Personalizing education with algorithmic course selection

Tyler Morrow

PERSONALIZING EDUCATION WITH ALGORITHMIC COURSE SELECTION

by

TYLER MORROW

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER SCIENCE

2017

Approved by

Ali R. Hurson, Co-advisor
Sahra Sedigh Sarvestani, Co-advisor
Wei Jiang

ABSTRACT

The work presented in this thesis utilizes context-aware recommendation to facilitate personalized education and assist students in selecting courses (or in non-traditional curricula, topics or modules) that meet curricular requirements, leverage their skills and background, and are relevant to their interests. The original research contribution of this thesis is an algorithm that can generate a schedule of courses with consideration of a student's profile, minimization of cost, and complete adherence to institution requirements. The research problem at hand - a constrained optimization problem with potentially conflicting objectives - is solved by first identifying a minimal sets of courses a student can take to graduate and then intelligently placing the selected courses into available semesters.

The distinction between the proposed approach and related studies is in its simultaneous achievement of the following: guaranteed fulfillment of curricular requirements; applicability to both traditional and non-traditional curricula; and flexibility in nomenclature - semantics are extracted from syntax to allow the identification of relevant content, despite differences in course or topic titles from one institution to the next. The course selection algorithm presented is developed for the Pervasive Cyberinfrastructure for Personalized eLearning and Instructional Support (PERCEPOLIS), which can assist or supplement the degree planning actions of an academic advisor, with the assurance that recommended selections are always valid. With this algorithm, PERCEPOLIS can recommend the entire trajectory that a student could take to graduation, as opposed to just the next semester, and it does so with consideration of course or topic availability.

ACKNOWLEDGMENTS

I would like to express my sincerest appreciation to a number of people, without whom this work would not have been possible. The road often diverted to places some of us would not have liked, but we still made it. Firstly, thank you to my family for constant love, support, and freedom to pursue what I want. Secondly, thank you to my advisors, mentors, teachers, and co-workers who have given me considerable help and opportunities. Lastly, thank you to my friends and extended family who challenged me to be a better human-being, thank you.

My appreciation for teaching and learning is a result of my relationships with all of you, and it is this appreciation that ultimately led me to conduct the work described here. Please know that I am grateful for all that you have done.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

NOMENCLATURE

Acronyms

AOE	Area of Emphasis
CSA	Course Selection Algorithm
ER	Entity-Relationship
ESA	Entity Selection Algorithm
ILP	Integer Linear Program
LP	Linear Programming
MCH	Modular Course Hierarchy
PERCEPOLIS	Pervasive Cyberinfrastructure for Personalized Learning and Instructional Support
RS	Recommender System
SSM	Summary Schemas Model
SVG	Singular Value Decomposition
UPGMA	Unweighted Pair Group Method with Arithmetic Mean

1. INTRODUCTION

Education-applicable technology, e.g., databases, pervasive computing, computational intelligence, has long been leveraged to support learning online, in the classroom, or elsewhere [1, 2, 3]. As a result, there has been a continuous attempt in using technology to engage those who wish to learn. The prevalence of mobile computers coupled with wireless connectivity has made a vast amount of information readily available. In addition, many schools are using software tools such as Blackboard and Canvas, which help teachers to organize their courses and create content that can be easily delivered and graded [4, 5]. Moreover, recently we have been witness to efforts to publish course content for consumption by anyone with an Internet connection, free of charge [6, 7, 8, 9]. Consequently, a vast amount of content already exists, and many technological solutions are available that deliver content anytime, anywhere. In spite of all of the content delivery solutions available to us, none have replaced the classroom as the predominant environment for content delivery. This work focuses on developing ways to improve learning by designing solutions that can ensure that students and instructors are better prepared before they even set foot in the classroom.

PERCEPOLIS allows one to develop a personalized education plan that ensures intended graduation date and is tailorable to the individual's interests or mandates. The first step toward this goal has been to understand entities within the educational space, such as students, instructors, courses, and technology, among others. To this end, this research developed an ontology denoted as the modular course hierarchy (MCH). The MCH defines and classifies entities, e.g., institutions, curricula, courses, according to their attributes and relationships with other entities. By arranging these entities into layers based on their classification, a hierarchical structure is formed. A curriculum is comprised of many courses; hence the curriculum can be observed as exhibiting modularity and hierarchical attributes.

Modularity is applied to nearly all levels within the MCH and is the primary concept used to define the relationships between entities. By having layers of entities and relationships, a higher resolution of information is achieved.

When implemented for persistent storage, such as a relational database, the MCH can be used by applications to intelligently extract information, such as student interests, skills, learning styles, and performance, at any level of the MCH. One such application is PERCEPOLIS, a pervasive cyberinfrastructure for personalized learning and instructional support, originally presented in [2]. The most recent implementation of PERCEPOLIS is a web-based application that utilizes the MCH in relational database form. PERCEPOLIS is used to represent curricula (managing entities and relationships) and make context-based recommendations to both students and instructors who are identified as users within the application.

1.1. MOTIVATION

The importance of addressing educational shortcomings in order to maintain competitiveness is seen at the highest levels of government [10] [11]. In a 2009 Pew Research survey that same year, 61 percent of Americans felt that improving education was a top priority, making it the fifth-highest ranking of the 20 issues in the inquiry [12]. The same survey repeated in 2016 reported an increase in public prioritization to 66%, making education the third-highest [13]. In spite of those statistics, proficiencies in elementary and secondary education have not seen significant improvements, and college enrollment decreased by 4 percent, both according to the National Center for Education Statistics (NCES) [14]. Despite the decrease, however, the same article by the NCES projects college enrollment to increase 14 percent from 2014 to 2025. The Department of Education reports that the average length of time it is taking students to achieve a B.A. or B.S. is about 6 years at 4-year institutions [15]. Outside the classroom, advisors and instructors play a crucial role in student confidence with educational planning that can save or waste students' time and money. This is becoming increasingly more important as the cost of higher education increases. Students are better able to plan their education and reduce costs if they are able to receive timely feedback from an expert advisor, but for many students the quality and availability of such feedback is lacking [16]. This is akin to coursework were student access to rapid and corrective feedback is key to ensuring information retention [17]. Collectively, the above studies show that demand for higher education is growing, but in order to maintain or increase the quality of education institutions will have to operate more efficiently.

One option is to introduce technology which seeks to help students decide what, when, and where to learn, and with whom to do so. This could involve decision support all the way from planning courses for multiple degrees down to which content to access and in what order. If one searches recent media reports, both mainstream and student organizations still call for improvements to degree planning. In February of 2017, The New York Times reported on Georgia State's nursing program where data analysis showed that failure in introductory math correlated more strongly with long term success in the program than did failure in a core curriculum nursing course [18]. The report emphasized the expanding efforts of using predictive analytics to determine the likelihood of long-term student success. The California State University System issued a press released on efforts to expand the use of e-advising systems on their campuses [19]. Over the past 4-5 years, electronic academic planning and course scheduling have been implemented on all of their 23 campuses, while predictive analytics has been implemented on only 15. This level of adoption illustrates a high-level recognition that students can benefit from better tools if implemented properly. Still, technology to handle curriculum changes such as modified requirements or course renumbering is not available everywhere. The Ithacan reported this year that some senior students at Ithaca University are barely managing to graduate because of changes to curriculum requirements [20]. These changes led to inaccurate advising, resulting in a perceived reduction in flexibility and mistakenly taking a course that supposedly fulfills two requirements, when in actuality it did not. This thesis aims to directly address aforementioned issues.

1.2. OBJECTIVES

The objectives of the research presented in this thesis are three-fold:

1. Define the course selection problem
2. Solve the problem with a course selection algorithm
3. Personalize the course selection
4. Implement and validate the proposed algorithm

The course selection problem continues to plague students in many different institutions, so this research defines the problem in a way that makes it more widely applicable. To accomplish objective 1, the problem is defined in terms of the entities and relationships within the educational environment while maintaining simplicity and institution independence. To satisfy objective 2, the problem is modeled as an integer linear program (ILP) and solved using an established solutions. To satisfy objective 3, achieving personalization requires the integration of algorithmic behavior which favors adherence to the target student's preferences. Lastly, satisfying objective 4 requires coding of the algorithm, creation of different data sets, and analyzing the results of execution under a variety of conditions.

1.3. OUTLINE

The remainder of this thesis presents the new course selection features within PERCEPOLIS, beginning in Section 2 with a treatise on such background research as ontology, context-aware recommender systems, semantic similarity, and course selection. Section 3 contains an in-depth look at what PERCEPOLIS was envisioned to be in [2], the goal of which is to make it clear how the contributions of this thesis impact the larger project. All of this establishes a backdrop for Section 4 where the research contributions are presented and the methodology of PERCEPOLIS's course selection algorithm is discussed in detail. Once the algorithm has been presented, Section 5 presents the results of numerous case studies. Lastly, the thesis concludes with some final thoughts and proposals for future research directions in this area.

2. BACKGROUND

Development and implementation of this course selection methodology, and more broadly, PERCEPOLIS, borrows knowledge from several areas including: ontology, context-aware recommender systems, data heterogeneity, and semantic similarity. Ontology enables the unification of entities and relationships within the educational space into one formal definition and ultimately, allowing the course selection algorithm (CSA) presented in this thesis to operate on educational entities from multiple institutions. PERCEPOLIS itself is a context-aware recommender system since it considers the available context surrounding a student prior to making recommendations. Familiarity with data heterogeneity has made aware of the eventual challenges associated with merging data from different institutions. Obtaining an effective tool for semantic similarity is important to matching student interests to available courses. Lastly, previous attempts at course selection have played a crucial role in providing insight on how to begin modeling the problem, let alone to solve it. The primary goal of this section is to provide the reader with a literary basis with which they are sufficiently prepared to understand PERCEPOLIS and its underpinnings.

2.1. ONTOLOGY

An ontology can be viewed as a meta-data schema [21]. With respect to computer science, ontologies serve to represent the real world in computer programs as visualizations of a shared conceptualization. Ontology-based models by nature exhibit great formal expressiveness and capabilities for applying ontology reasoning techniques [22]. As such, they are especially useful tools for modeling context-based pervasive computing environments. Each ontology establishes the essential components of that which is being modeled, and its semantics are explicitly defined and machine-processable. By defining shared and common domain theories, ontologies help people and machines to communicate concisely, supporting semantics exchange, not just syntax [23]. Ontology-based modeling also negotiates a trade-off between expressiveness and complexity of reasoning [24]. Examples of context-aware frameworks utilizing ontologies as the underlying context model are easy to find in the literature [25, 26, 27].

A 2004 survey sought the most relevant methods of modeling contexts for ubiquitous environments [28]. In the survey, six requirements were identified within ubiquitous computing: distributed composition, partial validation, richness and quality of information, incompleteness and ambiguity, level of formality, and applicability to existing environments. The authors evaluated six different models to see which of these requirements were met. These six models are: key-value, markup scheme, graphical, object-oriented, logic-based, and ontology-based. They concluded that the most promising methods of context modeling for ubiquitous computing environments were ontology-based. Object-oriented methods were close behind, falling short in partial validation and level of formality.

2.2. CONTEXT-AWARE RECOMMENDER SYSTEMS

This section discusses the definition of context-aware recommender systems and the influence related literature has on PERCEPOLIS. [29] provides a “coherent and unified repository of recommender systems’ major concepts, theories, methodologies, trends, challenges and applications.” Recommender systems (RS’s) are introduced as software and techniques that suggest one or more items to a particular user [30]. Traditional RS’s typically rate a user’s preference towards all items and recommend the item or items that are rated the highest. RS’s are called traditional (or two-dimensional) because their recommendation process only considers the user and item. Recommendations can also be tailored to recommending sequences or groups of related items, which is directly applicable to PERCEPOLIS since it recommends course schedules which technically fall into both categories. Considerable concern is also often given to the data and availability of data used by a recommender system [31]. A lack of data is often a major challenge for recommender systems within the educational space, and PERCEPOLIS is no exception. Because of this, much of the data PERCEPOLIS needed was partially obtained by automated means and curated by hand after the fact over a long period of time.

Context, a multifaceted concept, is used by recommender systems to improve the quality of recommendations by considering the circumstances surrounding users and items, forming what are known as context-aware recommender systems (CARS) [32]. The definition of context varies greatly by field and even subfield, but it typically involves information about time, location, and relationships between users and items. As such, one will find that context-aware systems often involve mobile devices. The definition of context within PERCEPOLIS falls under a *representational* view because the set of attributes for different user types can be observed and predefined with a reasonable expectation that they will not change over time [32], a quality established by the use of ontology. The exact contextual information considered by PERCEPOLIS is discussed in Section 4.3.

The term *context-aware* is often associated in literature with mobile devices that can sense some aspect of their environment and adapt accordingly [33]. CARS that use mobile devices can easily fall into the field of pervasive computing when the context gathering is conducted in an ubiquitous fashion and recommendations are made proactively [34]. Such systems could use student location data to draw conclusions about performance by classroom, or by knowing when the student usually studies, proactively suggest a quiet study location based on the current concentration of students across campus. Within PERCEPOLIS, there is an expectation that contextual information related to courses and students be updated frequently to make sure recommendations stay valid. This can be done for students by periodically searching for updates to courses or available content, and then recomputing the potential interest a student may have in a course. On the other hand, information such as course availability must be provided by institutions as they are able to provide it.

2.3. DATA HETEROGENEITY & SEMANTIC SIMILARITY

An increase in the number of institutions using learning technology results in a wealth of institution-specific data that is distributed and heterogeneous in nature [3]. Each institution maintains its own database(s) of student records, courses, instructors, etc., the autonomy of which must be preserved. At this stage, students and their institutions must provide PERCEPOLIS with their information. Connecting to and retrieving data from the proprietary schemas of institution databases, while technologically possible and vastly more convenient, is not within the current scope of this work. Such information reveals that the same course at different institutions may have slightly dissimilar names and content, which presents a challenge when comparing students at different institutions based on their enrollment behavior. The area of multidatabase systems, the closely associated challenge of resolving data heterogeneity, is not a new area of research [35]. Any adoption of this algorithm in a system which includes data from multiple institutions will either need to use the underlying ontology-based cyberinfrastructure of PERCEPOLIS or limit its use to a single institution, which has downsides.

By identifying equivalence or similarity across institutions, more information about certain types of students is made available. Moreover, deriving similarity enables us to intelligently share content from one institution to students in another. The issue of similarity also arises when the course selection algorithm presented in this thesis attempts to match student interests to existing courses. PERCEPOLIS uses an implementation of Word2Vec, a natural language processing model developed by Google to identify words related to the student's interests, to find courses of interest [36]. PERCEPOLIS then searches course information attempting to find content that matches the related set of words.

One of the basic principles behind this method is to represent words in terms of their neighbors as vectors. Word vectors can be formed by taking counts of neighboring words and forming what is called a co-occurrence matrix, however the dimensionality of this matrix can get unreasonably large. To solve the high dimensionality issue, Singular Value

Decomposition (SVD) was used on the co-occurrence matrix to obtain word vectors with lower dimensionality. However, for millions of words or documents, the co-occurrence matrix size creates a scalability problem increasing computational cost of performing SVD quadratically. For a more recent introductory reading on deep learning in natural language processing, the interested reader is referred to [37]. Another approach to creating word vectors, which is used by Word2Vec, is to predict the surrounding words of every word, as opposed to taking the global count for an entire text corpus. This allows one to identify complex patterns in addition to word similarity.

WordNet, an English, lexical database that has been developed at Princeton since the 1980's was also considered [38]. WordNet is similar to a dictionary, but instead of storing words with respect to their spelling, it stores words based on their relationships with other words. More specifically, words which are similar in meaning are linked together with pointers of varying types forming a semantic network, i.e., an acyclic graph. These pointer types correspond to the relationship type, e.g., synonym, hypernym, hyponym, etc. Deriving a similarity metric using WordNet essentially involves the path length between two words [39].

For this research, Word2Vec was chosen for a couple of reasons: firstly, unlike WordNet, Word2Vec does not require the storage and management of a large database of syntactic and semantic relationships. Word2Vec can just keep reading the latest available text corpus to stay up-to-date with new terminology and even improve existing word vectors. Secondly, working implementations of Word2Vec that integrated easily with PERCEPOLIS were more readily available thanks to its popularity. It should be noted that WordNet is limited to its English-based taxonomy, so supporting other languages would require even more human effort.

2.4. COURSE SELECTION

The problem of selecting courses for a particular student is primarily the role of academic advisors who are experts in institutional requirements. Many researchers have inevitably attempted to automate parts of the advising process because requirements are numerous yet well-defined. Selecting courses to fulfill requirements for some degree at a specific institution is relatively straight-forward. However, designing a system which accommodates multiple degrees across institutions, let alone attempting to personalize a selection, is much more complex and shown in literature without significant or reproducible results. Nonetheless, a number of related works have contributed to better course selection by providing more insight or solving smaller pieces of the course selection problem.

Murray and Le Blanc provide one of the earliest and most notable attempts at a decision support for academic advising [40, 41, 42]. A major contribution of their work was the identification of key observations and rules for course selection. Firstly, they categorized degree requirements, and while their categorization seemed correct, doing so is not necessarily needed. If different requirements lead to different categories across different degrees, then one must carry out the burdensome task of identifying and accounting for all of them. As a result, this thesis finds that it makes more sense to define requirements only in terms of what courses can and should fulfill them. Murray and Le Blanc also apply the following rules in order:

1. Deepest Layer: the longest chain of courses connected by prerequisite relationships represents the minimum number of semesters required to graduate.
2. Maximum Dependency: courses that *unlock* more courses for enrollment should be taken earlier.
3. Lower Course Number: courses with lower course numbers should be taken earlier.

The authors state that when courses are indistinguishable after applying all three rules, then the order of selection does not matter. Unfortunately, little detail is given on the implementation and their influence on this thesis ends here.

Since the 90s, a number of systems which perform automated advising have been developed by or for specific institutions, but there is no survey which attempts to catalog them. These systems vary greatly how they accommodate institution rules and perform course selection, and only those systems which met certain criteria are discussed from this point forward. The criteria used are defined as follows:

- the course selection methodology must be discussed in enough detail to be reproducible (excluding source code);
- the course selection methodology must be adaptable to more than one institution and degree, even if that is the only use-case discussed; and
- attempts must be made to personalize course selections for students.

What is left are relatively recent attempts.

SASSY performs course selection by using a proprietary Course-Petri Net (CPN) to model courses, requirements, semesters, and availability [43]. The creators of SASSY define a Petri Net (PN) as a four-tuple (P, T, I, O) where $P = p_1, p_2, \ldots, p_i$ is a finite set of places, $T = t_1, t_2, \ldots, t_j$ is a finite set of transitions, $P \cap T = \emptyset$, $I : T \rightarrow P^{\infty}$ is the input function mapping from transitions to multi-sets of places, $O : P \rightarrow T^{\infty}$ is the output function mapping from transitions to multi-sets of places, all of which come from [44]. A CPN is then defined as a Petri Net for which P and T are expanded. P is partitioned into the sets D, R, and A where D is a finite set of degree requirements, R is a finite set of prerequisites, and A is a finite set of availabilities. Thus, $P = D \cup R \cup A$ and $D \cap R \cap A = \emptyset$. T is then partitioned into the sets S and C where S is a finite set of semester transitions and C is a finite set of course transitions. Thus, $T = S \cup C$ and $S \cap C = \emptyset$. With this additional information, SASSY attempts to produce a complete selection of courses for the remainder of the student's time in school while considering issues such as course load, time to graduation, and course preferences specified by the student. SASSY and PERCEPOLIS agree on the intuitiveness of a graph-based solution to the problem since it naturally follows from the prerequisite relationships between courses.

However, the two systems differ primarily in scope with SASSY's considerations being a subset of those found in PERCEPOLIS leading to a greater degree of personalization with respect to performance and implicit interest in courses. Moreover, SASSY does not attempt to handle the modularization of courses as PERCEPOLIS does to reduce the time to graduation.

While Section 4 discusses the CSA in detail, an important topic with which to be familiar is linear programming, a classical computer science problem. A linear program (LP) involves the minimization or maximization of a linear function with respect to a set of linear equalities or linear inequalities [45]. Given a set of real numbers $a_1, a_2, \ldots, a_n$ and a set of variables $x_1, x_2, \ldots, x_n$, the general LP has a linear *objective function* of the following form [45]:

$$f(x_1, x_2, \ldots, x_n) = a_1x_1 + a_2x_2 + \cdots + a_nx_n = \sum_{j=1}^{n} a_jx_j.$$

Alternatively, the above definition is sometimes found in a more complex matrix representation. It is the objective function for which the minimum or maximum value is sought, and in practice constraints are applied which limit possible values. These constraints are represented as linear equalities or inequalities depending on the form being used. The problem becomes an integer linear program (ILP) when the variables $x_1, x_2, \ldots, x_n$ may only take integer values. The most common methods used to solve linear programs are simplex-based and interior points, and for ILP's something like COIN-OR CBC is used [46]. This research identifies that course selection consists of different linear programming problems, examples of which are requirement satisfaction and semester satisfaction. In the case of requirement satisfaction, the goal is to minimize the number of credit hours while ensuring that the right courses are used. For semester satisfaction, the goal is to keeping credit hours within some range while preserving requisite constraints.

IDiSC+ expands prior work and SASSY by considering more institution rules and creating a variety of selections from which a student must choose [47]. IDiSC+, similar to SASSY, formalizes its course selection methodology, except it uses integer linear programming. The overall process consists of three sequential phases: modeling, exploration, and consolidation. Constraints, such as desired courses, availability, credits per semester, courses per semester, GPA per semester, budget per semester, and others, are first set by the institution or student in the modeling phase. All of these constraints are input to an objective-function which provides a value of optimality. The exploration phase involves providing these constraints to third-party software tool that searches for "optimal" solutions, each with their own objective-function value. A "diverse" set of course plans for the student to choose from is created by running the software tool with slight variations on the constraints. The final phase of *consolidation* is where the student makes a decision or explores *what-if* scenarios. Note that the terms "course plan" or "study plan" are used in [47] while "course selection" is used in this thesis. The author primarily focuses on defining the constraints and discussing how to properly tune them for diverse, yet valid, solutions. The constraints that institutions and students may set in IDiSC+ are comparable to those in PERCEPOLIS, however the following differences should be noted: firstly, the separation of course requirements and the courses which can satisfy them is not resolved. Any CSA must guarantee adherence to certain institution rules and only after that provide a measure of optimality with respect to less strict constraints. Secondly, there is no mechanism by which IDiSC+ attempts to personalize the schedule for students whose interests are unknown. Another key difference is that students' grades only affect the selection of electives, whereas PERCEPOLIS uses grade information for both course selection as well as location in the schedule. As a result, IDiSC+ attempts to distribute courses within a schedule normally. The final notable difference is that individual semester difficulty is never considered by IDiSC+. Differences between the two systems, of which there are more, represent fundamental divergences in how to philosophically model and solve the problem.

2.5. CONCLUSION

This section laid out the related works that lend to the work presented in this thesis. Each section provides discussion primarily on only those works that have had direct influence. Note that in a small number of cases, a related work was not directly influential, but it is still discussed to give the reader awareness of particularly important works in the area. Additionally, concepts which are key to the problems being solved were outlined with a citation which covers the subject in significantly more depth. Ontology is the first area discussed, and its influence on this thesis is the establishment of a shared conceptualization of the existing educational space. This has affected nearly everything from terminology to the PERCEPOLIS implementation (database design, CSA, etc.). Context-aware recommender systems were then discussed at a high level because while the focus of this thesis is predominantly about the CSA, PERCEPOLIS in its entirety *is* a context-aware recommender system. As such, idiomatic terms and techniques of the field need to be established. The third section discussed data heterogeneity and work that has aided in connecting student interests to available options based on semantic similarity. The last section is most important as it covers specific attempts at course selection that have educated the design and implementation of the CSA. Altogether, the reader should come away from this section with a foundation sufficient enough to understand the course selection problem and how the CSA within PERCEPOLIS attempts to solve it.

3. PERCEPOLIS

This section presents an overview of PERCEPOLIS, a Pervasive Cyberinfrastructure for Personalized Learning and Instructional Support [2]. PERCEPOLIS is an educational platform developed to support personalized learning and networked curricular models in higher education. PERCEPOLIS carries out its functions by leveraging pervasive computing and communication through the use of intelligent software agents. These software agents consider all available contextual information from institution information and student profiles to customize courses. The goal of this section is to give the reader insight into the larger project (as described in [2]) such that this research's contribution can be better appreciated. This section discusses the design and anticipated outcomes of PERCEPOLIS in two distinct sections. The first section on the design of PERCEPOLIS covers modular course development, blended instruction and learning, intelligent access tools, personalization of course content, and tracking of student progress. The second and final section on anticipated outcomes describes what the expected effects are on students and classrooms if PERCEPOLIS is employed.

3.1. DESIGN

The PERCEPOLIS platform views a degree-granting program as three sets of entities: 1) the set of instructors/advisors, I; 2) the set of students, S; and 3) the set of courses, C. An instructor/advisor, $i \in I$, has expertise in one or more subjects. A student, $s \in S$, is studying towards a degree and is required to take courses from C, in an orderly fashion, to satisfy degree requirements and objectives. Each $c \in C$ represents a course in the curriculum. The courses in C are interrelated, per the structure of the curriculum. In addition, courses are tagged as required or elective, according to degree requirements. Each of I, S, and C, respectively, is represented by a community of software agents that communicate and negotiate with each other according to the defined tasks, e.g., personalizing the trajectory through a course or curriculum. A modular approach is used in the development and delivery of courses, so that each course $c \in C$ can be viewed as a collection of interrelated modules, where the degree of connectivity among modules can vary from course to course.

3.1.1. Modular Course Development. Fundamental to PERCEPOLIS is the modular approach to course development. Each course is divided into several content modules, a number of which are mandatory, as dictated by the curriculum to satisfy degree objectives, requirements, and accreditation requirements, e.g., ABET. The remaining content modules, typically smaller in number, include elective content that can be chosen to supplement the student's knowledge of prerequisites, or to engage an interested student in more advanced topics. The content modules for each course can be supplemented by *experiential* modules, which are intended to reemphasize key issues covered in the content modules through hands-on individual and group projects. The experiential modules are dynamic, as their contents can be determined based on topics that a majority of the students in a given class find challenging or confusing. Associated with each module are learning objects such as prerequisite modules, lecture notes, problems, sample solutions, and programming or laboratory exercises. Course modules are self-contained, self-paced, and designed to promote active rather than passive learning.

All courses are "mix-and-match," in the sense that each consists of several modules on interrelated topics which are, in turn, interrelated across the whole curriculum in a networked fashion. Some modules require prior knowledge; if the student does not have this background, he/she should be able to peruse a module from an earlier course before reentering the current module. Assessments that gauge the student's mastery of concepts at the module level are used to allow self-paced progression through the course. If a student has already mastered the material in a module and wishes to explore the topic in more depth, he/she can choose a more advanced module in that subject. To maintain a sense of community, advanced students who progress through the course more rapidly are encouraged to serve as group leaders. This peer learning approach fosters "soft" skills, such as communication and negotiation. Each module reinforces background knowledge and allows self-paced progression and options to study more advanced concepts and/or to gain missing basics.

In short, modular course design offers:

- Adaptability/flexibility: to a certain extent, course contents can be personalized based on a student's needs and interests. This is particularly beneficial to students in honors or certificate programs, graduate students, and transfer students.
- Continuity: the modular approach to the course design, combined with the ability to personalize the contents of a course allows greater continuity in the curriculum, while satisfying the needs of some students without sacrificing the needs of others.

3.1.2. Blended Instruction and Learning. Common experience shows that online availability, e.g., through a course management system such as Blackboard, of lecture notes and other instructional content generally leads to low class attendance by students. Even students who continue to regularly attend classes may become inattentive in the classroom and passive in any discussions that take place. Furthermore, there is an increasing expectation that the course coverage and examination must be limited to whatever is available online, and any topics discussed outside of class should not be covered on the tests. Periodic examinations and pop quizzes (though extremely unpopular among the students), to some degree, could be reasonable solutions; but are becoming scarce due to large class sizes. This research seeks to adopt a new teaching practice that is attuned to advances in technology, is interactive and motivates active class participation, enforces continuous learning, and fosters interpersonal skills that may be neglected in a traditional classroom. To achieve these objectives, PERCEPOLIS modifies the traditional lecture-based teaching environment by utilizing a blended approach to learning and instruction as follows:

- Most, if not all, of the class period is dedicated to interactive problem-solving based on course material covered by modules discussed in current and previous classes.

- Students are required to peruse specified learning objects before class. All content modules are made available online in advance, and group study (facilitated by the experimental modules) is highly encouraged.
- Students are expected to either communicate major issues with the instructor ahead of time, bring their questions to the class, and/or use online communication tools such as discussion boards to share their concerns and questions with their peers.
- The class period then is dynamically organized to address major issues raised by students and enhance their understanding of subject matter learned in the content module.
- To encourage independent learning and reflection and continuous involvement, at the beginning of each class period, students are quizzed on the major conceptual topics (mandatory modules) they have been asked to study. The remainder of the session is devoted to addressing students' questions, and discussing and analyzing the more complex topics covered in the mandatory modules.
- Traditional periodic homework assignments, projects, and examinations will continue to be utilized to reinforce learning and determine the students' overall mastery of course contents. Naturally, implementation details such as absences due to extenuating circumstances need to be addressed in a fair and appropriate fashion.
- Maintain a sense of community, advanced students who progress through the course more rapidly will serve as group leaders and are rewarded with extra credit for their efforts in mentoring and tutoring their peers.
- Finally, provisions are made to encourage, promote, and reward group activities and missing basics such as leadership, communication skills, and ethics.

In short, this perspective of the classroom promotes experimentation and discovery, and allows more time to address key issues in mandatory content modules.

3.1.3. Intelligent Access Tools. PERCEPOLIS leverages research in mobile agent technology, multidatabases, pervasive computing, and mobile data access systems to develop middleware that serves as global information sharing cyberinfrastructure. This middleware is positioned on top of existing database and course management systems, and allows anytime, anywhere, intelligent, and transparent access to content modules that comprise the courses of a curriculum. PERCEPOLIS research within the scope of global information sharing, multidatabases, and multimedia data processing has resulted in the design and implementation of an intelligent search engine, denoted as the Summary Schemas Model (SSM) [35], that supports automatic identification of semantically similar/dissimilar data that have different/same names and representations in a wired or wireless platform. The model uses word relationships, i.e., hypernym, hyponym, synonym, defined in a standard thesaurus such as Roget's, to automatically build hierarchical metadata of local access terms exported from underlying local databases. The embedded dynamic nature and bottom-up design approach of the SSM makes it particularly suitable for extending the scope of PERCEPOLIS beyond a single discipline or educational unit. The summary schemas concept underlies a semantic network and semantic metadata that integrate course modules, and hence courses, within and across multiple curricula.

Furthermore, with the aid of the navigational tools and imprecise content-based search capability of the SSM, users, e.g., students, instructors, and advisors, can browse through content modules from a range of courses and curricula to determine an appropriate learning trajectory. The choice of the SSM, regardless of one's familiarity with the concept and access to its prototype, stems from: 1) its ability to perform semantic search and imprecise queries, 2) the relatively small size of its metadata, 3) the robustness of the model, 4) its automatic generation of metadata, and finally, 5) the efficiency of the platform.

In the context of PERCEPOLIS, the SSM metadata hierarchy, at the root level, represents the curriculum, e.g., computer science and computer engineering curricula, with semantic terms that specify curriculum objectives and content requirements. The leaf nodes then represent the course modules. The synonym, hypernym, and hyponym links are used to relate modules to courses and/or areas, as defined by the curriculum.

The metadata structure of SSM can be built in either a centralized or distributed fashion on top of the content and experiential modules. The relational data model is expected to be used to model course modules. As a security measure, provisions are made to provide different access capabilities to different classes of users, e.g., students, administrators, or course developers, based on their individual profiles. In addition, PERCEPOLIS has an interactive, dynamic, and user-friendly graphical user interface (GUI) to facilitate user access to the system.

3.1.4. Personalization of Course Content and Tracking of Student Progress. PERCEPOLIS is extended into the areas of pervasive computing and mobile agent technology to create a virtual environment that supports a networked curricular model and a subject-based one-to-one student-faculty ratio for the purpose of mentoring, advising, guiding, and educating students within a program. As described earlier, a student agent, in communication with the agent of his/her academic advisor, in concert with his/her accumulated academic background, interests, and special needs, determines a personalized trajectory of courses and/or modules for the student. In this process, the SSM is consulted as a backbone data source to determine the personalized trajectory, as well as a contact course advisor (naturally, in most cases this would be the designated course instructor or course supervisor). At this point, various communication means are set up for the duration of the semester, to facilitate interaction between the student and instructor/advisor.

An agent is associated with each module. The agent is also linked to other agents (modules) in the same course, as well as related modules in other courses. The agents representing various modules within a course form a community among themselves, representing that course. The agent associated with a module reports directly to the instructor, but also serves as a virtual guide to the student for the duration of the course.

More specifically, when a student, s, is required to take a course, c, an agent, D_{sc}, is created to represent the student in that course. D_{sc} acquires information about the student's profile and determines the student's academic background, interests, and accumulated prerequisite knowledge, among other required information. D_{sc} interacts with the agents of course modules to identify a personalized trajectory of modules and learning objects for the student. Furthermore, D_{sc} ensures that the student peruses required material-notes, sample programs, exercises, and the like. D_{sc} also alerts the student to timelines, class schedules, learning/discussion schedules, project deadlines, appointments with the instructor, and corresponding preparations. The instructor agent for the course, D_{sc}, ensures that the student meets all requirements for each mandatory module, and collaborates with D_{sc} to ensure that the student is supplied with all required course material. D_{sc} also informs the instructor about the progress of the student and alerts the instructor whenever the student is progressing too slowly or too rapidly.

3.2. ANTICIPATED OUTCOMES

PERCEPOLIS aims to facilitate all four stages of cognitive apprenticeship:

1. Modeling: enabling students to observe an expert in action.
2. Scaffolding: provide support for students to emulate expert performance.
3. Coaching: providing students with deliberate, planned feedback that guides their performance from novice- to expert-level.
4. Fading: removing the scaffolding as students become more competent.

In the modeling stage, multimedia demonstrations and animations can be incorporated into the course modules as representations of expert performance. Solved problems, design examples, and guided exercises with "hints" are among the learning objects useful in the scaffolding stage. Online assessments, some of which are graded instantaneously, are examples of learning objects that facilitate coaching. The fine-grained data collection enabled by the cyberinfrastructure facilitates the provision of more accurate and detailed feedback to the students, which is especially helpful in the coaching stage. Graceful and gradual fading of instructor support is also supported, e.g., through learning objects that feature advanced problems or exercises. The availability of online learning objects that can be perused at a student's convenience provides subtle assistance that encourages and supports efforts in the absence of the instructor, further facilitating fading.

PERCEPOLIS enables blended learning, where online, computer-mediated instruction supplements traditional classrooms. Students are expected to peruse before class learning objects pertaining to concepts traditionally taught during the lecture portion of a course. This allows class time to be used for problem-based learning and other educational approaches conducive to the cognitive apprenticeship. As an example, the students can be required to peruse a learning object that presents the general steps of a design technique prior to class. In-class demonstrations of the technique on an actual design problem by the instructor or a guest expert can serve as a model for the students. Alternatively, they can work on a design problem in class (scaffolding) and receive feedback (coaching). Gradually increasing the sophistication of the in-class activities supports fading.

The flexibility of the proposed cyberinfrastructure in utilizing computing technology to support rich educational content has the added benefit of facilitating the accommodation of differences in learning styles. A traditional classroom setting mainly offers visual and auditory content; print- or video-based distance study does the same. The ability to provide tactile content, whether in reality, in a classroom setting, or through remote access and virtual reality, is especially helpful in the modeling stage of the cognitive apprenticeship, but can also be exploited to increase the efficacy of other stages. Furthermore, the predominant learning style of a student can be assessed and used in personalizing courses and curricular trajectories for each student. An intelligent access tool can allow these resources to be efficiently and dynamically incorporated into classroom discussions and linked to electronic learning objects available for perusal by the students outside of class.

The self-paced and modular nature of the courses and the "anytime, anywhere" availability of the learning objects make the platform well-suited to eLearning, distance education, outreach activities, and workforce education or certificate programs. The same attributes allow graduate students engaged in interdisciplinary research to acquire the requisite broad background without having to complete traditional undergraduate level courses. The software and content artifacts of PERCEPOLIS serve as effective advising tools that can facilitate student course scheduling. Finally, the special accommodations needed for students with learning disabilities or physical challenges are aspects of personalization facilitated by the cyberinfrastructure. Outside of academia, the intelligent personalization facilitated by PERCEPOLIS can be useful to streamlining professional training, whether technical or related to topics such as safety, workplace conduct, or cultural sensitivity.

Beyond the fundamental objective of personalizing the learning trajectory of a student through a networked curriculum, PERCEPOLIS aims to utilize teaching tools, animation techniques, and remote access to information to present the same information in different ways and accommodate differences in learning styles; allow self-pacing, privacy, and flexibility; achieve spatial and temporal uniformity in presentation of the same material; and ensure efficient utilization of resources and lower costs to students and providers. The following outcomes are anticipated as a result of successful adoption of the proposed cyberinfrastructure:

1. Creation of a dynamic, student-sensitive, and adaptable blended learning environment that allows a virtual one-to-one instructor-to-student ratio.
2. Definition of a novel methodology for modular course development, implementation, and delivery that promotes continuous, self-paced, active, and peer learning and supports networked curricula.
3. Development of a novel navigational tool for student advising and course management that facilitates access, browsing, development, expansion, and modification of curricula and courses across multiple disciplines and institutions.
4. Development of a new technology-based methodology for effective and efficient presentation of courses and active engagement of students in the classroom.
5. A shift to professional practice as the focus of educational efforts, rather than technical skills.

The novelty of PERCEPOLIS lies in its ability to leverage pervasive and ubiquitous computing and communication through the use of intelligent software agents that use a student's academic profile and interests, as well as supplemental information such as his/her learning style, to customize the content of a course for the student. Furthermore, PERCEPOLIS facilitates the collection of data on student performance and learning at a resolution that far exceeds what is currently available. Knowledge discovery from this rich data set can yield invaluable insights, such as the efficacy of particular instructional techniques and strategies embedded in the learning objects. The data collected is also helpful for accreditation self-studies. Learning objects, in particular assessments can be linked to specific learning outcomes of a course.

3.3. CONCLUSION

This section discussed PERCEPOLIS in order to give the reader a better understanding of how this research's efforts on course selection fit into the overarching project. PERCEPOLIS supports a shift from teaching technical skills to fostering the professional expertise required for a student to succeed as a practicing engineer. The dynamic nature of PERCEPOLIS lends itself to a wide range of disciplines beyond engineering. The self-paced and modular nature of the courses and the "anytime, anywhere" availability of the learning objects make the platform well-suited to eLearning, distance education, outreach activities, and workforce education or certificate programs. The software and content artifacts of PERCEPOLIS serve as effective advising tools that can facilitate student course scheduling. Finally, the special accommodations needed for students with learning disabilities or physical challenges are aspects of personalization facilitated by the cyberinfrastructure.

4. METHODOLOGY

This section describes the course selection algorithm (CSA) used by PERCEPOLIS to generate personalized schedules for a given student, as well as key conceptual underpinnings which may clarify or override certain designs decisions mentioned in Section 3. The CSA itself consists of four stages:

1. Data Collection
2. Schedule Creation
    (a) Curriculum/Requirement Satisfaction (Course Selection)
    (b) Course Availability Satisfaction (Semester Selection)
    (c) Course Satisfaction (Module Selection)
3. Schedule Enumeration
4. Schedule Presentation

This section is organized into six subsections that cover this thesis' contribution to PERCEPOLIS. The first two subsections cover conceptual underpinnings of the CSA: the MCH ontology and the course selection problem. The subsequent four subsections discuss the four stages of the CSA: data, schedule creation, schedule enumeration, and schedule presentation.

4.1. THE MODULAR COURSE HIERARCHY

This section articulates the modular course hierarchy (MCH) by examining the entities and relationships it contains. This hierarchy provides both the high-level definitions and the conceptual introduction to a data layer from which information is queried. As an ontology, the MCH is, in general, neutral in terms of how it is implemented, however the work presented in this thesis chose to define the database in both an object-oriented model and relational model.

4.1.1. Entities. It is the intention of this work to promote progressive improvements to current educational practices in order to help students become better educated and prepared for professional practice. To this end, the initial design of PERCEPOLIS involves interpreting the traditional educational space as a familiar ontology and then removing redundancies or adding useful aspects. An ontology originating from a traditional model was chosen because it is more easily understood and adoptable by both instructors and students. The ontology developed for PERCEPOLIS contains both a modular course hierarchy (MCH) and the user entities who interact with entities in the MCH. The first step in building this ontology is to establish the existence of *users* who are outside entities that have discretionary interaction with the MCH entities. A user bears simple attributes such as a unique identifier, a name, and contact information. *Instructor*, *advisor*, and *student* entities are users who exhibit specific behavior by having certain relationships with MCH entities, as opposed to being defined by some unique attributes. This distinction allows a user to play multiple roles in the system, i.e., a user can be both a student and an instructor. The relationships between entities are discussed in the next section.

The MCH is comprised of six entity types: institutions, curricula, courses, modules (topics), subtopics, and content. The levels of the MCH and their corresponding entity are shown in Table 4.1 such that the level represents depth. Each entity contains certain metadata, such as a unique identifier (identifiers are unique among entities of the same type) and name. For the purposes of establishing the hierarchy, this is the only information needed. However, a description attribute is also present. Moreover, content entities are defined as either consumable or evaluable. Examples of consumable content are lecture videos, plain text, and slides. Examples of evaluable content are quizzes and problem sets. Most of the entities are familiar concepts from traditional education, with the exception of modules and subtopics which are not usually considered. Previous work on PERCEPOLIS focused on splitting courses into their different topics (denoted as modules) for added granularity. This idea was then applied to all entities within the MCH, with the exception of content entities, which are at the lowest level (finest grain) of the hierarchy. By themselves, the entities do not form the MCH. The relationships between entities must also be considered, and it is with these relationships that the hierarchical structure is created.

Table 4.1. Modular Course Hierarchy Entities

Level	**Entity**	**Description**
1	Institution	Typically an organization that offers curricula, the management of which is carried out by administrators or instructors.
2	Curriculum	A set of requirements a student must satisfy, by completing certain courses, in order to receive a specific degree.
3	Course	A set of required or elective modules (i.e., topics).
4	Module	A set of required or elective subtopics.
5	Subtopic	A set of required or elective content.
6	Content	Individual items that a student can interact with to learn about a particular subtopic.

4.1.2. Relationships. MCH relationships are governed by the following rules:

1. Relationships can only exist between two entities of the same level or neighboring level; and
2. Only one type of relationship may exist between two distinct entities.

These rules provide limitations on complexity by drastically reducing the number of possible relationships in the MCH. Based on the aforementioned rules, a single entity is allowed to have relationships with multiple entities, and the nature of a relationship depends on the entities involved. Figure 4.1 provides a partial, conceptual visualization of an MCH, showing the entity types organized into their layers (as seen in Table 4.1) and connected by relationships depicted as lines. A complete visualization of a well-populated MCH can be quite large, and it is for this reason that Figure 4.1 visually implies additional entities and relationships with ellipses.

In Figure 4.1, two curricula, D_1 and D_2, are shown as being associated with institution I_1. Solid, undirected connections from an entity in a given level to entities in the next level below (sub-entities) are considered *containing* relationships, which are usually used for association. In addition to the relationships between levels, relationships between entities within the same level are also shown: C_1 to C_2; C_2 to C_3; M_1 to M_2; M_2 to M_3; and M_3 to M_4. Relationships between entities of the same type come in three different forms: prerequisite (striped and directed), corequisite (striped and undirected), and *consequisite* (solid and directed). Prerequisite relationships establish that some course (the postrequisite) must taken at some point after another course has been completed (the prerequisite). Corequisite relationships establish that a course must be taken in the same semester as another course. *Consequisite* (a term used by this thesis for convenience) is a special type of prerequisite relationship where the postrequisite course must by taken in the next semester where it is available. *Consequisites* are sometimes seen in courses such as senior design which can span two semesters.

A traditional educational hierarchy would not usually recognize the existence of modules and subtopics as individual entities. Introducing modularity to courses creates both a simple construct for course design and more course information. This increased resolution and potential for meaningful relationships creates a wide variety of new recommendations for PERCEPOLIS to make. Conceptually, any lower levels of the MCH are an added level of detail for all of the related entities that exist in levels above. To give the reader an idea of the MCH implementation complexity, Figure 4.2 provides a database entity relationship (ER) Diagram of only the top three levels of the MCH with attributes excluded. A representation of what attributes to expect can found later in Section 4.3.

Aside from establishing the MCH, relationships between entities are the mechanisms by which PERCEPOLIS derives contextual information. In addition to the relationships between entities within the MCH, additional relationships exist between users and the different layers of the MCH. Instructors tend to create the relationships between entities within the MCH, while students tend to create relationships between themselves and the MCH. Table 4.2 lists some of the relationships that can appear between entities and what they represent. Table 4.3 then provides a description for each of the relationship types shown in the last column of Table 4.2. Systems are free to incorporate only the relation-

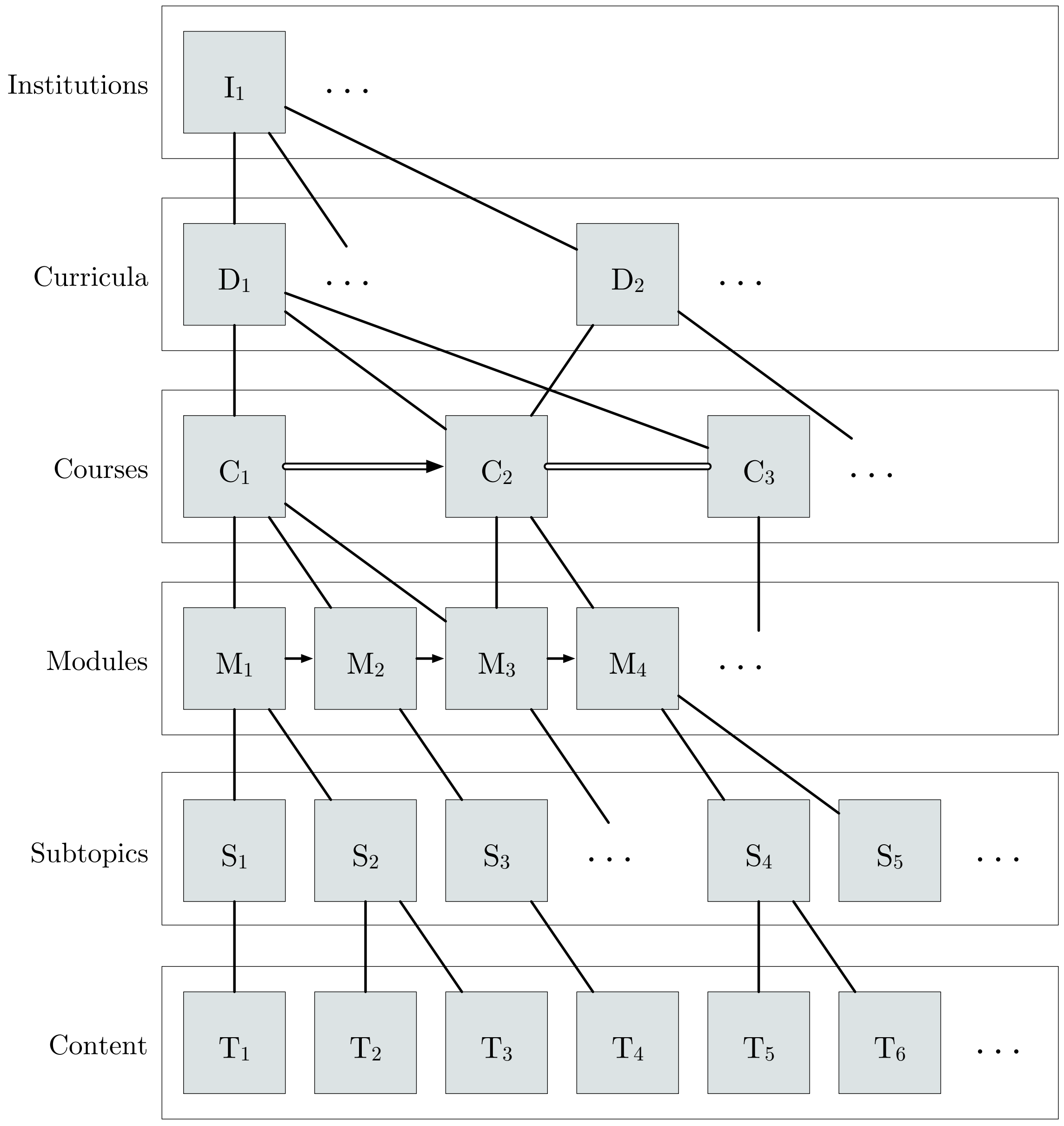

Figure 4.1. Example of a Modular Course Hierarchy

ships they may find useful, as is the case with PERCEPOLIS which does not recognize all possible relationships. The absence of a physical connection between two entities does not mean that they are not related to one another via other relationships, i.e., relatedness can be inferred from non-direct connections. All relationships imply a pairing of the unique identifiers between the two entities involved and any additional attributes specific to the relationship.

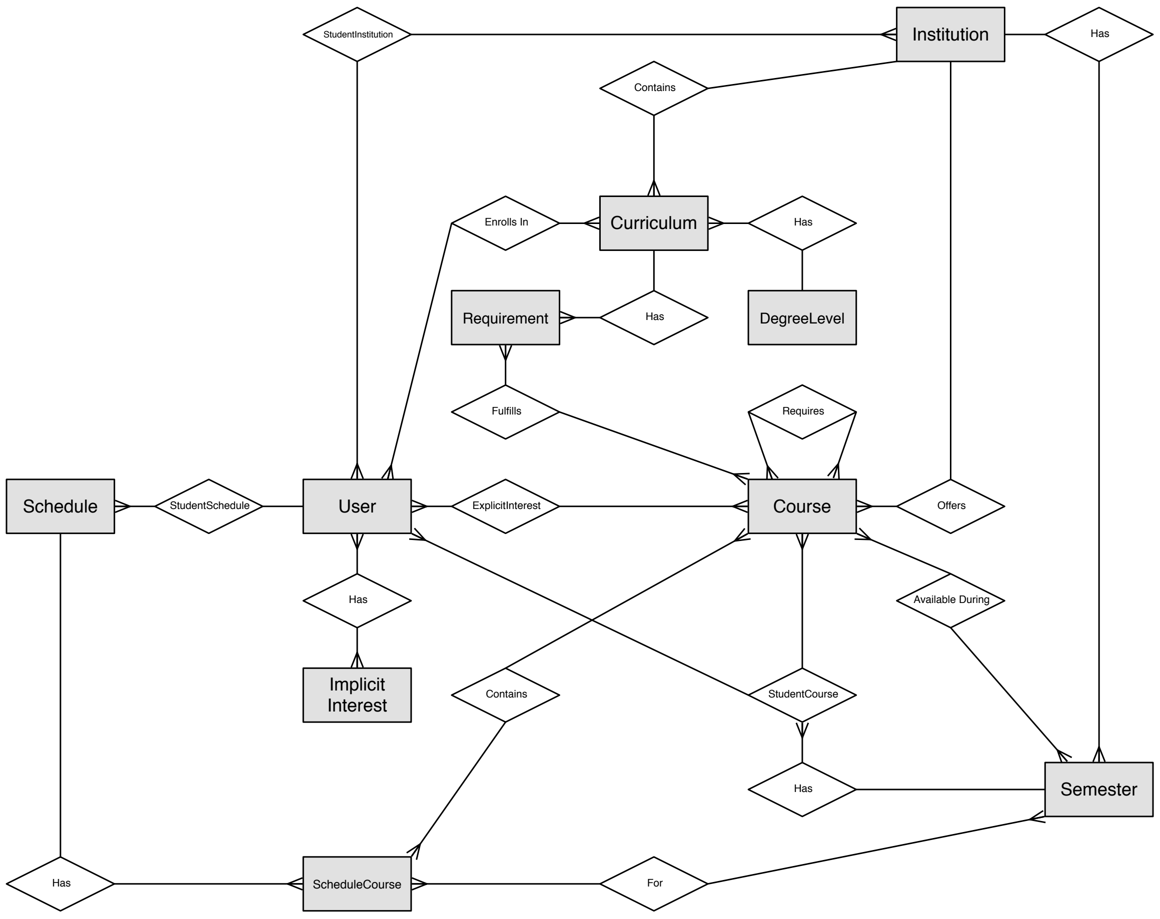

Figure 4.2. Database ER Diagram of MCH Levels 1-3

Some context is determined from entities alone, but the most useful context comes from the relationships between entities. With this in mind, recommendations can be viewed as a potential relationship. Identifying and validating potential relationships, in general, is simple. The implementing system must simply identify two entities between which a recognized (Table 4.2) relationship could exist without violating the MCH governing rules. As such, the number of relationships could become very large. It then becomes necessary to ascertain which potential relationships are more favorable and this is done by narrowing and prioritizing them through analysis of available and pertinent context.

In theory, a new student entity with no existing relationships is equally likely to establish a relationship with any other recognized entity, but in practice, that is not usually the case. One way to narrow down institution choices for the student is to ask them

Table 4.2. Example Entity Relationships

Level	Entities			Relationship
1	Institution	to	Curriculum	Association
2	Curriculum	to	Course	Requisition
3	Course	to	Course	Requisition, Ordering
	Course	to	Module	Requisition
4	Module	to	Module	Requisition, Ordering
	Module	to	Subtopic	Requisition
5	Subtopic	to	Subtopic	Requisition, Ordering
	Subtopic	to	Content	Requisition
6	Content	to	Content	Requisition, Order
All	Student	to	Student	Group
	Student	to	Institution	Association
	Student	to	Curriculum	Enrollment, Performance
	Student	to	Course	Enrollment, Performance
	Student	to	Module	Enrollment, Performance
	Student	to	Subtopic	Enrollment, Performance
	Student	to	Content	Enrollment, Performance

their interests and then give them a list of institutions most associated with those interests. Association with those interests can be determined by the number and depth of paths to sub-entities that are related semantically to the student's interests. It is very likely, however, that the institution with which the student will associate is predetermined. When a student associates with an institution, their concern becomes finding a curriculum, then a course schedule, and finally a study plan for specific courses.

Relationships between students and the lowest levels of the hierarchy provide the greatest insight into students' interests and performance. Aggregation of relationships at lower levels of the MCH can be used to quickly inform instructors about collective or individual student understanding at high granularity levels. Students can choose from existing subtopics in the system (direct matches) or provide interests which may or may not be semantically matched to existing subtopics. Direct or close matches carry immediate influence in recommendations while distant mismatches result in new subtopics that are waiting to be associated with modules. At first glance, disconnected subtopics (distant mismatches) can seem trivial, but they are very important to the development of different types of recommendations. Student interests that do not match subtopics can be interpreted as suggestions, which in turn can serve to generate recommendations to instructors. If an institution is planning to add new courses or modules, they can identify candidates by checking the unsatisfied interests of the students.

Table 4.3. Entity Relationship Type Descriptions

Relationship	**Description**
Association	A connection between two entities that establishes membership, ownership, inclusion. E.g., an institution owns a curriculum and a student is an undergraduate at an institution.
Enrollment	An obligation to participate in or consume an entity, one that typically involves learning or payment.
Group	An explicit desire or mandate to keep two entities together in certain situations, such as two students who wish to take courses together as much as possible.
Order	A restriction that under certain conditions, an interaction with one entity must be immediately followed by some other entity of the same type.
Performance	A measure of a student's proficiency with the concept represented by an entity.
Requisition	Prerequisites and corequisites.

4.2. THE COURSE SELECTION PROBLEM

A course selection algorithm (CSA) finds a set of courses (a *course selection*) for a particular student, grouped and ordered by time, that would satisfy all of that student's curricular requirements. The total adherence to institution constraints is mandatory because with it comes the guarantee of eventual graduation by the student if they are successful. Once validity is established, a course selection and the CSA from which it came can be further judged in numerous ways, such as: level of personalization, time until graduation, difficulty, and probability of student success. The course selection problem is roughly comparable to one that academic advisors attempt to solve, and previous attempts at automating this process are discussed in Section 2. However, PERCEPOLIS does not apply to just courses, and therefore the course selection problem simply serves as a stepping stone to the more generic problem of *selecting entities* which will be made more evident in the next section. For now, this section lays the groundwork by discussing course selection techniques.

The PERCEPOLIS course selection algorithm addresses the following areas of emphasis (AOE) for a target student:

1. requirements (including, but not limited to, those set by an institution);
2. personal interests;
3. time restrictions; and
4. increased likelihood of higher performance.

Most of the related work surrounding this area deals primarily with the first AOE, while personal interests are considered much less. Furthermore, this is by no means an exhaustive list of all of the possible AOE's.

The CSA presented in this thesis has gone through many different versions to become applicable to more than just courses. No attempt is made to prove an optimal solution that perfectly addresses all AOE's. This stems from the nature of some AOE's for which *satisfaction* can only be shown after the fact by feedback, and even then measures can be interpreted as subjective. Consider the fourth AOE - increased likelihood of higher performance: it is intentionally described as such because guaranteeing higher performance in this context is not possible. How might one go about verifying that a specific course selection will lead to better performance in an individual? One method would be to generate a personalized selection for a student (using a CSA), enumerate all the possible selections, have the same student carry out all selections under controlled conditions, record their performance, and then see if their performance with the recommended selection matches the best performance from the enumerated ones. However, it is not possible to have one or more students do this as there is only one opportunity that must be as successful as possible. A more realistic and straight-forward approach is to use heuristics derived from the known performance of similar students. With this information, courses could be selected in such a manner that students of all backgrounds are more likely to have a higher GPA.

The focus is therefore on selections which are optimal in some ways while "good enough" in others as determined by certain thresholds for AOE satisfaction. A selection can be optimal in terms of time-to-degree by not having the student take anymore than the minimum number of courses needed to graduate, as determined from institutions rules. A selection can also be optimal in terms of personal interests by catering to all of the student's interests. However, a selection does not completely guarantee that selecting one course over another will result in a higher GPA for the target student, and so it is this probability which introduces partial sub-optimality. The CSA sets conditions of satisfaction for each AOE, based on the information provided and then attempts to create a schedule that satisfies all of them.

The remaining subsections of Section 4 address the course selection problem directly, first by establishing the form of the data (derived from the MCH) on which the CSA operates. This data is then analyzed to derive the decision variables, constraints, and objective function to treat this problem as an ILP.

4.3. DATA

The PERCEPOLIS CSA generates recommended course schedules with consideration of four AOEs: degree requirements, interests, increased likelihood of higher performance, and desired time-to-degree. Each subsection below describes how information relates to the aforementioned AOE's. The following list outlines all of entities and their properties in order to perform the algorithms needed to carry out course selection.

- Student
  - *id*: unique identifier
  - *name*: the student's first and last name
  - *interests*: a list of keyword interests
  - *curricula*: a list of curriculum entities

  - $completedCourses$: a list of completed courses with known grade and semester
  - $explicitCourses$: a list of explicitly desired course entities provided by the student
  - $implicitCourses$: a list of implicitly desired course entities inferred from $interests$ [1]
  - $explicitModules$: a list of explicitly desired module entities provided by the student
  - $implicitModules$: a list of implicitly desired module entities inferred from $interests$ [2]
  - $completedModules$: a list of completed modules with known grade
- Curriculum
  - id: a unique identifier
  - $name$: a label describing the nature of the curriculum
  - $requirements$: a list of requirement entities needed to complete the curriculum
- Requirement
  - id: a unique identifier
  - $name$: a label describing the nature of the curriculum
  - $creditThreshold$: a numeric threshold towards which courses are applied and above which the requirement is considered fulfilled
  - $courses$: a list of courses that can be used to complete the requirement
- Course
  - id: a unique identifier
  - $name$: a label describing the nature of the curriculum
  - $credits$: a numeric value that can applied towards one or more requirement credit thresholds

[1]Note that keywords interests are assumed to have already been used to populate the sets of implicitly desired courses and modules.

[2]See footnote 1

  - $creditThreshold$: a numeric threshold towards which modules are applied and above which the course is considered fulfilled
  - $creditsRequired$: a numeric threshold representing the number of credits a student must have completed in previous semesters before taking this course
  - $modules$: a list of modules that can be used to complete the course
  - $prerequisites$: a list of prerequisite courses
  - $postrequisites$: a list of postrequisite courses [3]
  - $corequisites$: a list of corequisite courses
  - $consequisites$: a list of consequisite courses
  - $availabilities$: a list of semesters in which the course is known to be available
  - $unvailabilities$: a list of semesters in which the course is known to be unavailable
  - $opportunity$: the number of courses to which this course leads

- Module
  - id: a unique identifier
  - $name$: a label describing the nature of the curriculum
  - $credits$: a numeric value that can applied to a course's $creditThreshold$
  - $prerequisites$: a list of prerequisite modules
  - $postrequisites$: a list of postrequisite modules

- Semester
  - id: a unique identifier
  - $name$: a label describing the nature of the curriculum
  - $beginsOn$: a real-world beginning date/time
  - $endsOn$: a real-world end date/times
  - $maxCredits$: a maximum amount of credits allowable
  - $minCredits$: a minimum amount of credits allowable

[3]Courses for which this course is a prerequisite

4.3.1. Institutional Requirements. For students enrolled in a traditional college or university, institutional requirements are somewhat rigid. In order to produce a valid course schedule, the CSA needs to know what courses are available at a given institution, credit requirements, how those courses relate (prerequisites, corequisites, *consequisites*), and what courses must be taken in order to complete a given curriculum. If only institutional requirements and student transcripts are known, the CSA produces valid schedules. It is assumed that the CSA's database has up-to-date student transcript information about what courses they have taken. This gives the CSA the ability to use previously taken courses towards curriculum requirements. Previously taken modules can be applied to multiple courses which also feature that module, consequently shortening the amount of time it takes to complete the course. Alternatively, completing a module shared between courses enables the possibility of substituting that time with a more interesting module or more challenging one.

When courses are less formal, bear no time restrictions, and do not fall under a curriculum in the traditional sense, the possibilities for entity selection grow. Online courses, for example, are often available on-demand, so with respect to the CSA this means they can be started at any particular time. Regardless, the CSA requires that time steps be defined for any entity on which availability selection is eventually carried out, so for courses the time steps are obviously semesters. Discretizing the time within semesters to do availability selection on modules makes little sense at this time, especially since if a course is offered, the modules within are assumed to be available.

4.3.2. Interests. The CSA will try to personalize schedules based on the explicit and implicit interests of the target student. Students may explicitly specify courses and modules they want to take, or if they are unsure, they can provide keywords for topics they want to learn. The CSA will incorporate these words or phrases into the final schedule by attempting to match them to existing courses and modules. Unfortunately, a semantic gap typically exists between the keywords provided by the student and the information available for courses and modules, which may be limited to simply a name. To address this semantic gap, the CSA makes use of the Word2Vec algorithm [36] described in Section 2. The Word2Vec algorithm is first trained on a quality, english text corpus in order to produce meaningful vector representations of words in the language. Numerous corpora for any desired language can be found with a simple Google search. Once all of the word vectors have been generated as a binary file, cosine similarity can be used to find the words which are most similar to some target word. A simple Python program which makes use of a Word2Vec package can be found in the Appendix of this thesis. For each word or phrase provided by the student, a list of similar words and phrases is obtained using Word2Vec predictions. The names and descriptions of every course and module are then searched for direct matches to the words and phrases in the obtained lists. If a matching course or module can be used towards at least one of the student's requirements, it is added to the student's implicit interests.

For future work, two improvements in this area would be useful: the first is analysis on the quality for text corpora; the second is multiple language support. Whether it be by software agents or simple routine procedures, the text corpus used to train word vector models should be regularly improved to make sure that interest matching is effective. The goal of this would be to remain accurate despite changes to course content. Moreover, it would certainly benefit some learners and institutions if an algorithm could handle multiple languages, which is something for which Word2Vec is already suited. This could provide yet another way, in additional to relationships within the MCH, to tie content from different institutions together.

4.3.3. Promoting Improved Performance. The third AOE involves increasing the target student's likelihood of higher performance in a manner which is balanced with the other AOE's. This section discusses how one might do this by eliminating certain courses (where performance is abnormally low) from consideration prior to the course selection stage. In other words, a requirement with multiple courses may have some of its courses unavailable for selection in order to prevent a particular target student from taking them. Unfortunately, no results for this technique are provided due to a lack of data, however it is still worthwhile to present it here as it is an available feature of the PERCEPOLIS CSA if data were present.

Estimating how a target student will do in a course is a delicate and mult-faceted problem. The amount of factors which result in a particular grade for any given student are incredibly numerous, while the data for most of these factors is difficult to obtain and/or nonexistent. Therefore an approach is proposed that both relies on data which is surely available (course grades) and also attempts to reduce some of the noise introduced by unknown factors.

If performance data was available, the PERCEPOLIS CSA would identify a cluster of students to which the target is most similar, where similarity is determined by the grades received in mutual courses. The set of students used would first be filtered down from the population of students at the target student's institution to include only those who have the same major(s) as the target student. Another requirement for this is that the peer students must be further along in their academic career than the target student, which includes both former and current students of the institution. If a clustering method such as k-means were used, the value of k would need to be set based on the analysis of a real dataset. To avoid this, agglomerative hierarchical clustering is used which starts with all students in their own cluster and iterately merges clusters which are the closest. Whenever, the target student's cluster is sufficiently large, the clustering can stop and the peer students present in the cluster can provide the basis for estimation. New students with no prior courses present a special case that requires alternative sources of information for comparison including, but not limited to, high school performance and ACT/SAT score. Course recommendations made for improving performance are more important early on in a student's academic career in order to promote retention, and as such, properly making use of alternative information remains an important open issue.

Before clustering, metrics for the distance between between students and between clusters must be defined. Since course history is the focus, each student is represented as a vector of all the courses taken by the target student. Let the total number of students in the target's major(s), including the target, be m, and let n be the number of courses taken by the target student. Also let h_i be the vector of length n for the i-th student, and let H be the set of course history vectors containing $h_i \forall i \in [1, m]$. Lastly, let $h_{i,g}$ denote the grade received by student i for course g where $g \in [1, n]$. Deriving a distance measure is pretty easy with this representation, the most common of which would be Euclidean distance. For two student vectors, the Euclidean distance between them can be found as follows:

$$Euclidean(h_i, h_j) = \Big(\sum_{k=1}^{n} |h_{i,k} - h_{j,k}|^{\frac{1}{2}}\Big)^2 \tag{4.1}$$

However, other distance metrics such Minkowski, Mahalanobis, City Block, etc. exist, and even similarity measures such as cosine can be modified to produce a measure of distance. The results of using any particular metric on real data remain to be explored.

The criteria by which clusters of students are merged must also be defined since the metric defined previously is only for computing the distance between two particular students, not clusters of students. This is known as the "linkage criteria." There are many types of linkage criteria discussed in [48] and [49], however the recommended approach of this thesis is group average linkage. As a point of information, this technically turns the clustering method into what is known as an Unweighted Pair Group Method with Arithmetic Mean (UPGMA). In UPGMA, the distance between two clusters is the average distance between the vectors in each cluster. Given two clusters of student vectors C_1 and C_2, the distance between them would be defined as:

$$\frac{1}{|C_1| \cdot |C_2|} \sum_{a \in C_1} \sum_{b \in C_2} distance(a, b) \tag{4.2}$$

where two bars (|) represent the length of a vector and $distance(a, b)$ is whatever distance metric was chosen previously. The results of different linkage criteria should be compared on real data in order to determine that which is best, however the UPGMA approach is taken primarily because it tends to join clusters with small variances first, which can be interpreted as leading the target student to a cluster in which the students are less dispersed and therefore slightly more similar.

When the size of the target student's cluster becomes sufficiently large to provide a good result the peer students in the cluster are extracted. Note that when the vectors were formed, only the courses that were taken by the target student were used. In other words, the clusters were not formed using the entire history of peer students, but rather only the portion of their history which was the same as the target student's. Imagine for a moment that the peer students' full histories were used, increasing the length of the vectors in H. The target student would likely have a course history that is much more sparse, and therefore judged to be vastly more distant from everyone else. This would lead the target student to be merged into a cluster at a much later iteration. With that aside and with similar peer students obtained, the mean grade for each of the courses not shared with the target student is calculated. Curriculum requirements for which there are many course options, that also have a course with a low average grade from the cluster of peers, will have such courses removed entirely as options. The threshold for such removal, like many other things in this section, is yet another open question, but one that must be balanced so as not to eliminate too many choices. Moreover, if the target student is a high performer, the threshold should be removed in order to allow the target student to take courses perceived as more difficult. This prevents the objectionable scenario where harder courses are never selected. Therefore, this mechanism would only apply to underperforming students by disallowing the selection of courses where the difficulty is perceived to be abnormally high among students similar to the target.

Future work may consider investigating whether or not clustering is improved by deriving distance with different measures. Moreover, incorporating consideration for the order in which courses may lead to more accurate metrics. A starting point for considering the order in which courses were taken would be an edit distance calculation that finds the best global alignment and then subsequently considering grades [50, 49]. It would also be worthwhile to expand this approach to the module level as it would offer a higher level of detail with respect to a student's behavior and performance.

4.3.4. Desired Time-to-Degree. The final AOE relates to the amount of time the student would prefer to spend pursuing their degree. Schedule height, or amount of credits per semester, is initially set by the target student's institution as follows:

1. $minCr_t$: the minimum number of credits for semester $t \in T$ where T is the time-ordered set of all semesters
2. $maxCr_t$: the maximum number of credits in semester t

Student-provided parameters are defined as follows:

1. $minCr_{s,t}$: the minimum number of credits for student $s \in S$ where S is the unordered set of all students, and for semester $t \in T$ where T is the time-ordered set of all semesters
2. $maxCr_{s,t}$: the maximum number of credits for student s in semester t
3. $maxSemesters_s$: the maximum number of semesters preferred by student s

The CSA always tries to make the schedule as short as possible independent of the value of $maxSemesters_s$. $maxSemesters_s$ is used by the CSA to determine whether or not a created schedule adheres to the target student's preference. If no value for $maxSemesters_s$ is provided, the CSA will set $maxSemesters_s$ to $scheduleLength$ which is the resultant length of the created schedule, ensuring the schedule is valid with respect to length. Since students cannot override institution limits, the following must hold:

$$minCr_t \leq minCr_{s,t} \leq maxCr_{s,t} \leq maxCr_t \tag{4.3}$$

Let $totalCredits_t$ be the total credits placed in semester t for the target student, and note that its value must fall within the minimum and maximum institution and student parameters. In order for a schedule to be institutionally valid, the following must hold:

$$0 < minCr_t \leq totalCredits_t \leq maxCr_t, \forall t \in T \tag{4.4}$$

In order for a schedule to be preferentially valid, the following must also hold:

$$0 < minCr_{s,t} \leq totalCredits_t \leq maxCr_{s,t}, \forall t \in T \tag{4.5}$$

Since the bounds do not have to be uniform for all semesters, a target student who desired to could adjust their $minCr_{s,t}$ and $maxCr_{s,t}$ such that their workload would change semester-by-semester instead of remaining uniform. However, if the gap between a minimum and maximum is too small, the CSA could fail to find a preferentially valid solution.

4.4. SCHEDULE CREATION

This section describes the schedule creation process within the CSA which consists of the following:

1. Curriculum/Requirement Satisfaction (Course Selection)
2. Course Availability Satisfaction (Semester Selection)
3. Course Satisfaction (Module Selection)

Schedule creation is broken down into a series of three satisfaction stages, each of which seeks to find an optimal or valid solution to one or more ILP's before passing the results to the next stage. The way in which each ILP is modeled (most notably in constraint derivation for requirements) is what differentiates the CSA from similar works. A flowchart of the creation process for a single schedule is provided by Figure 4.3.

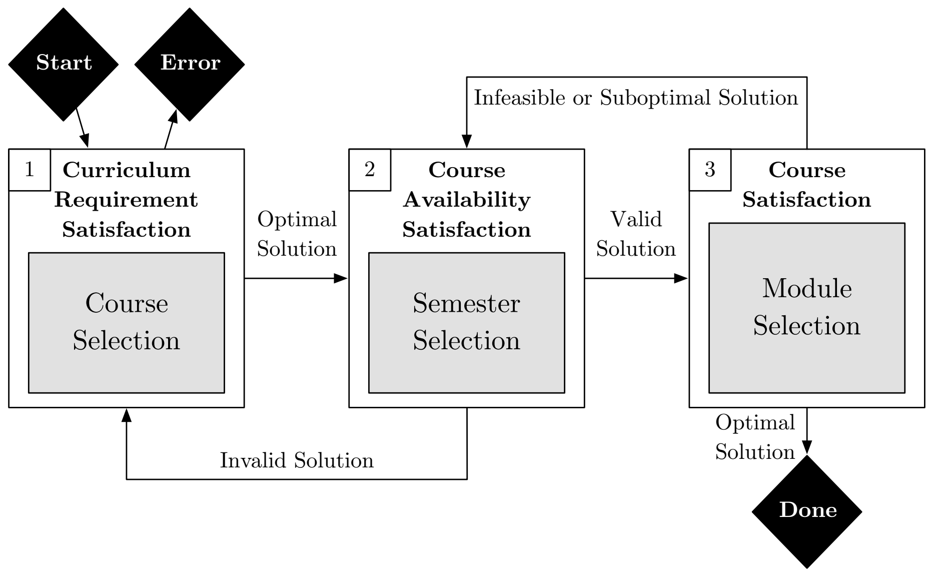

Figure 4.3. CSA Flowchart

Satisfaction stages 1 and 2 represent ILP's, while stage 2 is solved using proprietary algorithms. In general, progression from one satisfaction stage to another requires an optimal or valid solution from the previous stage. In general, the conditions for optimality can be known up front, and repeating previous stages is rare. Most errors or infeasible solutions arise in practice because the gap between the credit minimum and maximum is too narrow or because curriculum data was improperly entered. When implementing the CSA, analysis of the model can yield insight into what went wrong.

4.4.1. Curriculum/Requirement Satisfaction. Curriculum requirement satisfaction requires finding a selection of courses which satisfy all of the curriculum requirement credit thresholds. Recall from Section 4.3 that a Requirement entity consists of the courses which can satisfy it and a credit hour threshold to which those courses can be applied. An optimal solution for curriculum requirement satisfaction is a set of courses which minimizes the total number of credit hours taken while simultaneously adhering to all constraints. In order to solve the problem using one of the methods discussed in 2.4, decision variables, constraints, and an objective function must be created from the data we have.

4.4.1.1. Decision variables. Decision variables are created representing the decision to use a particular course i to satisfy a particular requirement j. Formally, the set of all decision variables for curriculum requirement satisfaction is defined as:

$$X = \{x_{i,j} : x_{i,j} \in \{0, 1\}, j \in R, i \in j.courses\} \tag{4.6}$$

where R is the set containing the target student's requirements and $j.courses$ is the set of all courses which can satisfy j. The entire set X is produced directly from the requirements R that must have been previously provided by institutions, and Algorithm 1 shows how to do this.

Algorithm 1 Iteratively generate all decision variables for the given set of requirements

```
1: procedure GetDecisionVariables($R$)
2:     $X \leftarrow []$                                  ▷ Empty list for decision variables
3:     for all $r \in R$ do
4:         for all $c \in r.courses$ do
5:             $X.append(x_{c,r})$
6:         end for
7:     end for
8:     return $X$
9: end procedure
```

4.4.1.2. Reusability constraints. It is possible for Algorithm 1 to create decision variables where a course could potentially satisfy different requirements. Allowing the creation of these variables is intentional because it is possible for one or more requirements to share one or more courses, which raises the question as to whether or not a course may count towards more than one requirement. Figure 4.4 illustrates the intersection of requirement course sets for a small curriculum of four requirements.

The general rule is that a course can only be used towards one requirement within any given curriculum. To ensure this rule holds, constraints must be created such that decision variables representing the same course within a curriculum are not selected more than once. Therefore the sum of the decision variables for course i must not exceed 1.

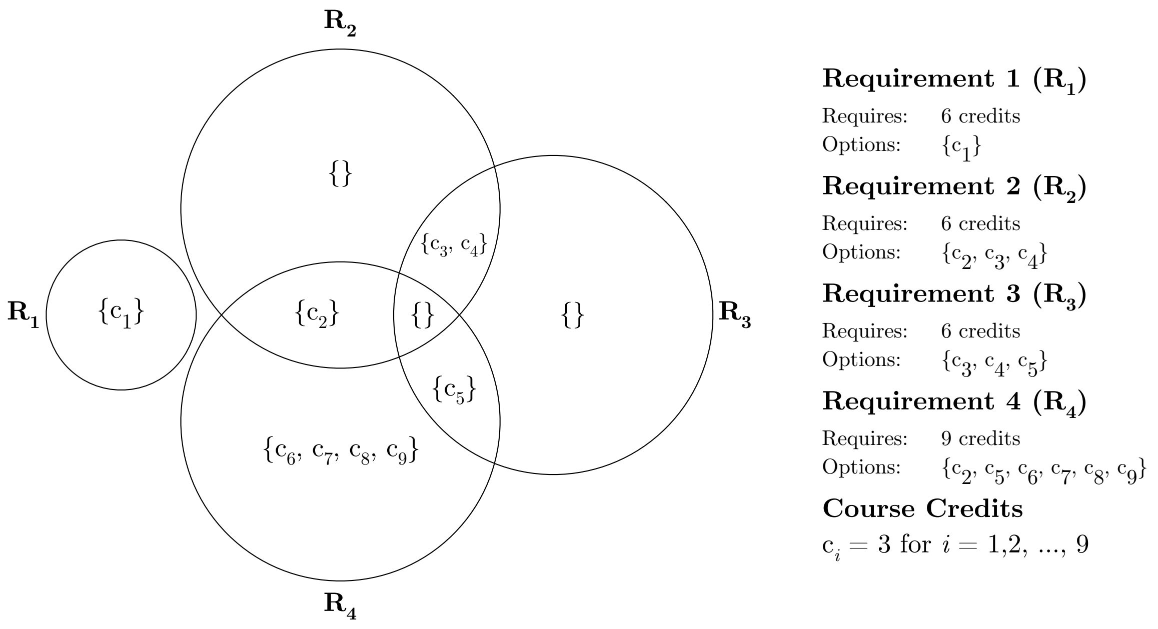

Figure 4.4. Shared Courses Between Requirements

Algorithm 2 can be used to obtain the reusability constraints for each curriculum. Note that D is the set of curricula for the target student and C is the set of all courses that can be used to fulfill the requirements in R.

Algorithm 2 Derive constraints such that variables for the same course within a curriculum can only be selected at most once

```
1: procedure GetReusabilityConstraints($X, D, C$)
2:     $N \leftarrow []$                                    ▷ Empty list for constraints
3:     for all $d \in D$ do
4:         for all $c \in C$ do
5:             $X_o \leftarrow \{x_{i,j} \in X : i == c \wedge j \in d.requirements\}$
6:             if $X_o.length > 1$ then
7:                 $constraint \leftarrow \sum_{x \in X_o} x \leq 1$
8:                 $N.append(constraint)$
9:             end if
10:        end for
11:    end for
12:    return $N$
13: end procedure
```

4.4.1.3. Prerequisite/corequisite constraints. A requirement's course list can sometimes include a course as well as one or more of its prerequisites. As such, up to this point, it would be possible to select a course without also selecting all of its prerequisites, but this is obviously not valid. While uncommon, this is a problem that arises within elective requirements where many options are available. Moreover, it must be the case that if a student is interested in a course, all of the prerequisites for the course must also be added. Because of this it is entirely possible that too many constraints can lead to solutions which do not minimize the number of credit hours, but do in fact include all of the student's interests. Conveniently, corequisite relationship can be ensured in the same way as a prerequisite relationship since both are, in essence, the constraint that one course is not selected without another. Algorithm 3 shows how to obtain inequality constraints that ensure that if a course is selected, all of its prerequisites must also be selected. The algorithm essentially creates a constraint based on the logical converse implication where a course B implies a prerequisite or corequisite course A.

Algorithm 3 Derive constraints such that a course is not selected without its prerequisites being selected.

```
procedure GetPrerequisiteConstraints($X, C$)
    $N \leftarrow []$                                        ▷ Empty list for constraints
    for all $c \in C$ do
        for all $p \in c.prerequisites$ do
            $X_c \leftarrow \{x_{i,j} \in X : i == c\}$               ▷ Decision variables for course
            $X_p \leftarrow \{x_{i,j} \in X : i == p\}$         ▷ Decision variables for prerequisites
            if $X_p.length > 0$ then
                $constraint \leftarrow |X_c| \cdot \sum_{x_p \in X_p} x_p - \sum_{x_c \in X_c} x_c \geq 0$
                $N.append(constraint)$
            end if
        end for
    end for
    return $N$
end procedure
```

4.4.1.4. Requirement credit constraints. Algorithm 4 creates a constraint for every requirement's credit hour threshold. For each requirement, the decision variables for that requirement are multiplied by their respective course credit and then summed. This sum must be greater than or equal to the requirement's credit hour threshold. The greater than aspect of the inequality creates flexibility in the event that requirement credit thresholds are not perfectly divisible by the requirement's courses.

Algorithm 4 Derive constraints such that variables for the same course within a curriculum are selected at most once

```
1: procedure GetRequirementCreditConstraints($R, X$)
2:     $N \leftarrow []$                                   ▷ Empty list for constraints
3:     for all $r \in R$ do
4:         $X_r \leftarrow \{x_{i,j} \in X : j == r\}$
5:         $constraint \leftarrow \sum_{x \in X_r} x.course.credits \cdot x \geq r.creditThreshold$
6:         $N.append(constraint)$
7:     end for
8:     return $N$
9: end procedure
```

4.4.1.5. Interest constraints. If one or more courses are implicit or explicit interests of the target student, then a constraint is added for each course such that the sum of all of the decision variables for that course are greater than or equal to 1. It is assumed that there is a requirement $r \in R$ for each course of personal interest. However, if the student is interested in a large number of courses, it is very possible that their total amount of credit hours will not be minimal. Therefore, the student can and should be warned of this in a manner which lets them know that while their interests are being satisfied, to carry out such a selection will require more money and possibly more semesters than they would like. Algorithm 5 provides a method for deriving interest constraints.

Algorithm 5 Derive constraint guaranteeing courses of interest are selected

```
1: procedure GetInterestConstraints($I, X$)
2:     $N \leftarrow []$                                   ▷ Empty list for constraints
3:     for all $c \in I$ do
4:         $X_I \leftarrow \{x_{i,j} \in X : i == c\}$
5:         if $X_I.length > 0$ then
6:             $constraint \leftarrow \sum_{x \in X_I} x \geq 1$
7:             $N.append(constraint)$
8:         end if
9:     end for
10:    return $N$
11: end procedure
```

4.4.1.6. Prior course or module constraints. The CSA also adds constraints to ensure selection of every prior course or module taken by the student. Previously created constraints for reusability will ensure that courses are not improperly chosen more than once, while modules do not currently have reusability constraints applied to them. By doing this, previously taken courses and modules are used wherever possible to work towards a goal of shortening the time-to-degree while satisfying curriculum requirements. Algorithm 6 obtains these constraints, and the process for modules is conceptually the same.

Algorithm 6 Iteratively generate constraints for previously taken courses

```
 1: procedure GetPriorCourseConstraints($C, X$)
 2:     $N \leftarrow []$                                   ▷ Empty list for constraints
 3:     for all $c \in C$ do
 4:         $X_c \leftarrow \{x_{i,j} \in X : i == c\}$
 5:         for all $x \in X_c$ do
 6:             $constraint \leftarrow \sum_{x \in X_c} x \geq 1$
 7:         end for
 8:         $N.append(constraint)$
 9:     end for
10:     return $N$
11: end procedure
```

4.4.1.7. Total credit constraint. A constraint is also added for the sum of requirement thresholds which is a lower bound for the eventual objective function $F(x)$.

$$F(x) \geq \sum_{r \in R} r.creditThreshold \tag{4.7}$$

4.4.1.8. Objective function. The objective function to be minimized is given as:

$$F(x) = \sum_{x_{i,j} \in X} u_i x_{i,j} \tag{4.8}$$

where u_i is the credit value for course i.

4.4.1.9. Solving. With the objective function, decision variables, and constraints all collected, the ILP is formed and can now be solved. As an ILP, an established method such as COIN-OR branch and cut can be used to find an optimal solution [46]. The result of this satisfaction stage is a list of courses C_s which are guaranteed to satisfy the student's curriculum requirements.

4.4.2. Availability Satisfaction. Availability satisfaction requires placing each course into the earliest semester possible in an attempt to shorten the time-to-degree. This is done by iterating over each future semester and assigning a course to the semester if possible. This particular stage does not necessarily call for an ILP but is one way of doing it. Instead, an iterative approach is taken which gives custom special priority to the certain courses with respect to when they enter the schedule. At each iteration (or semester), the CSA will iterate down a prioritized list of courses and place those courses which are available and have all of their prerequisites met.

4.4.2.1. Course prioritization. In order for the algorithms in Section 4.4 to work, the following additional properties are defined for the course and semester entities:

- Course
  - $semester$: the semester where the course is assigned which is initially nil
  - $isExplicitInterest$: a boolean value where $true$ denotes that the course was explicitly chosen by the student
  - $isImplicitInterest$: a boolean value where $true$ denotes that the course was identified as being of interest to the target student by matching the student's keyword interests to courses in their curriculum
- Semester
  - $creditsAdded$: the sum of credits that have been added to the semester which is initially 0

The additional properties are added so that the CSA can keep track of which courses have been added to the schedule, preventing courses from being scheduled more than once.

The list of courses, C_s, obtained in the previous stage are first grouped by the level of interest the student has in them, with higher priority groups being located towards the front of the list. Priority is first given to courses of explicit interest, then implicit interest, and finally low interest. Low interest simply represents that the student has not displayed any direct interest. Each interest group is then sorted in descending order of *opportunity value* represented as $c.opportunity, c \in C_s$. To establish opportunity value, a directed acyclic graph is created where the vertices are courses and arcs are established by the prerequisite relationships between courses. Figure 4.5 depicts the prerequisite graph for a 128-hour undergraduate degree in Computer Science for a student interested in robotics. The numbers in the top-right corner of each vertex represent the course's credit value, and the numbers in the bottom-left corner represent the opportunity value for that course.

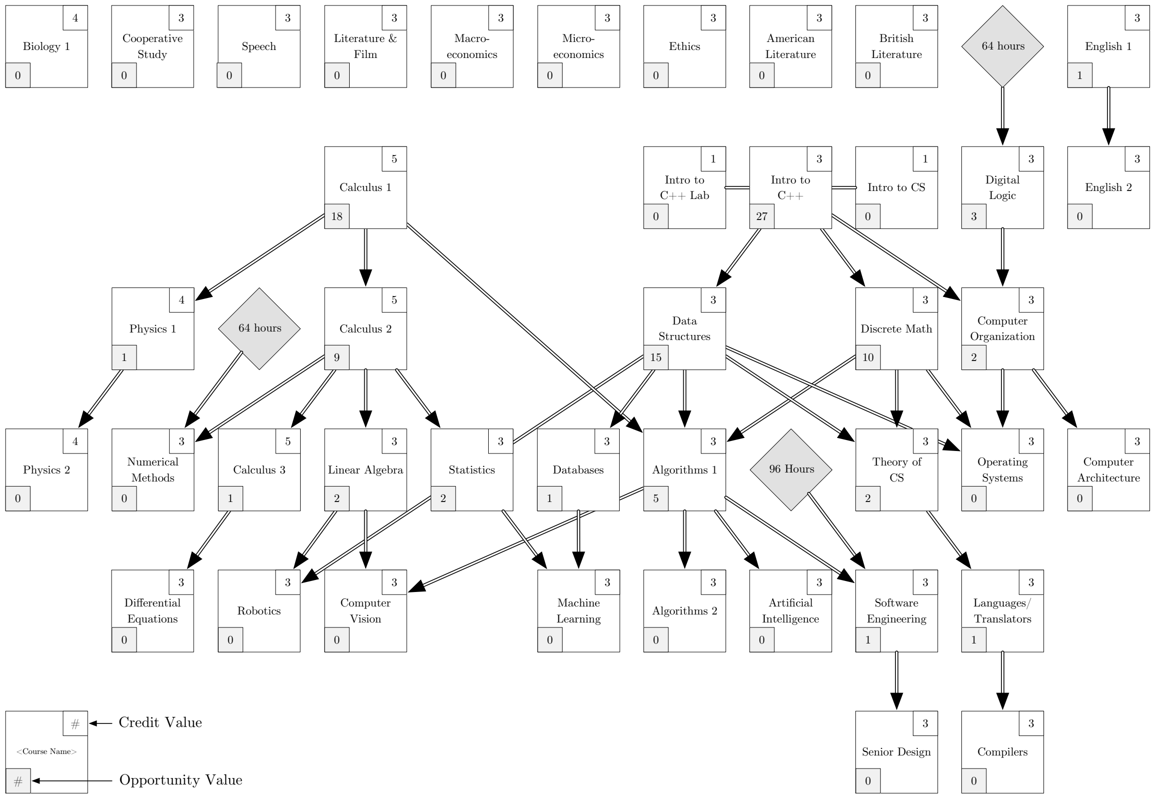

Figure 4.5. Prioritized Requirement Graph

When a student completes a course with postrequisites, the student unlocks the opportunity to take those subsequent courses. Naturally, each course can be valued in terms of the number of courses to which it provides access, whether the access be immediate or not. A course is therefore valued for its postrequisites and their postrequisites and so on. In order to ensure higher priority and earlier placement, a course's opportunity value should be greater than the opportunity value of any of its postrequisites. Courses are given their opportunity value by simple graph traversal. An artificial root node could be added to connect the entire graph and serve as a starting point, but that is an implementation decision of little concern here.

The definition of opportunity value for a course is the number of postrequisites plus the sum of opportunity values of the postrequisites. More formally, let the function $O(c)$ representing the opportunity value of course c be defined as:

$$O(c) = \begin{cases} cpl + \sum\limits_{d \in c.postrequisites} O(d), & \text{if } cpl > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

where $cpl = c.postrequisites.length$. Algorithm 7 shows one approach to setting the opportunity value for courses in a prerequisite graph.

Algorithm 7 A recursive approach to setting course opportunity values for a list of courses, C.

```
Require: The initial values for all $opportunity$ properties must be $nil$
 1: procedure OpportunityCount($c$)
 2:     $c.opportunity \leftarrow |i.postrequisites|$
 3:     if $c.opportunity > 0$ then                    ▷ If there are postrequisites
 4:         for all $d \in c.postrequisites$ do                 ▷ $d$ for descendant
 5:             if $d.opportunity == nil$ then       ▷ Uncounted postrequisite found
 6:                 $OpportunityCount(d)$
 7:             end if
 8:             $c.opportunity \leftarrow c.opportunity + d.opportunity$
 9:         end for
10:     end if
11: end procedure
```

Algorithm 7 employs bottom-up dynamic programming and memoization to reduce the need for repeated recursions, and as a result, the number of times the algorithm recurses is equal to the number of nodes in the graph. Note that the longest path in prerequisite graph represents the shortest possible number of semesters to graduation, a fact which can serve validation purposes if desired. The previously shown Figure 4.5 also depicted the results of executing Algorithm 7 on all of the courses in the prerequisite graph. Once every course has an opportunity value, the prioritized list of courses can be formed. For the prerequisite graph shown in Figure 4.5, the prioritized course list would look similar to Figure 4.6. Algorithm 8 can be used to obtain a prioritized list of courses for a given set of courses.

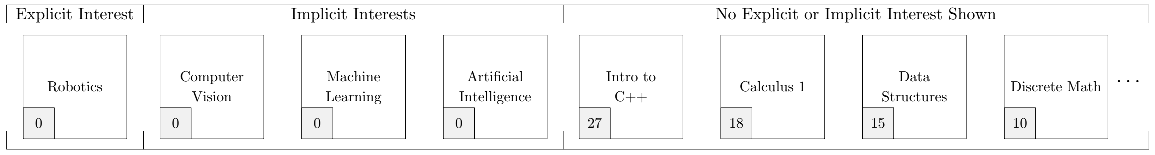

Figure 4.6. Prioritized Course List

Algorithm 8 An iterative approach to prioritizing a course list, C.

```
procedure PrioritizeCourses($C$)
    $P \leftarrow []$
    for all $c \in C$ do
        $CountDescendants(c)$
    end for
    for all $c \in C$ do
        if $c.isExplicitInterest$ then
            $P.add(c)$
            $C.remove(c)$
        end if
    end for
    for all $c \in C$ do
        if $c.isImplicitInterest$ then
            $P.add(c)$
            $C.remove(c)$
        end if
    end for
    $D \leftarrow C.sortedByDescendentCount()$
    $P.extend(D)$
    return $P$
end procedure
```

4.4.2.2. Semester selection. With a prioritized list of courses for the target student, C_p, every course $c \in C_p$ must be placed in some semester $t \in T$. A simple procedure which adds a given course c to a semester t is provided by Algorithm 9.

Algorithm 9 A procedure for adding a course, c, to a semester, s.

```
1: procedure AddCourseToSemester($c, t$)
2:     $t.courses.insert(c)$
3:     $t.creditsAdded \leftarrow t.creditsAdded + c.credits$
4:     $c.semester \leftarrow t$
5: end procedure
```

However, before a course can be added to a semester, the CSA must know that it is allowed to do so. Algorithm 10 shows how the CSA determines whether or not it can place a course c in semester s by performing a series of checks.

Algorithm 10 Check that all conditions needed for adding a course, c, to a given semester, s, have been met. a is the number of credits accumulated in previous semesters, and $maxCredits$ is the maximum amount of credits that can be added to the given semester.

```
Require: Corequisites can not have already been added to the given semester
1: procedure CanBeAdded($c, s, a, maxCredits$)
2:     if $a < c.creditsRequired$ then
3:         return false
4:     end if
5:     for all $p \in c.prerequisites$ do
6:         if $p.semester == nil$ or $p.semester >= s$ then
7:             return false
8:         end if
9:     end for
10:    $potentialCredits \leftarrow c.credits + s.creditsAdded$
11:    for all $c_{co} \in c.corequisites$ do
12:        $potentialCredits \leftarrow potentialCredits + c_{co}.credits$
13:    end for
14:    if $maxCredits < potentialCredits$ then
15:        return false
16:    end if
17:    if $s \in c.unavailabilities$ then
18:        return false
19:    end if
20:    return true
21: end procedure
```

First, the number of credits acquired thus far, a, is compared against the course's $creditsRequired$ to see if enough credits have been earned in previous semesters. If not, the function returns $false$. Secondly, the function verifies that all of the courses's prerequisites have been taken in prior semesters. Thirdly, the function checks to see if adding the course and its corequisites will exceed the maximum number of credits allowed, $maxCredits$, where $maxCredits = min(maxCredits_s, maxCreditss, t)$ (refer back to Section 4.3.4 if needed). Finally, if the course is known to be unavailable in the given semester, $false$ is returned.

When a course's availability is unknown (i.e., there is no record availability or unavailability in the database), the CSA will assume the course will be available. This assumption leads schedules to have a degree of uncertainty. As such, a course placed in a semester where its availability is unknown should be perceived as degrading the quality of the overall schedule. This problem is easily remedied by institutions if the availability or unavailability of all courses is reported prior to execution of the CSA.

The last function for semester selection involves iteration over all semesters. For each semester, for each course, Algorithm 11 determines whether or not the course can be added to the semester. If so, the course is added along with all of its corequisites. The course's consequisites are also added to the next semester in which they are not uavailable. Note that Algorithm 11 makes use of all prior algorithms introduced in Section 4.4.

4.4.3. Course Satisfaction. Up to this point, the CSA has dealt with courses, but the final stage involves selecting the modules (or topics) with the selected courses that student will take. This process is a reimagining of the curriculum requirement satisfaction stage where courses are now the entities that must be satisfied. Consequently, decision variables represent the binary decision to use a module for a course. The course credit constraints, prior module constraints, interest constraints, and total credit constraint are all used. However, modules are sometimes marked as required which produces additional constraints ensuring that their respective decision variables are selected. The objective function, minimizing the sum of credit hours, remains the same.

Algorithm 11 An iterative approach to adding courses to semesters

```
Require: Courses in $C$ which have already been taken by the student are assumed to already have their $semester$ attributes filled out prior to calling this function.
Require: $P$ must be checked after being returned to verify that all courses were assigned to a semester. When implemented in practice, exceptions can be thrown when edge cases are hit to pinpoint exactly where the error lies.
 1: procedure SemesterSelection($C, S, n, t$)    ▷ $n$ is the next semester from now    ▷ $t$ is the target
 2:     $P \leftarrow PrioritizeCourses(C)$
 3:     $S_f \leftarrow \{s : n \leq s\}$                                  ▷ Future semesters
 4:     $S_h \leftarrow \{s : s < n\}$                                  ▷ Past semesters
 5:     $priorCredts \leftarrow 0$
 6:     for all $s \in S_h$ do
 7:         $priorCredits \leftarrow priorCredits + \sum_{c \in s} c.credits$
 8:     end for
 9:     for all $s \in S_f$ do
10:         $maxCredits \leftarrow min(t.maxCr_s, s.maxCr)$
11:         for all $c \in P$ do
12:             if $c.semester \neq nil$ then
13:                 $continue$
14:             else if $s.creditsAdded \geq maxCredits$ then
15:                 $break$
16:             else if $CanBeAdded(c, t, priorCredits, maxCredits)$ then
17:                 $AddCourseToSemester(c, t)$
18:                 for all $c_{co} \in c.corequisites$ do
19:                     $AddCourseToSemester(c_{co}, t)$
20:                 end for
21:                 for all $c_{con} \in c.consequisites$ do
22:                     $T_f \leftarrow \{t' \in T : t' > t\}$
23:                     for all $t_f \in T_f$ do
24:                         if $CanBeAdded(c_{con}, t_f, priorCredits, maxCredits)$ then
25:                             $AddCourseToSemester(c_{con}, t_f)$
26:                         end if
27:                     end for
28:                 end for
29:             end if
30:         end for
31:         $priorCredits \leftarrow priorCredits + \sum_{c \in s} c.credits$
32:     end for
33:     return $P$
34: end procedure
```

4.5. SELECTION ENUMERATION

In practice, each ILP being solved usually has more than one solution. Because of this, it is useful to enumerate some more solutions than the first in order to give the student some choice. Enumerating all optimal selections for stages 1 and 3 (curriculum requirement satisfaction) is simple to do. All one must do is repeatedly solve the same ILP until a sub-optimal solution is found. However, to do this, a constraint must be added for each previously found solution which prevents the same combination of courses from being selected again. ILP solutions which are optimal can contain no fewer courses, otherwise they would violate some requirement credit constraint. This property can be used to prevent the same solution from being found again by adding a constraint where the set of decision variables that represent an optimal solution must sum to be less than or equal to the number of decision variables in the set minus 1. More formally, if $V = \{x_{i,j} : x_{i,j} == 1\}$ is a known optimal solution to the curriculum requirement satisfaction ILP where i is the course and j is the requirement, then the constraint needed to ignore V is:

$$\sum_{x_{i,j} \in V} x_{i,j} <= |V| - 1 \tag{4.10}$$

This is equivalent to requiring that at least one course in V is not taken in future solutions. The case where a solution has a greater number of courses than the optimal solution V would be sub-optimal according to the given objective function, hence they can be ignored. The enumeration of module selections is effectively the same.

It should be noted that enumerating the optimal selections of real-world curricula is simple in concept but costly in terms of time. As an example, the real-world curriculum on which Figure 4.5 is based has trillions of alternative course selections which are minimal in credit value. To illustrate this simply, consider that the real-world computer science curriculum has 32 requirements as shown in Table 4.4. Also note that 22 out of the 32 curriculum requirements are 1-to-1, meaning there is no decision that must be made with respect to which course to take in order to satisfy those requirements. Combinatorially, this can be written as $\binom{1}{1}^{22}$. However, the other 10 requirements contain quite a few choices, and this is enough to greatly increase the number of possibilities. Consider that each course option for these remaining requirements is worth 3 credits. If one were to attempt a simple count of the number of possible selections, the result would amount to trillions.

$$\binom{1}{1}^{22}\binom{2}{1}\binom{3}{1}^{2}\binom{6}{1}^{2}\binom{7}{1}\binom{6}{2}\binom{20}{2}\binom{4}{3}\binom{16}{3}\binom{20}{3} \tag{4.11}$$

Note that while this count does not consider courses that appear as usable for multiple requirements, the sample curriculum is conservative in terms of the number of options that typically exist for elective requirements. As such, it should be taken as a rough estimation that simply illustrates the number of possibilities the aforementioned ILP's could be dealing with.

As the enumeration is carried out, the number of decision variables remains the same but the number of constraints increases by one with every solution found. One can imagine that if this were carried out for multiple curricula or for modules, the problem explodes even more. Therefore, if a CSA wants to enumerate a large number of schedules, it should make an attempt to reduce the number of decision variables and constraints up front as much as possible. This can be done by removing course decision variables for which there is no future availability or availability is uncertain. A CSA could also remove decision variables for courses that a student desires to avoid if that information were collected.

4.6. SCHEDULE PRESENTATION

The number of possible schedules for any given student in a real-world curriculum far exceeds what can be feasibly presented either to them or in this thesis. Up to this point, the CSA's functionality has been described so as to produce a set schedules where each schedule is a set of courses assigned to some semester, and for each course, a set of mod-

Table 4.4. Sample Computer Science Curriculum Requirements

	Name	Credit Threshold	Course Options
1	Algorithms	3	1
2	Data Structures	3	1
3	Databases I	3	1
4	Digital Systems Design	3	1
5	Discrete Math	3	1
6	English I	3	1
7	English II	3	1
8	Ethics Elective	3	2
9	History Elective	3	6
10	Humanities Elective	3	6
11	Intro to Computer Engineering	3	1
12	Intro to Numerical Methods	3	1
13	Intro to Operating Systems	3	1
14	Linear Algebra I	3	1
15	Literature Elective	3	3
16	Programming Languages and Translators	3	1
17	Senior Design	3	1
18	Software Engineering I	3	1
19	Speech	3	1
20	Statistics	3	1
21	Theory of Computer Science	3	1
22	Intro to Programming	4	1
23	Laboratory Science	4	7
24	Physics I	4	1
25	Physics II	4	1
26	Calculus I	5	1
27	Calculus II	5	1
28	Computer Science Electives (any)	6	20
29	Social Science Electives	6	6
30	Computer Science Electives (>5000)	9	16
31	Engineering and Science Electives	9	4
32	Free Electives	9	20

ules that should be taken. Depending on the configuration of the CSA, the CSA will present on or more schedules to the student, where different schedules will vary in their elective selection and consequently, the semester in which courses are assigned. The constraints for course and module selection give a general assurance that whatever schedules are produced from many different course selections, all of them incorporate the target student's preferences. Moreover, if performance data was present, the CSA would have also eliminated certain courses in which similar students collectively have the lowest average grade.

4.7. CONCLUSION

Section 4 described the course selection algorithm (CSA) contributed to PERCEPOLIS in order to generate personalized schedules for a given student. The discussion started with sections 4.1 and 4.2 to present key concepts which expanded or clarified aspects of PERCEPOLIS which was discussed in section 3. Afterwards, the data used by the CSA was defined in Section 4.3 as an interpretation of the modular course hierarchy discussed in Section 4.1. In Section 4.4, the different parts of the CSA's are presented as three sequential stages of course selection, semester selection, and module selection, the result of which is a selection of courses which satisfies all institution requirements and attempts to satisfy a student's interests and promote higher performance. Finally, Section 4.5 discusses how to turn course or module selections into constraints in order to rerun the CSA to enumerate schedules, which is an expensive operation.

5. RESULTS

The results of this section are produced by selecting a set of courses and modules for a target student, with the additional intention of constraining solutions in such a way to achieve different degrees of personalization. An outline of the objects and their characteristics can be found in Section 4.3. Each subsection of this Section corresponds to a student with a distinct profile. Scenario 5.1 presents the results of course selection broken down by stage as described in Section 4 for a student no prior courses or modules, or any known interests. However, Scenario 5.2 focuses on illustrating the effect that prior courses and known interests have on the curriculum from Scenario 5.1. By presenting the results in this manner, the goal is to further illustrate how the algorithm behaves with and without information from the student.

5.1. NO PRIOR COURSES OR PREFERENCES

The results shown here provide one solution for each stage of the CSA. In this scenario, the goal is to find a course, semester, and module selection for Student 1 who has no prior courses or known course preferences. Curriculum requirements are shown in Table 5.1. For this scenario each course is assumed to be 3 credits.

Table 5.1. Curriculum Requirements

Requirement 1	Needs 3 Credits
Course Options:	1
Requirement 2	Needs 6 Credits
Course Options:	2, 3, 4
Requirement 3	Needs 6 Credits
Course Options:	3, 4, 5
Requirement 4	Needs 9 Credits
Course Options:	2, 5, 6, 7, 8, 9

5.1.1. Course Selection. Curriculum requirement satisfaction deals with making a course selection that fulfills all requirements within a curriculum, includes all courses of interest to the student, and ultimately, minimizes the total number of credits taken in order to reduce time-to-degree.

5.1.1.1. Decision variables.

$$X = \{x_{i,j} : x_{1,1}, x_{2,2}, x_{3,2}, x_{4,2}, x_{3,3}, x_{4,3}, x_{5,3}, x_{2,4}, x_{5,4}, x_{6,4}, x_{7,4}, x_{8,4}, x_{9,4}\} \tag{5.1}$$

where i = course, j = requirement.

5.1.1.2. Requirement reusability constraints.

$$x_{2,2} + x_{2,4} \leq 1 \tag{5.2}$$

$$x_{3,2} + x_{3,3} \leq 1 \tag{5.3}$$

$$x_{4,2} + x_{4,3} \leq 1 \tag{5.4}$$

$$x_{5,3} + x_{5,4} \leq 1 \tag{5.5}$$

5.1.1.3. Requirement credit constraints.

$$C_1 x_{1,1} \geq R_1 = 3 \tag{5.6}$$

$$C_2 x_{2,2} + C_3 x_{3,2} + C_4 x_{4,2} \geq R_2 = 6 \tag{5.7}$$

$$C_3 x_{3,3} + C_4 x_{4,3} + C_5 x_{5,3} \geq R_3 = 6 \tag{5.8}$$

$$C_2 x_{2,4} + C_5 x_{5,4} + C_6 x_{6,4} + C_7 x_{7,4} + C_8 x_{8,4} + C_9 x_{9,4} \geq R_4 = 9 \tag{5.9}$$

where C_i represents the credit value for course i.

5.1.1.4. Prerequisite constraints.

$$x_{1,1} - x_{2,2} - x_{2,4} \geq 0 \tag{5.10}$$

$$x_{1,1} - x_{3,2} - x_{3,3} \geq 0 \tag{5.11}$$

$$x_{2,2} + x_{2,4} - x_{4,2} - x_{4,3} \geq 0 \tag{5.12}$$

$$x_{2,2} + x_{2,4} - x_{5,3} - x_{5,4} \geq 0 \tag{5.13}$$

$$x_{3,2} + x_{3,3} - x_{5,3} - x_{5,4} \geq 0 \tag{5.14}$$

$$x_{5,3} + x_{5,4} - x_{8,4} \geq 0 \tag{5.15}$$

$$x_{5,3} + x_{5,4} - x_{9,4} \geq 0 \tag{5.16}$$

$$x_{4,2} + x_{4,3} - x_{7,4} \geq 0 \tag{5.17}$$

$$x_{4,2} + x_{4,3} - x_{8,4} \geq 0 \tag{5.18}$$

$$x_{4,2} + x_{4,3} - x_{6,4} \geq 0 \tag{5.19}$$

5.1.1.5. Total credit constraint.

$$F(x) = \sum_i \sum_j C_i x_{i,j} \geq 24 \tag{5.20}$$

5.1.1.6. Objective function.

$$\text{Minimize } F(x) = \sum_i \sum_j C_i x_{i,j} \tag{5.21}$$

5.1.1.7. Solutions. Because the curriculum is small, all of the minimal solutions can be enumerated here in order to illustrate the discussion mentioned in Section 4.5. All of the minimal selections are shown in Table 5.2. For the remaining results regarding semester and module selection, the first course selection from Table 5.2 is used.

Table 5.2. Course Selection Solutions

Requirement	**Solutions**							
	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	2, 4	2, 4	2, 3	2, 3	2, 3	2, 4	2, 4	2, 3
3	3, 5	3, 5	4, 5	4, 5	4, 5	3, 5	3, 5	4, 5
4	6, 7, 8	6, 7, 9	6, 8, 9	7, 8, 9	6, 7, 9	7, 8, 9	6, 8, 9	6, 7, 8

5.1.2. Semester Selection. Availability satisfaction deals with selecting the semesters in which the previously selected courses should be placed. As a reminder, this stage involves the use of Algorithm 11. Figure 5.1 depicts the prerequisite graph of the selected courses shown in the first solution of Table 5.2, and Figure 5.2 shows the list of courses after being prioritized. All courses are unavailable in every third semester, every semester's max credits is 6, and semester 1 will be treated as the next semester.

5.1.2.1. Iteration 1. The schedule after iteration 1 appears as follows:

Semesters						
1	2	3	4	5	6	7
1						

5.1.2.2. Iteration 2. The schedule after iteration 2 appears as follows:

Semesters						
1	2	3	4	5	6	7
1	2					
	3					

5.1.2.3. Iteration 3. All courses are courses are unavailable - no change.

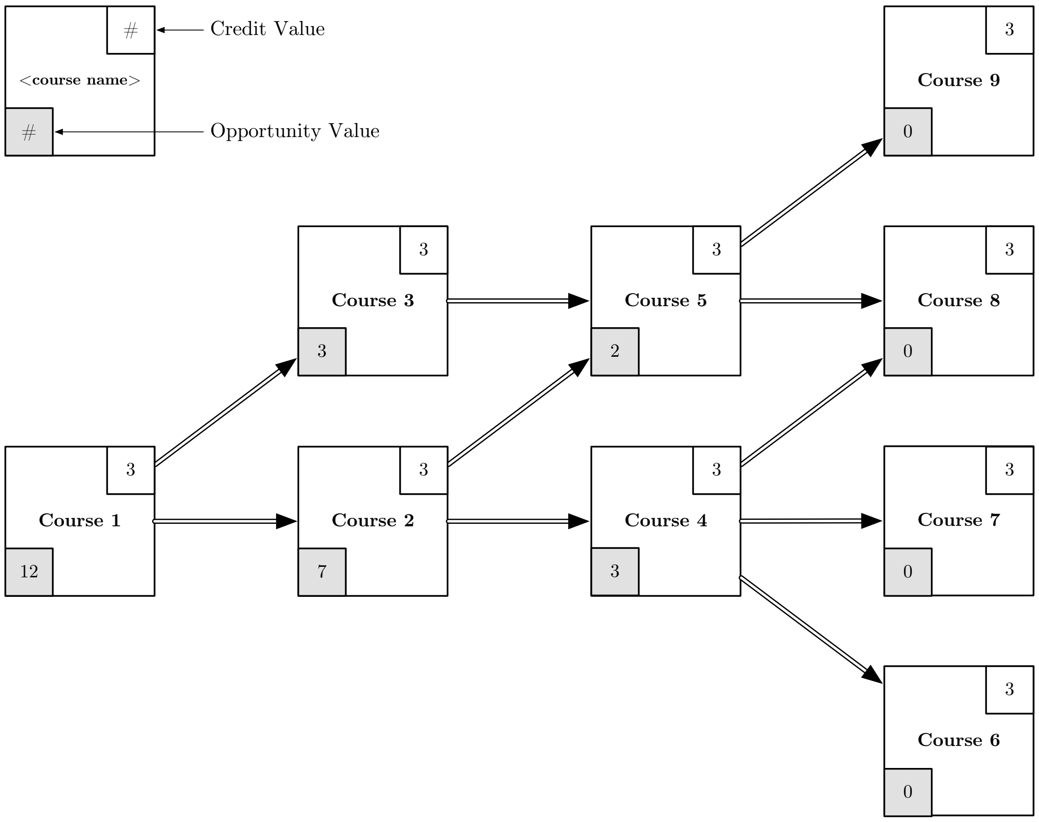

Figure 5.1. Prerequisite Graph

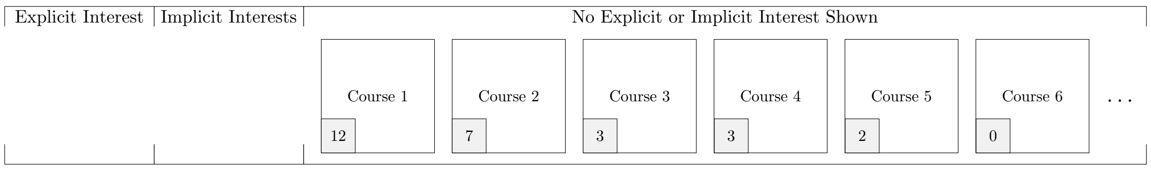

Figure 5.2. Priority List

5.1.2.4. Iteration 4. The schedule after iteration 4 appears as follows:

Semesters						
1	2	3	4	5	6	7
1	2		4			
	3		5			

5.1.2.5. Iteration 5. The schedule after iteration 5 appears as follows:

Semesters						
1	2	3	4	5	6	7
1	2		4	6		
	3		5	7		

5.1.2.6. Iteration 6. All courses are courses are unavailable - no change.

5.1.2.7. Iteration 7. The schedule after iteration 7 appears as follows:

Semesters						
1	2	3	4	5	6	7
1	2		4	6		8
	3		5	7		

5.1.3. Module Selection. Module selection deals with making a selection of modules such that all courses are fulfilled, and as a reminder, this stage is effectively the same as course selection. Figure 5.3 shows the courses, their modules, and all credit values. Note that Student 1 has no prior modules or known modules preferences, which satisfies the curriculum requirements.

5.1.3.1. Decision variables.

$$\begin{aligned} X = \{x_{i,j} :& x_{1,1}, x_{2,1}, x_{3,1}, x_{4,1}, x_{5,2}, \\ & x_{6,2}, x_{7,2}, x_{8,2}, x_{9,3}, x_{10,3}, \\ & x_{11,3}, x_{12,3}, x_{13,4}, x_{14,4}, x_{15,4}, \\ & x_{16,4}, x_{9,5}, x_{17,5}, x_{18,5}, x_{13,6}, \\ & x_{19,6}, x_{20,6}, x_{21,6}, x_{15,7}, x_{22,7}, \\ & x_{23,7}, x_{24,7}, x_{17,8}, x_{25,8}, x_{26,8}, x_{27,8}\} \end{aligned} \tag{5.22}$$

where i = module, j = course.

Course 1 (C_1)

Requires: 3 credits

Options: $\{*\underline{m_1}, \underline{m_2}, m_3, m_4\}$

Course 2 (C_2)

Requires: 3 credits

Options: $\{\underline{m_5}, \underline{m_6}, m_7, m_8\}$

Course 3 (C_3)

Requires: 3 credits

Options: $\{\underline{m_9}, \underline{m_{10}}, m_{11}, m_{12}\}$

Course 4 (C_4)

Requires: 3 credits

Options: $\{\underline{m_{13}}, \underline{m_{14}}, \underline{m_{15}}, m_{16}\}$

Course 5 (C_5)

Requires: 3 credits

Options: $\{\underline{m_9}, \underline{m_{17}}, \underline{m_{18}}\}$

Course 6 (C_6)

Requires: 3 credits

Options: $\{\underline{m_{13}}, \underline{m_{19}}, m_{20}, m_{21}\}$

Course 7 (C_7)

Requires: 3 credits

Options: $\{\underline{m_{15}}, \underline{m_{22}}, m_{23}, m_{24}\}$

Course 8 (C_8)

Requires: 3 credits

Options: $\{\underline{m_{17}}, \underline{m_{25}}, m_{26}, m_{27}\}$

Course 9 (C_9)

Requires: 3 credits

Options: $\{\underline{m_{18}}, \underline{m_{28}}, m_{29}, m_{30}\}$

Module Credits

All modules are assumed to be 1 credit.

*Underlined modules are required

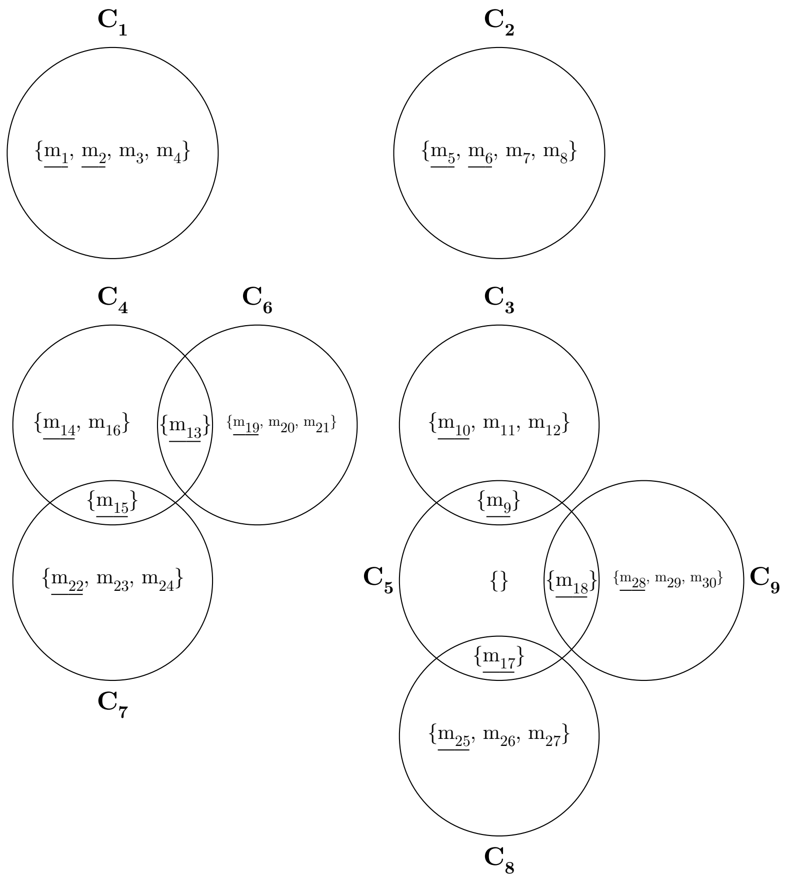

Figure 5.3. Shared Modules Between Courses

5.1.3.2. Required module constraints.

$$x_{1,1} \geq 1 \tag{5.23}$$

$$x_{2,1} \geq 1 \tag{5.24}$$

$$x_{5,2} \geq 1 \tag{5.25}$$

$$x_{6,2} \geq 1 \tag{5.26}$$

$$x_{10,3} \geq 1 \tag{5.27}$$

$$x_{14,4} \geq 1 \tag{5.28}$$

$$x_{19,6} \geq 1 \tag{5.29}$$

$$x_{22,7} \geq 1 \tag{5.30}$$

$$x_{25,8} \geq 1 \tag{5.31}$$

$$x_{9,3} + x_{9,5} \geq 1 \tag{5.32}$$

$$x_{13,4} + x_{13,6} \geq 1 \tag{5.33}$$

$$x_{15,4} + x_{15,7} \geq 1 \tag{5.34}$$

$$x_{17,5} + x_{17,8} \geq 1 \tag{5.35}$$

$$x_{18,5} \geq 1 \tag{5.36}$$

5.1.3.3. Course credit constraints.

$$M_1\underline{x_{1,1}} + M_1\underline{x_{2,1}} + M_1x_{3,1} + M_1x_{4,1} \geq C_1 = 3 \tag{5.37}$$

$$M_2\underline{x_{5,2}} + M_2\underline{x_{6,2}} + M_2x_{7,2} + M_2x_{8,2} \geq C_2 = 3 \tag{5.38}$$

$$M_3\underline{x_{9,3}} + M_3\underline{x_{10,3}} + M_3x_{11,3} + M_3x_{12,3} \geq C_3 = 3 \tag{5.39}$$

$$M_4\underline{x_{13,4}} + M_4\underline{x_{14,4}} + M_4\underline{x_{15,4}} + M_4x_{16,4} \geq C_4 = 3 \tag{5.40}$$

$$M_5\underline{x_{9,5}} + M_5\underline{x_{17,5}} + M_5\underline{x_{18,5}} \geq C_5 = 3 \tag{5.41}$$

$$M_6\underline{x_{13,6}} + M_6x_{19,6} + M_6x_{20,6} + M_6\underline{x_{21,6}} \geq C_6 = 3 \tag{5.42}$$

$$M_7\underline{x_{15,7}} + M_7x_{23,7} + M_7x_{24,7} + M_7\underline{x_{25,7}} \geq C_7 = 3 \tag{5.43}$$

$$M_8\underline{x_{17,8}} + M_8\underline{x_{25,8}} + M_8x_{26,8} + M_8x_{27,8} \geq C_8 = 3 \tag{5.44}$$

where M_i represents the credit value for module i and underlined decision variables represent those modules that are required.

5.1.3.4. Total credit constraint.

$$F(x) = \sum_i \sum_j M_i x_{i,j} \geq 24 \tag{5.45}$$

5.1.3.5. Objective function.

$$\text{Minimize } F(x) = \sum_i \sum_j M_i x_{i,j} \tag{5.46}$$

5.1.3.6. Solution. The solution is presented in Table 5.3.

Table 5.3. Module Selection Solution

Course	Module(s) Selected
1	1, 2, 3
2	5, 6, 7
3	10, 11, 12
4	13, 14, 15
5	9, 17, 18
6	19, 20, 21
7	22, 23, 24
8	25, 26, 27

5.2. PRIOR COURSE AND EXPLICIT PREFERENCE

In this scenario, Student 2 has the same curriculum as Student 1 (shown in Table 5.1), but Student 2 has previously completed course 3 and has a known interest in taking course 9 and all of its modules (18, 28, 29, and 30). The model from Scenario 5.1 is almost identical to the one used here, but the prior course and course preference lead to additional constraints. Each course is still 3 credits.

5.2.1. Course Selection. As a reminder, curriculum requirement satisfaction deals with making a course selection that fulfills all requirements within a curriculum, includes all courses of interest to the student, and ultimately, minimizes the total number of credits taken in order to reduce time-to-degree.

5.2.1.1. Decision variables.

$$X = \{x_{i,j} : x_{1,1}, x_{2,2}, x_{3,2}, x_{4,2}, x_{3,3}, x_{4,3}, x_{5,3}, x_{2,4}, x_{5,4}, x_{6,4}, x_{7,4}, x_{8,4}, x_{9,4}\} \tag{5.47}$$

where i = course, j = requirement.

5.2.1.2. Requirement reusability constraints.

$$x_{2,2} + x_{2,4} \leq 1 \tag{5.48}$$

$$x_{3,2} + x_{3,3} \leq 1 \tag{5.49}$$

$$x_{4,2} + x_{4,3} \leq 1 \tag{5.50}$$

$$x_{5,3} + x_{5,4} \leq 1 \tag{5.51}$$

5.2.1.3. Requirement credit constraints.

$$C_1 x_{1,1} \geq R_1 = 3 \tag{5.52}$$

$$C_2 x_{2,2} + C_3 x_{3,2} + C_4 x_{4,2} \geq R_2 = 6 \tag{5.53}$$

$$C_3 x_{3,3} + C_4 x_{4,3} + C_5 x_{5,3} \geq R_3 = 6 \tag{5.54}$$

$$C_2 x_{2,4} + C_5 x_{5,4} + C_6 x_{6,4} + C_7 x_{7,4} + C_8 x_{8,4} + C_9 x_{9,4} \geq R_4 = 9 \tag{5.55}$$

where C_i represents the credit value for course i.

5.2.1.4. Prerequisite constraints.

$$x_{1,1} - x_{2,2} - x_{2,4} \geq 0 \tag{5.56}$$

$$x_{1,1} - x_{3,2} - x_{3,3} \geq 0 \tag{5.57}$$

$$x_{2,2} + x_{2,4} - x_{4,2} - x_{4,3} \geq 0 \tag{5.58}$$

$$x_{2,2} + x_{2,4} - x_{5,3} - x_{5,4} \geq 0 \tag{5.59}$$

$$x_{3,2} + x_{3,3} - x_{5,3} - x_{5,4} \geq 0 \tag{5.60}$$

$$x_{5,3} + x_{5,4} - x_{8,4} \geq 0 \tag{5.61}$$

$$x_{5,3} + x_{5,4} - x_{9,4} \geq 0 \tag{5.62}$$

$$x_{4,2} + x_{4,3} - x_{7,4} \geq 0 \tag{5.63}$$

$$x_{4,2} + x_{4,3} - x_{8,4} \geq 0 \tag{5.64}$$

$$x_{4,2} + x_{4,3} - x_{6,4} \geq 0 \tag{5.65}$$

5.2.1.5. Course preference constraint.

$$x_{9,4} \geq 1 \tag{5.66}$$

5.2.1.6. Prior course constraint.

$$x_{3,2} + x_{3,3} \geq 1 \tag{5.67}$$

5.2.1.7. Total credit constraint.

$$F(x) = \sum_i \sum_j C_i x_{i,j} \geq 24 \tag{5.68}$$

5.2.1.8. Objective function.

$$\text{Minimize } F(x) = \sum_i \sum_j C_i x_{i,j} \tag{5.69}$$

5.2.1.9. Solution. The selections are presented in Table 5.4. For the remaining results regarding semester and module selection, the first course selection from Table 5.2 is used.

5.2.2. Semester Selection. With the first selection from Table 5.4 chosen, the prioritized course list is different as shown in Figure 5.4, but the process is the same as in

Table 5.4. Course Selection Solutions

Requirement	**Solutions**					
	1	2	3	4	5	6
1	1	1	1	1	1	1
2	2, 4	2, 3	2, 3	2, 3	2, 4	2, 4
3	3, 5	4, 5	4, 5	4, 5	3, 5	3, 5
4	6, 7, 9	6, 8, 9	7, 8, 9	6, 7, 9	7, 8, 9	6, 8, 9

scenario 5.1. Every semester's max credits is once again 6. Semester 1 will be treated as the next semester while all courses are unavailable in semesters 3 and 6. The previously taken course 3 was taken in semester 0.

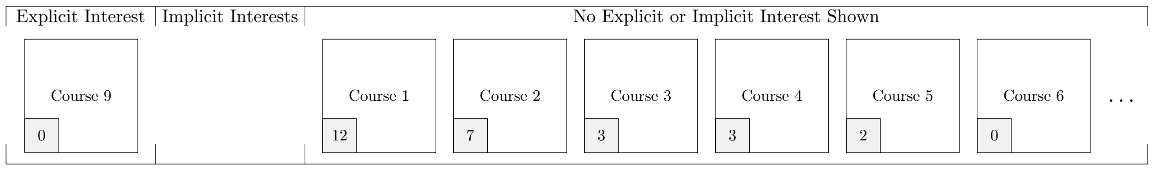

Figure 5.4. Priority List

5.2.2.1. Iteration 1. The schedule after iteration 1 appears as follows:

Semesters							
0	1	2	3	4	5	6	7
3	1						

5.2.2.2. Iteration 2. The schedule after iteration 2 appears as follows:

Semesters							
0	1	2	3	4	5	6	7
3	1	2					

5.2.2.3. Iteration 3. All courses are courses are unavailable - no change.

5.2.2.4. Iteration 4. The schedule after iteration 4 appears as follows:

Semesters							
0	1	2	3	4	5	6	7
3	1	2		4			
				5			

5.2.2.5. Iteration 5. The schedule after iteration 5 appears as follows:

Semesters							
0	1	2	3	4	5	6	7
3	1	2		4	9		
				5	6		

5.2.2.6. Iteration 6. All courses are courses are unavailable - no change.

5.2.2.7. Iteration 7. The schedule after iteration 7 appears as follows:

Semesters							
0	1	2	3	4	5	6	7
3	1	2		4	9		7
				5	6		

Note that if Module 2 happened to have been taken previously, then the time-to-degree for this student would have been shortened by two semesters, but instead there is a smaller workload in semester 2.

5.2.3. Module Selection. Module selection for Student 2 is also similar to module selection Scenario 1 except the problem is further constrained to use modules 18, 28, 29, and 30, as well as 9, 10, 11, and 12 from having previously taken course 3.

5.2.3.1. Decision variables.

$$X = \{x_{i,j} : x_{1,1}, x_{2,1}, x_{3,1}, x_{4,1}, x_{5,2}, x_{6,2}, x_{7,2}, x_{8,2}, x_{9,3}, x_{10,3}, x_{11,3}, x_{12,3}, x_{13,4}, x_{14,4}, x_{15,4}, x_{16,4}, x_{9,5}, x_{17,5}, x_{18,5}, x_{13,6}, x_{19,6}, x_{20,6}, x_{21,6}, x_{15,7}, x_{22,7}, x_{23,7}, x_{24,7}, x_{18,9}, x_{28,9}, x_{29,9}, x_{30,9}\} \tag{5.70}$$

where i = module, j = course.

5.2.3.2. Course reusability constraints.

$$x_{9,3} + x_{9,5} \leq 1 \tag{5.71}$$

$$x_{13,4} + x_{13,6} \leq 1 \tag{5.72}$$

$$x_{15,4} + x_{15,7} \leq 1 \tag{5.73}$$

$$x_{18,5} + x_{18,9} \leq 1 \tag{5.74}$$

5.2.3.3. Required module constraints.

$$x_{1,1} \geq 1 \tag{5.75}$$

$$x_{2,1} \geq 1 \tag{5.76}$$

$$x_{5,2} \geq 1 \tag{5.77}$$

$$x_{6,2} \geq 1 \tag{5.78}$$

$$x_{10,3} \geq 1 \tag{5.79}$$

$$x_{14,4} \geq 1 \tag{5.80}$$

$$x_{19,6} \geq 1 \tag{5.81}$$

$$x_{22,7} \geq 1 \tag{5.82}$$

$$x_{28,9} \geq 1 \tag{5.83}$$

$$x_{9,3} + x_{9,5} \geq 1 \tag{5.84}$$

$$x_{13,4} + x_{13,6} \geq 1 \tag{5.85}$$

$$x_{15,4} + x_{15,7} \geq 1 \tag{5.86}$$

$$x_{17,5} \geq 1 \tag{5.87}$$

$$x_{18,5} + x_{18,9} \geq 1 \tag{5.88}$$

5.2.3.4. Course credit constraints.

$$M_1\underline{x_{1,1}} + M_1\underline{x_{2,1}} + M_1x_{3,1} + M_1x_{4,1} \geq C_1 = 3 \tag{5.89}$$

$$M_2\underline{x_{5,2}} + M_2\underline{x_{6,2}} + M_2x_{7,2} + M_2x_{8,2} \geq C_2 = 3 \tag{5.90}$$

$$M_3\underline{x_{9,3}} + M_3\underline{x_{10,3}} + M_3x_{11,3} + M_3x_{12,3} \geq C_3 = 3 \tag{5.91}$$

$$M_4\underline{x_{13,4}} + M_4\underline{x_{14,4}} + M_4\underline{x_{15,4}} + M_4x_{16,4} \geq C_4 = 3 \tag{5.92}$$

$$M_5\underline{x_{9,5}} + M_5\underline{x_{17,5}} + M_5\underline{x_{18,5}} \geq C_5 = 3 \tag{5.93}$$

$$M_6\underline{x_{13,6}} + M_6x_{19,6} + M_6x_{20,6} + M_6\underline{x_{21,6}} \geq C_6 = 3 \tag{5.94}$$

$$M_7\underline{x_{15,7}} + M_7x_{23,7} + M_7x_{24,7} + M_7\underline{x_{25,7}} \geq C_7 = 3 \tag{5.95}$$

$$M_9\underline{x_{18,9}} + M_9\underline{x_{28,9}} + M_9x_{29,9} + M_9x_{30,9} \geq C_9 = 3 \tag{5.96}$$

where M_i represents the credit value for module i and underlined decision variables represent those modules that are required.

5.2.3.5. Module preference constraint.

$$x_{29,9} \geq 1 \tag{5.97}$$

$$x_{30,9} \geq 1 \tag{5.98}$$

Note that constraints for modules 18 and 28 do not need to be added because they are already required.

5.2.3.6. Prior module constraints.

$$x_{11,3} \geq 1 \tag{5.99}$$

$$x_{12,3} \geq 1 \tag{5.100}$$

Note that constraints for modules 9 and 10 do not need to be added because they are already required.

5.2.3.7. Total credit constraint.

$$F(x) = \sum_i \sum_j M_i x_{i,j} \geq 24 \tag{5.101}$$

5.2.3.8. Objective function.

$$\text{Minimize } F(x) = \sum_i \sum_j M_i x_{i,j} \tag{5.102}$$

5.2.3.9. Solution. The solution is presented in Table 5.5.

Table 5.5. Module Selection Solution

Course	**Module(s) Selected**
1	1, 2, 3
2	5, 6, 7
3	10, 11, 12
4	13, 14, 15
5	9, 17, 18
6	19, 20, 21
7	22, 23, 24
9	28, 29, 30

6. CONCLUSION

The primary contribution of this thesis was a sound methodology for helping students and institutions overcome the course selection problem by automating the process of selecting a schedule for students. With respect to this goal, this thesis showed how the PERCEPOLIS CSA generates schedules that are more likely to result in higher performance and adhere to institution rules and the student's interests and with an eye towards shortening the time-to-degree.

This thesis presented the course selection algorithm within PERCEPOLIS, beginning in Section 2 with background research in ontology, context-aware recommender systems, semantic similarity, and course selection. The overall goal of the Section 2 was to give the reader a foundation sufficient enough to understand the course selection problem and how the CSA within PERCEPOLIS attempts to solve it, especially with respect to other related works. Section 3 contains an in-depth look at what PERCEPOLIS was envisioned to be in [2], the goal of which is to make it clear how the contributions of this thesis impact the larger project. All of this establishes a backdrop for Section 4 where the research contributions were presented and the methodology of PERCEPOLIS's course selection algorithm were discussed in detail. Once the algorithm was presented, results of execution under varying conditions were shown in Section 5.

One avenue for expanding this work lies within personalizing course schedules for the students who do not specify their interests. A CSA could look at previous students who were similar to the target in order to find opportunities to personalize and reduce the number of decision variables. For example, elective courses in wich similar students did especially well might be worth recommending to a target student, but if only if taken in a certain semester or accompanied by some other course. What constitues similarity is also an open question, but getting it correct is hypothesized to leader to higher student satisfaction even though it requires a long time to measure. However, different similarity metrics could be compared in the short term if applied to module selection instead.

A final promising extension to this work are those where the course selection problem is solved as a multi-objective optimization problem. Multi-objective optimization problems seem to be a far more natural representation of the course selection problem since each AOE is really a competing objective, and the AOE's mentioned in this thesis are not all there are to consider. This thesis handles this problem by breaking the problem up into stages, but exploring more sophisticated solutions in evolutionary algorithms seems promising as it would allow the simultaneously consideration of multiple objective functions, such as those for expected performance (if an appropriate prediction is used) and interests. Considering multiple objective functions immediately allows the researcher to effectively consider each AOE within the context of the others, rather than intelligently one at a time as presented in this thesis. If the multi-objective optimization approach were adopted, the number of AOE's could be expanded in exciting ways to include accommodating deficiencies, preferred learning styles, and special needs. Moreover, constraint derivation for the semester selection stage could be replaced with an objective function and intelligent constraint derivation for the prerequisite, corequisite, and consequisite relationships.

APPENDIX

The following program provides an example usage of Word2Vec as described in Section 4.3.2.

```
"""interest_matching.py"""
import argparse
import pprint as pp
import word2vec

CORPUS_PATH = "data/corpus"
CORPUS_PHRASES_PATH = "data/corpus-phrases"
CORPUS_BINARIES_PATH = "data/corpus-binaries.bin"
CORPUS_CLUSTERS_PATH = "data/corpus-clusters.txt"

def train():
    """Convert similar words to phrases and train the model.
    The clusters of vectors in the model are also output.
    It is unnecessary to train every time a prediction is
    desired. Once a model is obtained, it can be reused over
    and over again."""
    word2vec.word2phrase(CORPUS_PATH, CORPUS_PHRASES_PATH)
    word2vec.word2vec(CORPUS_PHRASES_PATH,
                      CORPUS_BINARIES_PATH,
                      size=100)
    # Continues on next page
    word2vec.word2clusters(CORPUS_PATH,
                           CORPUS_CLUSTERS_PATH,
                           100)


def predict(target_word, model):
    """Obtain the top 10 words most similar to the
    target_word assuming a trained model is available"""
    indexes, metrics = model.cosine(target_word)
    return model.generate_response(indexes, metrics).tolist()


def main():
    """Main function that parses command line arguments"""
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "word_or_phrase",
        help="The word or phrase on which to predict")
    parser.add_argument(
        "--train",
        action="store_true",
        help="Train a model first")
    parser.add_argument(
        "--tracebacks",
        action="store_true",
        help="Show full tracebacks")
    args = parser.parse_args()

    # continues on next page
    try:
        if args.train:
            train()
        model = word2vec.load(CORPUS_BINARIES_PATH)
        responses = predict(args.word_or_phrase, model)
        pp.pprint(responses)
    except Exception as exc:
        if args.tracebacks:
            raise exc
        else:
            print(exc)
        raise SystemExit(1) from exc

if __name__ == "__main__":
    main()
```

BIBLIOGRAPHY

[1] S. M. Ross, G. R. Morrison, and D. L. Lowther, “Educational Technology Research Past and Present: Balancing Rigor and Relevance to Impact School Learning,” *Contemporary Educational Technology*, vol. 1, no. 1, pp. 17–35, 2010.

[2] A. R. Hurson and S. Sedigh, “PERCEPOLIS: Pervasive Cyberinfrastructure for Personalized Learning and Instructional Support,” *Intelligent Information Management*, vol. 2, pp. 583–593, Oct. 2010.

[3] T. Morrow, S. S. Sarvestani, and A. R. Hurson, “Pervasive Cyberinfrastructure for Personalized Education,” in *Handbook of Research on Applied Learning Theory and Design in Modern Education*, vol. 2, pp. 817–839, Hershey, PA: IGI Global, 2016.

[4] “Blackboard.” http://www.blackboard.com/.

[5] “Canvas by Instructure.” https://www.canvaslms.com/.

[6] “Khan Academy.” http://www.khanacademy.org.

[7] “edX.” https://www.edx.org/.

[8] “Coursera.” https://www.coursera.org/.

[9] “Knewton.” https://www.knewton.com/.

[10] B. Obama, ed., *A Strategy for American Innovation: Driving Towards Sustainable Growth and Quality Jobs*. DIANE Publishing, 2011.

[11] D. DeSilver, “Before Obama’s last State of the Union, a Look Back at His Early Hopes,” Jan. 2016.

[12] Pew Research, “Economy, Jobs Trump All Other Policy Priorities In 2009,” Jan. 2009.

[13] Pew Research, “Budget Deficit Slips as Public Priority,” Jan. 2016.

[14] National Center for Education Statistics, “Undergraduate Enrollment.” https://nces.ed.gov/programs/coe/indicator_cha.asp, May 2016.

[15] U.S. Department of Education, Institute of Education Sciences, National Center for Education Statistics, “Undergraduate Retention and Graduation Rates,” tech. rep., May 2016.

[16] N. A. Raisman, *From Admissions to Graduation*. Administrator’s Bookshelf, 2014.

[17] P. C. Brown, H. L. Roediger, and M. A. McDaniel, *Make It Stick*. Harvard University Press, 2014.

[18] J. B. Treaster, "Will You Graduate? Ask Big Data," *The New York Times*, Feb. 2017.

[19] California State University (CSU) Chancellor's Office, "E-Advising - Course Redesign with Technology." http://courseredesign.csuprojects.org/wp/eadvising/, Jan. 2017.

[20] F. Stempler, "Students Struggle to Graduate Due to the ICC." https://theithacan.org/news/students-struggle-to-graduate-due-to-the-icc/, Jan. 2017.

[21] V. Swaminathan and R. Sivakumar, "A Comparative Study of Recent Ontology Visualization Tools with a Case of Diabetes Data," *International Journal of Research in Computer Science*, vol. 2, no. 3, p. 31, 2012.

[22] M. Baldauf, S. Dustdar, and F. Rosenberg, "A Survey on Context-Aware Systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[23] A. Maedche and S. Staab, "Ontology Learning for the Semantic Web," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 72–79, 2001.

[24] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, "A Survey of Context Modelling and Reasoning Techniques," *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, 2010.

[25] M. Knappmeyer, N. Baker, S. Liaquat, and R. Tönjes, "A Context Provisioning Framework to Support Pervasive and Ubiquitous Applications," in *Smart Sensing and Context*, pp. 93–106, Springer, 2009.

[26] D. Ejigu, M. Scuturici, and L. Brunie, "An Ontology-Based Approach to Context Modeling and Reasoning in Pervasive Computing," in *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference On*, pp. 14–19, IEEE, 2007.

[27] A. Badii, M. Crouch, and C. Lallah, "A Context-Awareness Framework for Intelligent Network Embedded Systems," pp. 105–110, IEEE, 2010.

[28] T. Strang and C. Linnhoff-Popien, "A Context Modeling Survey," in *Workshop on Advanced Context Modelling, Reasoning and Management*, (Nottingham, England), 2004.

[29] F. Ricci, L. Rokach, and B. Shapira, *Recommender Systems Handbook.* Springer, 2 ed., 2015.

[30] F. Ricci, L. Rokach, and B. Shapira, "Recommender Systems: Introduction and Challenges," in *Recommender Systems Handbook*, pp. 1–34, Springer, 2 ed., 2015.

[31] M. Erdt, A. Fernandez, and C. Rensing, "Evaluating Recommender Systems for Technology Enhanced Learning: A Quantitative Survey," *Learning Technologies, IEEE Transactions on*, vol. 8, pp. 326–344, Oct. 2015.

[32] G. Adomavicius and A. Tuzhilin, "Context-Aware Recommender Systems," in *Recommender Systems Handbook*, pp. 191–226, Springer, 2015.

[33] K. Verbert, N. Manouselis, X. Ochoa, M. Wolpers, H. Drachsler, I. Bosnic, and E. Duval, "Context-Aware Recommender Systems for Learning: A Survey and Future Challenges," *IEEE Transactions on Learning Technologies*, vol. 5, pp. 318–335, Oct. 2012.

[34] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context Aware Computing for the Internet of Things: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[35] M. Bright, A. R. Hurson, and S. Pakzad, "Automated Resolution of Semantic Heterogeneity in Multidatabases," *ACM Transactions on Database Systems (TODS)*, vol. 19, no. 2, pp. 212–253, 1994.

[36] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv preprint arXiv:1301.3781*, 2013.

[37] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural Language Processing (Almost) From Scratch," *Journal of Machine Learning Research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[38] C. Fellbaum, *WordNet*. Wiley Online Library, 1998.

[39] T. Pedersen, S. Patwardhan, and J. Michelizzi, "WordNet: Similarity: Measuring the Relatedness of Concepts," in *Demonstration Papers at HLT-NAACL 2004*, pp. 38–41, Association for Computational Linguistics, 2004.

[40] W. S. Murray and L. A. Le Blanc, "A Decision Support System for Academic Advising," in *Proceedings of the 1995 ACM Symposium on Applied Computing*, pp. 22–26, ACM, 1995.

[41] W. S. Murray, L. A. LeBlanc, and C. T. Rucks, "A Decision Support System for Academic Advising," *J. End User Comput.*, vol. 12, pp. 38–49, July 2000.

[42] L. A. Le Blanc, C. T. Rucks, and W. S. Murray, "A Decision Support System for Prescriptive Academic Advising," in *Advanced Topics in End User Computing*, vol. 1, pp. 263–284, IGI Global, 2002.

[43] R. R. Hashemi and J. Blondin, "SASSY: A Petri Net Based Student-Driven Advising Support System," in *Information Technology: New Generations (ITNG), 2010 Seventh International Conference On*, pp. 150–155, IEEE, 2010.

[44] J. L. Peterson, *Petri Net Theory and the Modeling of Systems*. Prentice Hall, 1 ed., 1981.

[45] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, vol. 6. MIT press Cambridge, 2001.

[46] R. Lougee-Heimer, “The Common Optimization INterface for Operations Research,” *IBM Journal of Research and Development*, vol. 47, no. 1, pp. 57–66, 2003.

[47] A. Mohamed, “A Decision Support Model for Long-Term Course Planning,” *Decision Support Systems*, vol. 74, pp. 33 – 45, 2015.

[48] B. Everitt, S. Landau, and M. Leese, *Cluster Analysis*. Hodder Arnold Publication, Wiley, 5 ed., 2001.

[49] R. Xu and D. Wunsch, *Clustering*, vol. 10. John Wiley & Sons, 2008.

[50] C. S. Wagner, S. S. Sarvestani, and A. R. Hurson, “Algorithms and Techniques for Proactive Search,” *Journal of Information Processing*, vol. 22, no. 3, pp. 425–434, 2014, invited.

VITA

Tyler Joseph Morrow was born and raised in St. Peters, MO, a suburb of St. Louis, MO. He first earned his Bachelor's degree in Computer Science from Missouri University of Science & Technology in May 2014. He then received his Master's in Computer Science from Missouri University of Science & Technology in May 2017. During his time as a graduate student, Tyler taught two introductory computer science courses. The first was an introductory laboratory course in C++ and the other was an expiremental course on contemporary programming languages that was co-developed and co-taught with another graduate student. In addition to teaching and research, Tyler performed cooperative study with Sandia National Labs.