A Constraint Processing Approach to Assigning Graduate Teaching Assistants to Courses

Robert Glaubius
Honors Thesis
Constraint Systems Laboratory
Department of Computer Science and Engineering
University of Nebraska-Lincoln
glaubius@cse.unl.edu

December 23, 2001

Abstract

In recent years, scheduling problems have been advantageously modeled as Constraint Satisfaction Problems (CSPs) and effectively solved by constraint propagation and processing techniques. In this document, we discuss one such application: the assignment of graduate teaching assistants (GTAs) to courses as instructors or grading assistants in a university setting. We analyze and model this problem in the real-world setting of the Department of Computer Science and Engineering of the University of Nebraska-Lincoln. This problem has traditionally been over-constrained. The results of our investigations are as follows. We enrich the definition of the problem by including students' own preferences to be assigned to courses. We propose, implement, and evaluate a representational model of the entities and constraints that constitute this problem, and explain how the current model has emerged from, and improves upon, a series of modeling attempts. We also propose, implement, and test various optimization criteria that model 'satisfactory' solutions. Finally, we describe, implement and test various computational mechanisms for solving this problem such as ordering heuristics, constraint propagation mechanisms, and search mechanisms.

Contents

1 Introduction

Constraint Satisfaction has emerged as a powerful paradigm for modeling and solving large combinatorial problems. One of the earliest application domains has addressed scheduling problems [4]. Today, and less than 15 years after these initial efforts, commercial companies such as Ilog (http://www.ilog.com/), On Time Systems (http://www.cirl.uoregon.edu/otsys/), and J.D. Edwards (http://www.numetrix.com/) are successfully commercializing this technology. In this document, we discuss a specific application of constraint satisfaction techniques to a real-world application. This is the assignment of graduate teaching assistants to courses in the Computer Science department of the University of Nebraska at Lincoln.

There are multiple goals to our study. The idea for this particular application is borrowed from Rina Dechter, at the University of California, Irvine. The initial motivation for our investigations was the collection of data in order to articulate interesting homework topics for CSCE476/876 and CSE421/821. We soon realized that what at first appeared to be a relatively simple problem was actually quite complex.

One of our intents is to define and carry out a challenging thesis motivated by a real-world application. In this vein, we seek to understand and evaluate the application of theoretical concepts that exist in the literature, while gaining insight into the research experience.

We also undertake this research in order to aid our department in this task. This task is a difficult and time-consuming undertaking as it is currently performed within the department. It is carried out manually each semester, and involves at least three administrators (i.e. the department chair, the vice-chair, and the Graduate Program secretary) in addition to the faculty and students involved. In addition to the disadvantage of a large and painful investment of effort, the results of this process tend to be less than satisfactory[1] We expect that automation of this process will facilitate the discovery of substantially less problematic solutions.

A final important research objective is the identification of directions for future research that is relevant to real-world applications. Such direction can only serve to build and strengthen tools for critical applications, and benefit the community as a whole.

Before we delve into the details of the data collection phase of our system development, we introduce the reader to the terminology of constraint satisfaction problems then explain the structure of this report.

1.1 Definitions

A Constraint Satisfaction Problem (CSP) is represented as a tuple $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, where $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ is a set of variables, and $\mathcal{D} = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\}$ is the set variable domains, such that D_{V_i} is the do-

[1]GTAs are often assigned to help with courses that have time conflicts with their other commitments, or assigned to courses in which they have inadequate proficiency, among other difficulties.

main of variable V_i. $\mathcal{C} = \{C_i, C_{j,k}, \ldots, C_{i,j,\ldots,m}, \ldots C_n\}$ is a set of constraints such that $C_{i,j}$ indicates a constraint between variables V_i and V_j. For a given constraint $C_{i,j,\ldots,m}$, the set of variables $V_i, V_j, \ldots, V_m$ is called the constraint's *scope* and the size of this set is the constraint's *arity*. Formally a CSP is defined as follows. Given the tuple $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, find an assignment to each $V_i \in \mathcal{V}$ from its domain D_{V_i} such that no constraint is violated.

A constraint C_{V_i,V_j} specifies the allowable tuples that can be assigned to the variables $V_i and V_j$. The set of all such allowable tuples constitute the constraint's definition. If all allowable tuples are completely enumerated and stored, the constraint is *extensively* defined. If allowable tuples are functionally represented, the constraint is *intensively* defined.

An *instantiation* to a variable V_i is the assignment of value $a \in D_{V_i}$ to V_i. We write such an instantiation as an ordered pair (V_i, a). An instantiation that does not violate any constraints is consistent it; otherwise, it is inconsistent. A solution to a CSP is a set of instantiations $\{(V_i, a), (V_j, b), \ldots, (V_k, x)\}$. We use the term *global* solution to describe a solution that instantiates all variables; if some but not all variables are instantiated, then it is a *partial* solution.

For a specific example of a constraint satisfaction problem, we visit the map coloring problem. The map coloring problem is defined as follows. Given a set of states, and a set of x different colors, color each state such that no two states that share a border have the same color. An example of such a problem, and a subsequent mapping to a CSP, are given in the following example.

Example 1.1 Imagine that we want to color the map of the US, as shown in figure 1 with the 5 colors red, blue, green, gold, and orange.

Each state must have one color and no two neighboring states can have the same color. Additionally, suppose we have the following restrictions:

- NE must be colored red.
- FL must be colored either green or orange.
- Exactly one of NV, CA, or OR must be colored gold.

We model this problem instance as a CSP choosing as variables the states on the map:

$$\mathcal{V} = \{CA, OR, WA, ID, NV, AZ, \ldots\} \tag{1}$$

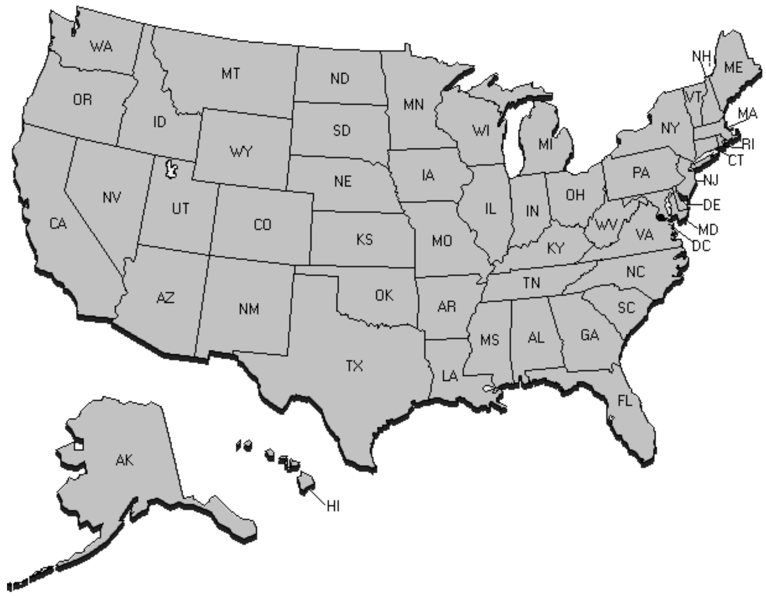

Figure 1: Map of the United States of America.

Each of these variables has the set of colors {red, blue, green, gold, orange} as its domain. For example:

$$D_{NM} = \{red, blue, green, gold, orange\} \tag{2}$$

The constraints in the problem specify that no two adjacent states can have the same color. For example, there would be a binary constraint between GA and AL :

$$C_{GA,AL} = \{((GA, x), (AL, y)) \mid (x \in D_{GA}) \wedge (y \in D_{AL})(\wedge x \neq y)\} \tag{3}$$

We also model each of the restrictions listed above as a constraint. The first and second ones are unary constraints:

$$C_{NE} = \{(NE, red)\} \tag{4}$$

$$C_{FL} = \{(FL, green), (FL, orange)\} \tag{5}$$

The last is a ternary constraint:

$$\begin{aligned} C_{NE,CA,OR} \quad = \{((NE, x), (CA, y), (OR, z)) \mid \quad & (x = gold \wedge y \neq gold \wedge z \neq gold) \vee \\ & (x \neq gold \wedge y = gold \wedge z \neq gold) \vee \\ & (x \neq gold \wedge y \neq gold \wedge z = gold)\} \end{aligned} \tag{6}$$

1.2 Overview

This document is structured as follows. In Section 2, we discuss the data collection process. In particular, we identify the attributes of courses and graduate teaching assistants that need to be specified for the task of assigning GTAs to courses. In Section 3, we report on our strategies for modeling GTAs, courses, and the constraints that dictate under what conditions we were able to assign a given GTA to a particular course. In Section 4, we discuss the constraint processing techniques we have considered for solving this problem. Section 5 describes our current implementation. Section 6 presents and discusses the results of our experiments on two data sets pertaining to the academic semesters Spring 2001 and Fall 2001. Finally, Section 7 concludes this documents and draws directions for future research.

2 Data Collection

In this section, we discuss our methods for data collection. This includes the methods in place when we began our study, our modifications to this process, as well as a description of the collected data. The type of data collected heavily impacts the types of relationships we can model. For a parallel, in the map coloring example earlier, if we were given the name of the states to be colored, but not their relative locations and borders, it would not be possible to model the problem in a meaningful fashion. By this same token, without acquiring key data for our problem, it is impractical to attempt to solve a model of the problem, as it will bear little or no resemblance to the actual problem seen in practice.

2.1 Background

Previous to our study, when a graduate student applies for a teaching assistantship (TAship) in the Department of Computer Science and Engineering (CSE) at the University of Nebraska-Lincoln (UNL), he or she is asked to submit information about his or her academic history. The department has designed paper forms that the students fill to provide this information. A sample form is included as an appendix to this paper. Once assistantships are awarded, the information submitted by each graduate teaching assistant (GTA) is then used, in addition to other information such as ITA certification status and previous TA experience. This data is then used to determine the courses that the student may potentially be assigned to.

Further, depending upon the availability of funds or their involvement in research projects, graduate students may receive one of two types of teaching assistantships. These are half or full teaching assistantships.

Typically, GTAs that receive a half TAship are assigned to fewer or smaller courses, while students with full TAships receive a more demanding workload.

With the above information in hand, faculty and administration members then proceed to manually assign the GTAs to scheduled courses. Typically this procedure results in barely satisfactory assignments. GTAs frequently have time conflicts with courses that they are assigned to and requiring their physical attendance. Sometimes, they are inadequately prepared for the material of their assigned courses. These conflicts are sometimes detected and GTAs are swapped between courses or brought up to speed. The detection and correction of these conflicts result in frustration and a loss of time and effort for for the students enrolled in the classes, the GTAs, the teaching faculty, and the administrators.

Our goal in automating this task is to reduce the quantity and magnitude of conflicts observed in practice, while proposing solutions that better satisfy the involved parties – the faculty, students, and the GTAs themselves.

At the time that we began this project, the information listed was collected by the department for each GTA. The majority of these items are briefly mentioned here.

- *Enrollment status:* A list of courses that a GTA is enrolled in.
- *ITA Certification:* A boolean value that indicates whether an international student has or has not passed the International Teaching Assistant Evaluation.
- *Half/Full TAship:* Indicates whether a GTA is employed full-time or half-time.
- *Faculty advisor:* The GTA's individual advisor.
- *Course deficiencies:* Courses required for completion of the graduate program that have yet to be taken.
- *Past TA experience:* A list of courses that the GTA has been assigned to in previous semesters.
- *Grade point average (GPA):* An indicator of scholastic aptitude.

Of the above, a GTAs advisor, deficiencies, experience, and GPA are not taken into account in the current version of the problem model. The rest of these attributes are discussed in Section 3.

2.2 Refining data collection: preferences

We observed that the data collected by the department from the students thus far is actually sufficient to avoid conflicting decisions such as the ones mentioned above. Consequently, it is quite realistic to implement an automated system that monitors decisions to avoid the generation of conflicting assignments. We are able to utilize past procedures for data collection for a large portion of relevant data, including course attributes such as meeting times and types, as well as GTA information such as ITA certification and enrollment status.

The information and mechanisms for evaluating how satisfactory a given solution is and for rating and comparing distinct solutions, however, were not existent at the outset of this study. For this reason, we introduced a range of six (6) integer values, from 0 to 5, and requested students to rate each offered course using this scale. A rating of 5 indicates a strong preference for a course, while 0 indicates that the GTA has a demonstrable justification for why he or she should not be assigned to the course.

This mechanism provides us with a basis for comparing two consistent solutions, thus otherwise equally acceptable. We designed an optimization function that uses this preference schema to better fit the preference of GTAs with their actual assignments, see Section 3.4. We expect that matching GTAs to courses for which they declare a stronger preference will enhance the quality of their performance in their duties.

We generated a supplement to the existing questionnaire for students to fill out shown in the Appendix. The collected data now has the information necessary for designing a representation of the problem, implementing it and processing it on a computer. We choose the CSP paradigm for its simplicity, flexibility, and the wealth of propagation methods and processing techniques available. In the next section we specify our constraint satisfaction problem model formulation.

3 Modeling

Many of the challenges inherent to this problem are best addressed during the modeling stage. The CSP paradigm, as introduced in Section 1.1, provides a most natural way to formulate scheduling problems such as the one we address here. In this section, we describe how we formulate the GTA assignment problem during a single academic semester as a CSP. We choose to model courses as variables and GTAs as domain values. Further, we elicit and translate the relationships between GTAs and courses as seen in practice into constraints that can be processed.

3.1 Variables

Each course constitutes a CSP variable and is represented by the following attributes in our model:

- *Course type:* There are three (3) types of courses offered by the CSE department; namely lecture, laboratory, and recitation. A course of a given type may require a GTA as either an instructor or a grading assistant. Some courses may require both. Labs and recitation courses require a graduate student as an instructor, while lectures utilize GTAs as grading assistants in most cases. for example, in the Fall 2001 semester, there were 44 lectures, 25 laboratories, and 3 recitations.

- *Course duration:* The duration of most courses offered during a semester spans over the entire length of the semester. However, a few exceptions exist. There are some courses that occupy only either the first or second half of a semester. These courses typically require both one GTA as grading assistant and another GTA as instructor. For example, in the Fall 2001 semester, there were 37 full-semester courses and 7 half semester courses, 4 of which were held during the first half, and the rest during the second half.

- *Meeting times:* It is important to include the meeting times of each course in order to avoid time conflicts in the assignments. For example, a GTA assigned as instructor to a course must be able to attend all sessions of this course and thus should not be enrolled in another course that overlaps with the course he or she is assigned to. The course schedule is given as input to the system. Each course has one, two, or three sessions per week. For example, in the Fall 2001 semester, CSCE310 was held on Mondays, Wednesdays and Fridays between 9:30 a.m. and 10:20 a.m.

- *Expected load:* The load of the course is a relative measure of the amount of effort that a GTA is expected to spend on the course in question. This value is determined by department staff members, based on anticipated enrollment in the course, as well as the typical amount of course work. Typical values for courses are 0.5 and 1, where a course with 0.5 load is a course with lower enrollment and small course work expectations, and a course with load of 1 has a larger number of students enrolled, and may require a moderate to large amount of course work.[2] For example, during the Fall 2001 semester, CSCE155, an introductory level course in computer science, had load of 1, while CSCE496/896, a senior-level special topics course, had load of 0.5. This reflects the fact that CSCE155 tends to have enrollment of more than 100 students, while CSCE496/896 is likely to see between 20 and 30 students.

In general, each course is represented by a single variable within the CSP. More specifically, each variable represents the task an assigned GTA is expected to perform with respect to a given course. For this reason, half semester courses requiring a GTA as an instructor, and another one as a grading assistant are

[2] A typical semester will have several courses with 0 load as well. This indicates that these courses do not require the services of a GTA, and therefore are not considered for assignment in the model. These courses tend to be graduate level courses (900 level) or seminar sections that have relatively low enrollment, and the advanced nature of the course precludes the majority of GTAs from possessing sufficient proficiency in the area.

represented by two variables. Similarly, classes with load of 0 do not have a variable representation. Typically a given semester in CSE has around 70 variables; for example, the Fall 2001 semester is modeled with 66 variables.

3.2 Domains

The domain of a CSP variable is the set of values that can be assigned to it. In our model, each value is a graduate teaching assistant. Initially, the domain of each variable consists of the pool of all available GTAs. Each GTA has a number of relevant attributes that should be checked before including him or her in the domain of a variable. These include the following:

- *Enrollment:* The enrollment status is a list of courses that a GTA indicates he or she will enroll in during the semester in question. It serves two purposes in our model. First, it is used to determine when a given GTA is not available as an instructor. Second, it is used to prevent a student from being assigned as either a grader or instructor to a course that he or she is actually enrolled in.

- *ITA Certification:* International students constitute the majority of our pool of GTAs. Due to various considerations, it is required for these students to acquire International Teaching Association (ITA) certification prior to assignment as instructors for courses. Incoming international students are usually not ITA certified. This procedure may take a semester or two before an incoming international student becomes ITA certified; in the meanwhile, he or she can be assigned only as a grading assistant. For instance, during the Fall 2001 semester, 14 out of all 34 GTAs hired required but did not possess ITA certification.

- *Half/Full TAship:* As mentioned in Section 2.1, graduate students may receive either a half or full TAship. Typically, it is desirable to rely more directly on GTAs with full teaching assistantships. In practice, a GTA with a half-TAship will be assigned courses with the sum of their loads equal to 1, while a GTA with a half-TAship is assigned a to courses with a load total of 0.5.

- *Preferences:* Preferences, as discussed in Section 2.2, are integer values ranging from 0 to 5 associated with each class for a given GTA. These are used in the optimization criterion that discriminates among consistent solutions.

Each of these attributes is taken into account before assigning a GTA to a given course. The values of these attributes are checked by the constraints, which we will discuss next.

3.3 Constraints

There are several different types of constraints considered when assigning a GTA to a course. This system uses unary (arity = 1), binary (arity = 2), and n-ary constraints to represent the variety of relations that dictate valid instantiations. Several of these constraints were hinted at in previous sections; all constraints that we model are covered in detail here. The unary constraints are:

- *ITA certification* - As touched on in Section 3.2, international students are required to be ITA certified prior to instructing any courses. The ITA certification constraint is a unary constraint that enforces this condition upon any variable that requires a GTA as an *instructor*.
- *Enrollment* - The enrollment constraint is a unary constraint that prevents GTAs from being assigned to courses that they are taking. Clearly it would be disadvantageous to allow a student to grade his or her own work!
- *Overlap* - Overlap constraints prevent any GTA from being assigned as an instructor to a course that he or she cannot attend. This is based on the GTAs enrollment status, using the meeting times of courses he or she is enrolled in to indicate when he or she is unavailable.
- *Zero Preference* - Zero preference constraints prevent any GTA from being assigned to a course for which he or she has indicated preference value zero. This constraint serves to restrict attention only to those GTAs that have some desire (or little objection) to instructing or grading for a given course.

There is only one type of binary constraint used in this model:

- `Mutex` - `Mutex` constraints[3] are binary constraints in place between any two courses that require GTAs as instructors and meet during overlapping times. This enforces the condition that a GTA must attend courses he or she is instructing.

We choose to use non-binary constraints to model the capacity restrictions and containment situations observed in practice. While the majority of processing techniques in CSPs have focussed on binary constraints and relied on reformulation techniques for mapping [1] non-binary formulations into binary ones, it is commonly acknowledged that the representation should remain as faithful to the real-world constraint (i.e., a capacity constraint should be represented as a non-binary constraint) and the processing should use the encoding that is most likely to yield efficient processing. Few investigations have been carried out to

[3] a.k.a. constraints of difference, or coloring constraints.

establish the superiority of a binary versus a non-binary formulation with respect to the efficiency of the processing technique. This is a new research area and no definite conclusions exist yet. We implement 3 types of high arity constraints:

- *Equality* - Equality constraints are n-ary constraints between a set of courses, all of which should be assigned the same GTA.
- *Capacity* - Capacity constraints are n-ary constraints that prevent any GTA g from being assigned more than some maximum load max_g. This is enforced by restricting g's current load $curr_g \leq max_g$, where $curr_g$ is the sum of the load of all courses that g has been assigned to.
- *Containment* - Containment constraints are n-ary constraints, similar in spirit to the aforementioned `mutex` constraint. In general, a number of lab and recitation courses may be associated with a given lecture course; for instance, the lecture course "Introduction to Computer Science I", section 1, has had as many as five associated labs. It is desirable that GTAs assigned to instruct one of these labs not be assigned to any other course, unless it is another lab associated with this same lecture section. Containment constraints enforce this condition. Since a GTA can be assigned to multiple courses, containment constraints enforce assignments of a specific GTA to a subset of labs or recitations associated to a same lecture section after he or she has been assigned to any one of these sections.

In addition to our stated goals – to find good solutions that cover many courses while optimizing solution preference – we have two other goals in mind while modeling this problem. First, we wish to obtain solutions that match the criteria used in practice. Secondly, we want to produce a model that does not force model processing to be too complex. The introduction of many complex constraints can often increase the amount of effort that is expended in finding a solution. We intend our model to be descriptive, yet practical.

In order to do this, we feel that non-binary constraints are the most effective technique for modeling these complex criteria, such as capacity and containment constraints. While the superiority of non-binary over binary representations is still an open question Bacchus:98, the non-binary approach allows an intuitive approach to handling these constraints. The capacity and containment constraints, in particular, are very loose, highly disjunctive constraints, and as such, typical methods for translating binary problems into non-binary ones can become expensive using either an intensive or extensive constraint definition. An initial approach to our problem, that utilized only unary and binary constraints is discussed in Section 3.5.

3.4 Optimization objectives

The goal of modeling is to provide the basis for finding correct, practical solutions. Thus far, we have discussed the aspects of problem modeling that deal with the decision portion of our problem. That is, given the variables, values, and constraints, determine whether or not a solution that satisfies all constraints exists.

The problem of GTA assignment also involves an optimization aspect as well. While a given instance of the GTA decision problem may or may not be satisfiable, we are still required to discover the best solution possible. We optimize solutions with respect to three criteria - consistency, solution size, and GTA course preferences.

We will only select a solution that does not violate any constraints. All constraints in our model are consistent provided that all assigned variables in the scope of each constraint is consistent. This allows solutions to leave some variables unassigned. This is an important consideration, as this problem is often overconstrained. Most algorithms that deal with CSPs will simply exit if no solution that satisfies all constraints and assigns all variables can be found. A notable exception is work by Freuder and Mackworth on MAX-CSP [5].

If consistency were our only consideration, our system would select its initial state, in which no course is assigned a GTA, as a good solution and exit. To resolve this, we optimize the returned solution with respect to solution size. Solution size is defined as the number of instanciated variables.

This begs the question, what if two solutions have the same size? In this case, we rely on the third optimization criteria, GTA preferences. The preference of a solution is the preference of each GTA for the course that he or she is assigned to in a given instantiation. We have experimented with two different measures that optimize solution preference – maximizing the geometric mean and maximizing the minimum preference.

The geometric mean of preferences is straightforward. Given two consistent instantiations with the same solution size, we will select the solution with the greater geometric mean of preferences. This is defined as $\sqrt[n]{\prod_i^n \text{pref}_i}$, where i corresponds to the i^{th} variable V_i that receives an assignment in the current solution. The second optimization measure is maximization of the minimum preference. In this measure, given our two instantiations from earlier, we will select the one with fewer preferences of 1. If there are an equal number of 1's, we'll select the one with fewer 2's, and so on. The following example illustrates each of the optimization criteria discussed here.

Example 3.1 We are given a semester with four courses and four GTAs. From this set, we have several possible solutions.

- *Courses* - CSCE 310, CSCE 155, CSCE 235, CSCE 476
- *GTAs* - Kurt Gödel, Alan Turing, David Hilbert, Alonzo Church

Each GTAs preference for each course is shown in table 1

GTA	155	235	310	476
Alonzo Church	1	2	3	0
Kurt Gödel	3	5	5	2
David Hilbert	2	5	1	3
Alan Turing	5	3	2	5

Table 1: Example set of courses and GTAs with preferences

Given the following two instantiations denoted by tuples (*course, GTA, preference*)

(1) (155, Turing, 5) (235, Hilbert, 5) (310, Gödel, 5) (476, ϕ)

(2) (155, Hilbert, 2) (235, Church, 2) (310, Turing, 5) (476, Gödel, 2)

We will choose instantiation (2), since it covers more classes than (1) does. Comparing the next two instantiations using the maximization of minimum preference evaluation function

(3) (155, Church, 1) (235, Gödel, 5) (310, Hilbert, 1) (476, Turing, 5)

(2) (155, Hilbert, 2) (235, Church, 2) (310, Turing, 5) (476, Gödel, 2)

We will again choose instantiation (2), as instantiation (1) has 2 assignments with preference 1, while instantiation (2) has 0 assignments of preference 1. Finally, comparing the next two instantiations according to the maximization of geometric mean,

(4) (155, Gdel, 3) (235, Turing, 3) (310, Church, 3) (476, Hilbert, 3)

(3) (155, Church, 1) (235, Gödel, 5) (310, Hilbert, 1) (476, Turing, 5)

In this last case, instantiation (4) has geometric mean 3, while (3) has geometric mean of about 2.24. Therefore, we prefer (4) to (3). Notice also that this last pair illustrates the benefit of geometric versus arithmetic mean, as (4) and (3) both have arithmetic mean of 3.

3.5 Previous models

Our model is the result of several prior attempts. While the variables, domains, and optimization criteria have remained fairly unchanged, the constraints have undergone several modifications. The past models can be broken up into two distinct precursors – an initial model with binary constraints, and a previous model using non-binary constraints. Examination of these two models will yield some insight into the design decisions and problems we faced in representing the GTA assignment problem.

3.5.1 Binary model

Early in this project we tried to model this problem using only unary and binary constraints, in order to take better advantage of and build off of existing research. All of the unary and binary constraints in this early model we still utilize in the current model. Containment constraints and equality constraints were not modeled. Capacity constraints were simulated by replacing the current k-ary constraint with a fully-connected graph of `mutex` constraints between all k variables in the simulated capacity constraint's scope. This solves the problem of preventing GTAs from being overloaded; however, it restricts GTAs to at most one assignment. As a workaround, GTAs were then replicated, allowing each GTA to be assigned to a maximum of two courses.

This replication strategy increased the size of the CSP substantially. For a given CSP with $n = |\mathcal{V}|$ and $d = |\mathcal{D}|$, the size, or number of possible solutions of a CSP is d^n. Methods for solving CSPs rely on navigating this search space in order to find a solution; by uniformly doubling domain size, we increase the size of the solution space by a factor of 2^n, with a direct impact on run-time and effectiveness of search. This size problem was the primary motivation for moving to our second, non-binary model.

3.5.2 Initial non-binary model

In the second model, we eliminated the fully-connected mutex networks and replaced them with a non-binary capacity constraint. In addition to maintaining the solution space to d^n size, this also allows a degree of flexibility in specifying workload. Capacity constraints can be given a parameter, `maximum capacity`, that sets a cutoff limit to the amount of load GTAs can be assigned. While we have only utilized capacity constraints with a maximum capacity of two load units for full GTAs, it is possible to configure this maximum to allow GTAs to take larger loads when an insufficient number of GTAs is available, or it can be lowered if there is a surplus of GTAs.

The difference between this non-binary model and the model in use is the inclusion of the equality constraint, and the containment constraint. The addition of these two constraints was motivated by a desire for increased accuracy in fitting to new department strategies.

4 Processing

Generally, problem instances involve around 70 courses and 35 GTAs. Courses tend to have an expected load of about 1 unit, though many have less, and few have load of more than 1. Due to this and many of the constraints on the problem, it is expected that many problem instances will be unsatisfiable. However, it is essential that we find some partial solution in the absence of any global solutions. For this reason, the assignment of the empty value ϕ to a a variable V_i is considered consistent with any constraint $C_{i,j,\ldots,m}$ such that $V_i \in Scope(C_{i,j,\ldots,m})$. In other words, this system necessarily will consider an instantiation to be consistent even when some courses are not assigned GTAs.

We focus our research on the use of systematic techniques for solving CSPs. These methods are based on a systematic, depth-first search of the solution space. We select these methods for their soundness and completeness. Other solution techniques, such as local search, are powerful methods, but lack completeness. As this project presents proof-of-concept, we desire the guarantee to find a consistent global solution, when one exists.

In addition to the solution methods mentioned above, we've experimented with various methods for constraint propagation. These methods attempt to reduce the effort of search by eliminating domain elements in a variable's domain if they are found to be inconsistent with the constraints. For example, enforcement of node consistency is used to propagate the effects of unary constraints prior to search. For instance, if GTA Winston Smith is enrolled Information Retrieval, he will be removed from Information Retrieval's domain when node consistency is enforced due to the enrollment constraint that is in place on the course. In this section, we will first examine some of the propagation methods we have experimented with, and then discuss the solution methods that we have implemented.

4.1 Propagation

The most basic propagation algorithm is node consistency (NC). Node consistency is enforced by examining each value a in the domain of each variable V_i. If a is not consistent with some unary constraint C defined on V_i, a is removed from the domain of V_i. This simple algorithm tends to yield a significant reduction to the size of GTA assignment problems, increasing the speed of finding a solution.

In Section 3.5.1, we referred to an earlier model that represented the capacity constraint as a network of `mutex` constraints. It is worth mentioning here some of our findings in using the arc-consistency propagation method for `mutex` constraints proposed by Régin [10]. This filtering algorithm is based on finding a variable-to-value maximal matching in a fully-connected network of `mutex` constraints. The network of variables is first transformed into a bipartite graph, with one set of nodes representing variables and the other representing values. Edges between these sets tie each variable to the values in its domain. A matching in this graph is a set of edges such that each variable is associated with a unique value [4]. Independent of all other constraints besides the mutices, this matching represents a solution to this set of variables. However, due to the presence of other constraints, Régin's algorithm then uses this initial matching to find all possible matchings. For each value in the graph, if it is not incident upon an edge in any matching, it can be filtered from the problem, as it cannot be included in any global solution.

This algorithm proves useful when we are searching specifically for a global consistent solution. However, since we do not focus on finding a global solution (due to the fact that in many cases they do not exist), this algorithm was not often particularly useful. Once the problem model evolved away from the binary model, and large networks of `mutex` constraints were replaced by capacity constraints, this algorithm was removed from the solver system.

Node consistency and Régin's `mutex` filtering algorithm are the two stand-alone constraint propagation algorithms that we've implemented and used in this system. Other methods for constraint propagation that we have tested are forward-checking [6], and forward-checking for non-binary constraints [2]. These methods are used during search; they will be covered in the next section.

4.2 Solution techniques

We utilize systematic search techniques based on depth-first search to solve the GTA assignment problem. In order to cope with the problem size, we include some look-ahead strategies when searching. Earlier models have used standard forward-checking to propagate the effect of past assignments; the current implementation uses a form of forward-checking for non-binary constraints. Branch-and-bound mechanisms are another feature integrated into our search strategy.

Forward checking (FC) is perhaps best described by Prosser in [9]. This is a strategy used in CSPs containing only unary and binary constraints. Each time an assignment is made, all *future* variables (variables that have not yet been assigned at the current point in search) have values removed from their domains if these values are not consistent with the new assignment.

Alterations to the model that involved the inclusion of non-binary constraints forced a change in the

[4]The algorithm used to compute this matching is attributed to Hopcroft and Karp [8].

look-ahead strategy used. We implement and use forward-checking for non-binary constraints, as discussed in [2]. Specifically, at each assignment, we filter future variables with respect to all constraints that involve the new assignment, and check with respect to all assigned variables in the scope of the constraint. This is an improvement over earlier methods for FC with non-binary constraints that required that all but one variable in the scope of a constraint be assigned before any filtering can occur.

Each of the prior methods attempt to ease computational effort in the average case by eliminating the need to check against past assignments when making new ones. The next augmentation to search, a branch-and-bound mechanism, attempts to reduce computational effort by detecting and rejecting when a partial solution will not yield a better solution than one that's already been encountered.

The branch-and-bound mechanism we include in search on this problem keeps track of the size and evaluation value of the best solution seen up to any point in search. As depth-first search expands nodes in a search path by assigning values to variables, we check to see if the search path being expanded can improve on the current best solution. Once the current best cannot be improved upon, search backtracks until a path that presents the potential for improvement is encountered.

One alternative to the search methods mentioned above is the use of Least Discrepancy Search (LDS). LDS is another type of systematic search that alters the order of node expansion significantly in order to give better coverage of the search space. LDS operates by first expanding every node that is suggested by some variable/value ordering heuristic. If this fails to find an acceptable global solution, it then expands all search paths according to the ordering heuristic, except that it ignores the heuristic at exactly one node. Ignoring this one point is referred to as a *discrepancy*. If this second set of search paths is also unsatisfactory, search then allows 2 discrepancies, and so on up to $|D|$ [7].

Least discrepancy search attempts to navigate around two major difficulties present in systematic search. First, backtrack search is prone to *thrashing*; it tends to spend a large amount of time trying to repair paths that cannot yield a good solution due to mistakes early in search. Second, ordering heuristics are much more discriminative later in search, but tend not to have a significant effect early in search; due to this, mistakes tend to be made early in search. Combined, these difficulties tend to induce a large amount of wasted computation. LDS allows search to reconsider early assignments much sooner than these same assignments would be reconsidered in other systematic searches [7].

Least Discrepancy search was considered prior to the change to a non-binary model. However, it has not yet been adapted to the current non-binary model.

5 Implementation

This system is implemented in ANSI Common Lisp. We rely on the Common Lisp Object System (CLOS) for implementation of the CSP structures (the variables, values, constraints, and the problem itself), as well as the GTAs and courses. We choose this language for the speed of prototyping and the potential the ease of incremental development. Here we discuss the design of GTA, course, and CSP objects, such as variables, values, constraints, and a solution container.

Each course is represented by a class object with slots for each important attribute. GTAs are represented similarly; each GTA object stores an associative list of (course, preference) pairs that are used when building variable domains. These structures are read in from ASCII files and stored in globally-accessible hash tables.

A CSP object is then built, which stores its essential components – the variables, values, and constraints. The CSP object also has slots for storage of a solution, and bookkeeping elements used during search. Variables are stored in the CSP object as a list, as the primary action performed on this list is a sort operation. Values are stored the same way. Constraints are stored in a hash table that is keyed by scope; each entry to the hash table is a list of constraints on the key. The bookkeeping elements stored in a CSP object are the future and past variables; these are lists that operate as stacks during search, and are initialized from the list of variables by one of the variable ordering functions.

The solution structure is used to preserve the instantiation that best satisfies the evaluation function during search. It stores a list of assignments in triples of the form (variable, value, preference). The solution object has several methods that compute the size and evaluation value of the stored instantiation.

The hash table of GTAs is used to build each CSP value. Each CSP value is a wrapper for a GTA structure that serves to provide a named value object. Once this set is constructed, it is stored in the CSP object. The set is also made available to CSP variables as they are constructed, in order to construct each variable's domain.

The CSP variables are then constructed for each course. Variables store their associated course, domain, assignment, and some bookkeeping information for use during search. The domain of each variable is constructed from the pool of available CSP values. The domain is stored as a list of (value, preference) pairs; each pair is drawn from the CSP value's stored GTA object. Assignments made during search are stored in the variable as a pair drawn from the variable's domain. Each variable stores a list of references to constraints involving itself. The variable also stores the following for bookkeeping during search: a list of future-checked variables, and a list of reductions, both for use during forward checking. The list of future-checked variables keeps track of which variables the variable in question has been checked against, and the reductions list acts as a stack of values filtered from the domain (See [9] for a more in-depth discussion of each of these structures).

There are a wide variety of constraint objects in our system. We use an extensive enumeration of valid tuples for the unary constraints, while using an intensive enumeration of the valid tuples for constraints with arity ≥ 1. Extensively defined constraints use a list to store acceptable tuples, and have a generic associated consistency check method. Each intensively-defined constraint uses a specialized consistency check method. This could be remedied by storing a constraint predicate as a member of the constraint object.

6 Experiments

6.1 Data

We studied and experimented with two sets of data obtained from the CSE department. These sets pertained to the Spring and Fall 2001 semesters. GTA half and full TAships and course loads were not defined in our data for Spring 2001. In our tests, we used the default of full TAship, and a default course load of 0.5. Preferences were not provided for the entire Fall 2001 data; in this case we artificially enforced a default preference value of 3 for GTAs who did not report preferences. 20 GTAs did not report preferences in this data set.

	Spring 2001	Fall 2001
GTAs	25	34
Lectures	40	44
Labs	24	24
Recitations	3	3
Half-semester	6	7

Table 2: Description of test data

6.2 Experiment Design

We tested our system's performance on these two data sets using 8 tests per data set. Each test involved one of the two possible evaluation functions, maximizing the minimum or the geometric mean. Values were ordered according to one of two value ordering heuristics, and variables were ordered according to one of two variable ordering heuristics. These heuristics are described briefly below.

We implemented static value ordering heuristics; the first ordered all values by sorting according to preference such that values with high preference would be considered first. The second heuristic considered values that occurred in the fewest domains before those that occurred in many. Ties were broken according to preference, with higher preferences considered first. These heuristics were used in conjunction with either least-domain variable ordering, or domain-degree ratio ordering. Least domain variable ordering considers first the variable with smallest domain, while domain-degree ratio ordering chooses the variable that has the smallest ratio $\frac{|domain|}{|degree|}$, where $degree$ is the number of constraints on the variable.

Each test was run for approximately one hour. We report here the best solution found during that time by each test. Table 3 shows these results in terms of constraint checks, nodes visited, CPU time, and solution quality in terms of solution size and the two evaluation criteria.

Test	CC	NNV	Time	Size	GM	Preferences 1	2	3	4	5
Spring 2001										
GM-DP-LD	167301144	219282	3454900	49	3.7717085	5	1	6	8	29
GM-DP-DD	295782625	87074	2943150	51	3.8908036	3	5	4	6	33
GM-MIN-LD	72185616	51698	1288900	49	2.2945633	16	7	10	4	12
GM-MIN-DD	9889354	4853	155170	52	2.8628445	11	4	11	9	17
MM-DP-LD	166043703	212144	3444040	49	3.7717085	5	1	6	8	29
MM-DP-DD	193803188	50099	1901350	51	3.8908036	3	5	4	6	33
MM-MIN-LD	71184570	50068	1277380	49	2.2945633	16	7	10	4	12
MM-MIN-DD	9176961	3668	124400	52	2.8628445	11	4	11	9	17
Fall 2001										
GM-DP-LD	5886422	112	32690	56	3.167192	7	0	28	0	21
GM-DP-DD	3249132	51	18820	40	3.8350053	1	0	15	6	18
GM-MIN-LD	7094746	1764	65880	51	2.8961947	8	0	28	3	12
GM-MIN-DD	5249870	123	27100	48	2.976746	4	3	29	4	8
MM-DP-LD	5886422	106	31680	56	3.167192	7	0	28	0	21
MM-DP-DD	3249132	51	19340	40	3.8350053	1	0	15	6	18
MM-MIN-LD	6978914	1448	53970	51	2.8961947	8	0	28	3	12
MM-MIN-DD	97282099	51256	1640270	48	3.0200438	3	4	29	4	8

Table 3: Best solution found during one hour interval for each test; each test is described by GM (geometric mean) or MM (maximize minimum) evaluation, DP (preference ordered) or MIN (value that occurs in fewest domain) value ordering, and LD (Least-domain) or DD (domain degree ratio) variable ordering. Categories are CC (constraint checks), NNV (number of nodes visited), CPU time (10ms resolution), GM (geometric mean) and the number of occurrences of each preference.

When viewing the results in table 3, it is important to keep in mind how constraint checks are counted. In the binary case, it is typical to count each constraint check once. However, in the non-binary case, a single count is not likely to be indicative of the amount of work performed. For this reason, every constraint check increments the total number of checks according to the arity of the constraint. For example, checking a 57-ary capacity constraint increases the number of constraint checks by 57.

Our tests indicate that the geometric mean and maximization of minimum preference evaluation functions lead to the same result in most cases. The only exception to this are the GM-MIN-DD and MM-MIN-DD tests. In these tests, the maximization of minimum preferences (MM) evaluation function seems to find these similar solutions with slightly less effort, in terms of node expansion, constraint checks, and CPU time than does the Geometric mean function. Maximizing the minimum was likely able to backtrack earlier, as some path was discarded more quickly under this strategy than it was using maximization of the geometric mean. This allowed the MM test to improve its solution at least one more time before the time limit was reached.

No single variable-value ordering combination appears to dominate in these tests. It appears that the effectiveness of each combination varies strongly between these two test sets. Since variable/value ordering heuristics seem to have a strong effect on the performance of our system, experiments with more powerful orderings is likely to be beneficial.

7 Conclusions and future research

We have shown that it is feasible to use CSPs to model and solve this problem using this approach. There are several apparent improvements that can be made to the system, that we briefly discuss in this section.

This system has been used in practice at the time of this writing, with satisfactory results. We were able to reduce the amount of time spent on this task by a large interval. In practice, scheduling GTAs often would take four or five days before a satisfactory solution was found, as GTAs were shuffled between courses. We were able to generate solutions that reduced the time spent to one day. We intend to further reduce this over the course of further work on this project.

7.1 Future research

There are many possible approaches that may improve the run time performance and accuracy of our system. Below are some of the techniques we intend to employ in future research on this task.

- Aggregation and Reformulation - We expect that the implementation of the methods discussed in section 7.2 will improve the performance of this system.
- Parallelization - The implementation of parallel solvers presents an interesting approach to CSPs in general. We intend to explore the usefulness of asynchronous methods (as described in [11] and [3]), as well as the potential for using decomposition strategies in conjunction with parallel programming.

- Local search strategies - Local search strategies, based on incomplete hill-climbing strategies, present a powerful method for solving scheduling problems. We intend to examine the quality of solutions and time necessary to find solutions using local strategies and comparing the results with systematic approaches.

- A weakness with our implementation is the storage of GTA and course data. This data is currently stored in ASCII format. The design of an interface for online data input has exposed the weakness and difficulty of this strategy; we hope to replace this approach with a database for storage to increase flexibility and reduce effort.

7.2 Ideas for reformulation

We hope to improve the performance of our solver at a high level by the use of some reformulation strategies. In particular, the approach to containment constraints and equality constraints may be modified in order to enhance performance.

Equality constraints are used to indicate that all courses in scope should be assigned the same GTA. Under the current formulation, it is possible that some courses in scope may be given an empty assignment. To alleviate this, we intend to aggregate variables connected by an equality constraint into a single variable. The union operation will most likely be used to establish constraints on these variables, as well as to establish the time intervals of the course offering. Refinement of this process can then occur, once we have evaluated its effectiveness in practice.

The second reformulation strategy is an alteration to the modeling of containment constraints. We do not wish to abandon this constraint altogether, but our experience has shown it to be unwieldy in practice. Further, we wish to retain it as an explicit constraint. We intend to evaluate the cost effectiveness of implementing these constraints as a network of `mutex` constraints, in terms of CPU time and constraint checks.

A third reformulation step involves the translation of some containment constraints into equality constraints. For sufficiently small containment sets, it is desirable that all courses in the contained set of the constraint be assigned the same GTA, in order to maximize course coverage. While an equality constraint does not explicitly prevent courses out of scope from being assigned the same GTA as is assigned to variables in scope, the capacity constraint in conjunction with the equality constraint tend to prevent the "outside" assignments. The motivation for this change is a hope that this will reduce the amount of time to find a solution by eliminating some expensive containment constraints.

Acknowledgments

This work was partially supported by the Department of Computer Science & Engineering and the Constraint Systems Laboratory. The author is thankful to the following people for their help and support. Dr. Berthe Choueiry, who advised the development of this project; Marilyn Augustyn, our "client" for this project; Chris Hammack, who has been developing an online system interface for this project; and the members of the Constraint Systems Laboratory for their suggestions and support.

References

[1] Fahiem Bacchus and Peter van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *AAAI/IAAI*, pages 310–318, 1998.

[2] Christian Bessière, Pedro Meseguer, Eugene C. Freuder, and Javier Larrosa. On forward checking for non-binary constraint satisfaction. In *Principles and Practice of Constraint Programming (CP'99)*, pages 88–102, 1999.

[3] Marius C.Silaghi, Djameela Sam-Haroud, and Boi Faltings. Asynchronous search with aggregations. In *AAAI 2000*, pages 917–922, 2000.

[4] Mark S. Fox. *Constraint-directed Search: A Case Study of Job-Shop Scheduling*. Morgan Kaufmann Publishers, 1987.

[5] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Journal of Artificial Intelligence*, 58:21–70, 1992.

[6] Robert M. Haralick and Gordon L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.

[7] William Harvey and Matthew Ginsberg. Limited discrepancy search. In *IJCAI'95: Proceedings of the International Joint Conference on Artificial Intelligence*, Montreal, 1995.

[8] John Hopcroft and Richard Karp. An n 5 2 algorithm for maximum matchings in bipartite graphs, 1973.

[9] Patrick Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9 (3):268–299, 1993.

[10] Jean-Charles Régin. A filtering algorithm for constraints of difference in constraint satisfaction problems. In *AAAI94P*, pages 362–437, AAAI94L, 1994.

[11] Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, 2000.

Figure 2: GTA data collection sheet

SURVEY ON SPRING 2001 ASSISTANTSHIP

(Complete both sides and return to Marilyn Augustyn, by **Monday, January 29, 2001**)
INCOMPLETE FORMS WILL BE SUBJECT TO PENALTY!

Name ______________________________ Advisor: ______________________

Degree program: ☐ M.S. (thesis) ☐ M.S. (project) ☐ Ph.D.

Semester admitted _________ Expected graduation date _________ # Years supported by CSE _____

Undergraduate GPA (if available): _______ Graduate GPA: _______

I currently ☐ have ☐ do not have a teaching assistantship in the CSE Dept.

My current assistantship is in the amount of $_________ (per semester)

Last 2 teaching assignments:

Course	Semester	Instructor

Deficiencies still to be taken:

GRE:

☐ General

Verbal _______ ____ %
Quantitative _______ ____ %
Analytical _______ ____ %

☐ Subject

Area ____________________
Score ________ _____%

Number of talks attended in the past year: Colloquia ☐ Ph.D. oral ☐ M.S. oral ☐

Foreign students only

SPEAK test: Date______________ Score ______________

ITA (Institute for International Teaching Assistants): Date passed _________
Date failed _________ Did not attend _______

COMMENTS:

(over)

Figure 3: GTA preference collection sheet

NAME________________________ **SPRING 2001 - TA Preferences**

Course	Course#	Sec.	Time	Day	Courses you are taking in Spring 2001	Please indicate your preferences ranking all courses between 0 and 5 (5 indicating highest preference)
Computer Science Fundamentals	101	001	0200-0315P	T R		
Computer Science Fundamentals Lab	101L	001	1030-1220P	R		
Computer Science Fundamentals Lab	101L	002	0130-0320P	W		
Computer Science Fundamentals Lab	101L	003	0600-0750P	T		
Intro to Computer Programming	150	150	0800-0915A	T R		
Lab	150	151	1230-0120P	M		
Lab	150	152	1230-0120P	T		
Lab	150	153	0830-0920A	M		
Intro Comp Sci I	155	150	0830-0920A	M W F		
Lab	155	151	0930-1020A	M		
Lab	155	152	0930-1020A	T		
Lab	155	153	1130-1220P	W		
Lab	155	154	1130-1220P	M		
Intro Comp Sci I	155	250	0130-0220P	M W F		
Lab	155	251	0830-0920A	W		
Lab	155	252	1030-1120A	M		
Lab	155	253	1230-0120P	R		
Intro Comp Sci I	155H	150	0130-0220P	M W F		
Lab	155H	151	0230-0320P	F		
Intro Comp Sci II	156	150	1030-1120A	M W F		
Lab	156	151	0830-0920A	T		
Lab	156	152	0930-1030A	W		
Lab	156	153	1130-1220P	F		
Lab	156	154	0230-0320P	M		
Intro Comp Sci II	156H	150	1030-1120A	M W F		
Lab	156H	151	0200-0250P	R		
Lab	156H	152	0130-0220P	M		
Computer Organization	230	001	0330-0420P	M W F		
Computer Organization Lab	230 L	001	0830-1020A	R		
Computer Organization Lab	230 L	002	0830-1020A	F		
Computer Organization Lab	230 L	003	1030-1220P	T		
Computer Organization	230H	001	0330-0420P	M W F		
Computer Organization Lab-Honors	230HL	004	0130-0320P	T		
Intro to Discrete Structures	235	150	1130-1220P	M W F		
Reci	235	151	0830-0920A	R		
Reci	235	152	0130-0220P	W		
Reci	235	153	0230-0320P	W		
C Programming 3/1-4/26/2001	251K	951	0830-0920A	T R		
UNIX Programming 1/9-2/27/2001	251U	951	0830-0920A	T R		
UNIX Programming 1/10-2/28/2001	251U	952	0130-0220P	W F		
UNIX Programming 1/9-2/27/2001	251U	953	0700-0850P	T		
X-Windows 3/2-4/27/2001	251Y	951	0130-0220P	W F		
X-Windows 3/6-4/24/2001	251Y	952	0700-0850P	T		
Cobol Programming	252A	001	0430-0520P	W		
Fortran Programming	252D	001	0430-0520P	M		
Data Struc. & Algorithms	310	001	0230-0320P	M W F		
Numerical Analysis I	340/840	001	0200-0315P	T R		
Human-Computer Interaction	378	001	1230-0120P	M W F		
Design and Analysis of Algorithms	423/823	001	1100-1215P	T R		
Compiler Construction	425/825	001	0130-0220P	M W F		
Computer Architecture	430/830	001	1030-1120A	M W F		
VLSI Design	434/834	001	0230-0320P	M W F		
Operating Systems Principles	451/851	001	1230-0145P	T R		
Distributed Operating Systems	455/855	001	1230-0120P	M W F		
Software Engineering	461/861	001	0930-1045A	T R		
Comm. Networks	462/862	001	0330-0445P	M W		
Software Des Methodologies	466/866	002	1100-1215P	T R		
Computer Graphics	470/870	001	0200-0315P	T R		
Intro to Artificial Intelligence	476/876	001	1130-1220P	M W F		
Computer Engineering Prof Dev	488	001	0930-1120A	W		
Computer Eng Senior Design Course	489	001	0930-1020A	M W F		
Spec Topics-Clustered Computing	496/896	004	0330-0420P	M W F		
Masters Project-Software Design	897	002	0200-0315P	T R		
Adv Topics in Database Systems	913	001	1230-0145P	T R		
Graph Algorithms	924	001	0200-0315P	T R		
Adv Computer Architecture	930	001	0930-1030A	M W F		
Math Theory of Finite Automata	935	001	0230-0320P	M W F		
Pattern Recognition	970	001	1130-1220P	M W F		
Genetic Algorithms	974	001	0130-0220P	M W F		
Seminar-Optical Comm Networks	990	001	0930-1045A	T R		
Seminar-Network Systems	990	003	1100-1150A	T		

1/23/2001