effectiveness of our pruning strategies in the goal-driven learning paths algorithm. We compare the number of paths and runtime of the algorithm with and without pruning. From Table 1 we can see in average more than 99% of the paths which cannot lead to a goal are pruned early by the pruning strategies and the runtime improves more than 91% in average. Among the pruned paths, 82% of them are pruned using time-based pruning strategy and 18% are pruned by course-availability pruning strategy. The high percent of pruning is not surprising due to the fact that class schedules are designed to allow students to complete some core courses first leaving them with more elective option for the following semesters. So the output learning paths have high overlap in the first several semesters and only branch out after a certain academic period. Therefore, the learning paths that can lead to the goal nodes only account for a small part of the total learning paths and early termination of paths not leading to goals and thus reducing the graph size improves the algorithm efficiency substantially.

Comparison with Existing Learning Paths In our experiments with goal-driven learning paths algorithm, we generated all paths that evaluate the effectiveness of our pruning strategies in the goal-driven learning paths algorithm. We compare the number of output paths and the runtime of the deadline-driven and the goal-driven learning paths algorithms. The comparison is done for different academic periods starting from 4 and up until 7 semesters. We can observe that the goal-driven algorithm generates a smaller set of learning paths than the deadline-driven learning paths algorithm for the same academic period. When the exploration spans a long academic period (6 semester or higher), both algorithms’ scalability suffers. For deadline-driven algorithm, we cannot generate the learning graph for deadline-driven algorithm for more than 5 semesters because the graph is huge and we were not able to store it in memory. For goal-driven algorithm, there are more than 40 million learning paths when the exploration spans 6 semesters, which is impossible for the system to visualize effectively and for the student to understand. So both algorithms are suitable for short-term exploration.

Going forward there are a number of challenges we would like to address. First, offering a more personalized service to the students would improve the usability of our services. Incorporating more complex ranking functions, allowing for higher expressivity with respect to the goal requirements and incorporating customizable filters of the final learning paths could not only attract more users but could also reduce the size of the output paths. Finally, we plan to deploy at fully working prototype at the Computer Science Department. This will allows to better debug our service, collect and analyze usage logs and eventually build a robust, highly usable learning path exploration service.

7. REFERENCES