

GTAAP: AN ONLINE SYSTEM FOR MANAGING AND ASSIGNING
GRADUATE TEACHING ASSISTANTS TO ACADEMIC TASKS

by

Lim Way Hoong Ryan

A PROJECT

Presented to the Faculty of
The Graduate College at the University of Nebraska
In Partial Fulfillment of Requirements
For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Berthe Y. Choueiry

Lincoln, Nebraska

August, 2006

GTAAP: AN ONLINE SYSTEM FOR MANAGING AND ASSIGNING
GRADUATE TEACHING ASSISTANTS TO ACADEMIC TASKS

Lim Way Hoong Ryan, M.S.

University of Nebraska, 2006

Advisor: Berthe Y. Choueiry

We have developed GTAAP (Graduate Teaching Assistants Assignment Project), a fully operational prototype system for the management of GTAs in the Department of Computer Science & Engineering at the University of Nebraska-Lincoln. This online system assists in the assignment of Graduate Teaching Assistants (GTAs), based on their qualifications, availability, and preferences, to academic tasks such as grading, supervising laboratory sessions, and conducting recitations and lectures for introductory courses. Our system includes the following interacting components: (1) a MySQL database for storing courses and students data; (2) a web-based, secure interface for the candidate GTAs and another one for the department manager that are accessible even beyond the university network; (3) a web-based, secure interface for collecting the instructors' evaluations of the performance of their assigned GTAs; and (4) a new Java-based, interactive solver that is an improvement on the existing web-based interactive solver built by Thota [2004]. Additionally, our system smoothly integrates the components developed by other members of the Constraint Systems Laboratory for completing their graduate work [Zou, 2003; Guddeti, 2004b; Thota, 2004].

We have demonstrated the working prototype at the International Conference on Automated Planning and Scheduling (ICAPS 2004) and the International Conference on Principle and Practice of Constraint Programming (CP 2004) [Lim *et al.*, 2004a].

We have also discussed our work at the Conference on Prestigious Applications Intelligent Systems (PAIS/ECAI) [Lim *et al.*, 2004b] and the 2005 Annual Meeting of the Institute for Operations Research and the Management Sciences (INFORMS).

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Berthe Y. Choueiry who has made enormous contributions in making this project a reality. I am grateful for her constant support, insights, and guidance.

I also thank my colleagues (present and former) at the Constraint Systems Laboratory, especially Chris Reeson and Venkateshwar Rao Thota for their direct contributions to this project. Special thanks also to the CSE system-administrator, Mr. Charles Daniel who has graciously provided us with technical support.

Special thanks also to all my other colleagues at the Center on Children, Families, and the Law (especially Allison Jones and Charlie Lewis) who have all been very supportive and have provided continuous encouragement.

I am also grateful to my best friends Seng Yee Wong, Wei Jet Hew, and Chi Min Seow all of whom have not only have provided moral support, but also have at many times, taken my mind off work and have been great people to hang out with. To my good friends, Hooi Ling Lee, Hui Nee Chin, and Alvin Woon, thanks for being very supportive!

Lastly, but certainly the greatest, my parents, Mr. Lim Yen Chung and Mrs. Irene Lim, both whom have always supported me financially and morally. I am greatly indebted to them for everything.

This work was supported by the Department of Computer Science and Engineering of the University of Nebraska-Lincoln and CAREER Award #0133568 from the National Science Foundation.

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Constraint model	3
1.3	Previous work	4
1.4	Contributions	5
1.5	Organization of the report	6
2	System Implementation	8
2.1	System architecture	8
2.2	Database design	11
2.3	Password protected web-access	12
2.3.1	Web-access for the GTAs	13
2.3.2	Web-access for the manager	16
2.4	Instructors evaluations of GTAs	23
2.5	Interactive solver	23
2.5.1	Methodology	24
2.5.2	Features and benefits	25
2.5.3	Interface of the interactive solver	26
3	Conclusions and future work	30
A	Process description	33
B	Database Documentation	36
B.1	Authentication and logging	36
B.1.1	Detail description	36
B.2	GTA	39
B.2.1	Detail description	40
B.3	Manager	53
B.3.1	Detail description	54
B.4	Auxiliary tables	66
B.4.1	Detail description	66

C	Documentation for the Web-interface and SQL Queries	70
C.1	Web-interface file structure and hierarchy	70
C.2	MySQL queries	83
C.2.1	Queries posted from the GTA mode	83
C.2.2	Queries posted from the Classes mode	94
D	Interactive Solver: Java API Documentation	96
D.1	Package edu.unl.consystlab.gui	99
D.2	Interfaces	102
D.2.1	INTERFACE CanEnable	102
D.3	Classes	103
D.3.1	CLASS AboutBox	103
D.3.2	CLASS ComboItem	104
D.3.3	CLASS ComboListener	106
D.3.4	CLASS DisplayPreferences	107
D.3.5	CLASS GTAComboBox	108
D.3.6	CLASS GTAComboBox.horizontalScrollComboUI	111
D.3.7	CLASS GTAComboBoxEditor	111
D.3.8	CLASS GTAComboBoxRenderer	113
D.3.9	CLASS Interactive	114
D.3.10	CLASS JCheckBoxRenderer	127
D.3.11	CLASS JComboBoxEditor	129
D.3.12	CLASS JComboBoxRenderer	130
D.3.13	CLASS OpenDialog	131
D.3.14	CLASS PortForwardingL	133
D.3.15	CLASS PortForwardingL.MyUserInfo	134
D.3.16	CLASS PrintPage	135
D.3.17	CLASS RAButton	137
D.3.18	CLASS ResourceAssignmentCellEditor	139
D.3.19	CLASS ResourceAssignmentCheckBox	142
D.3.20	CLASS ResourceAssignmentCourseCellEditor	145
D.3.21	CLASS ResourceAssignmentDialog	147
D.3.22	CLASS ResourceAssignments	151
D.3.23	CLASS SaveDialog	154
D.3.24	CLASS SemesterYearSelector	157
D.3.25	CLASS StringRenderer	161
D.3.26	CLASS TunnelAuthenticationDialog	162
D.4	Package edu.unl.consystlab.gtaap.constraints	164
D.5	Classes	165
D.5.1	CLASS CapacityConstraint	165
D.5.2	CLASS CertificationConstraint	168
D.5.3	CLASS EqualityConstraint	170
D.5.4	CLASS MutexConstraint	173

D.5.5	CLASS NilPrefConstraint	176
D.5.6	CLASS OverlapConstraint	178
D.5.7	CLASS TakingCourseConstraint	180
D.6	Package edu.unl.consyslab.gtaap	183
D.7	Classes	184
D.7.1	CLASS Course	184
D.7.2	CLASS GTA	187
D.7.3	CLASS Preference	190
D.8	Package edu.unl.consyslab.gtajava	192
D.9	Classes	193
D.9.1	CLASS GTAAPConfig	193
D.9.2	CLASS Main	197
D.9.3	CLASS RandomGTAAPGenerator	198
D.9.4	CLASS YearSemester	200
D.10	Package edu.unl.consyslab	203
D.11	Classes	205
D.11.1	CLASS CSPConstraint	205
D.11.2	CLASS CSPProblem	206
D.11.3	CLASS CSPUtils	210
D.11.4	CLASS CSPValue	211
D.11.5	CLASS CSPVariable	214
D.11.6	CLASS CSPVVP	218
D.11.7	CLASS EVector	219
D.11.8	CLASS FC_Solver	220
D.11.9	CLASS jAssignment	226
D.11.10	CLASS LabelParams	231
D.11.11	CLASS Log	233
D.11.12	CLASS Setup	234
D.11.13	CLASS Utils	236

List of Figures

1.1	Constraint network of a simple problem instance.	4
2.1	System architecture of GTAAP.	9
2.2	GTAAP access for GTAs.	14
2.3	The page for updating one's academic record.	14
2.4	The page for input teaching preferences.	15
2.5	Results of the optional online survey of GTAAP by applicants.	17
2.6	The modes available to the manager.	17
2.7	The GTA mode in the manager's web-interface.	18
2.8	Manager's view of the GTAs who applied.	19
2.9	Selective queries used to filter GTAs based on a number of criteria.	19
2.10	GTAAP access for the department manager.	20
2.11	List of available and unavailable GTAs in the web-based interactive-selections.	22
2.12	Summarized results of the instructor evaluations of their assigned GTAs.	24
2.13	Interactive solver main screen. Two perspectives are visible in the same window pane.	27
2.14	Interactive solver doing a task assignment.	28
2.15	Interactive solver doing a resource assignment.	28

List of Tables

B.1	Table <code>auth_log</code>	37
B.2	Table <code>users</code>	38
B.3	Table <code>consider_application</code>	40
B.4	Table <code>exams_comps</code>	41
B.5	Table <code>exams_qual</code>	42
B.6	Table <code>gta_apply</code>	43
B.7	Table <code>gta_assignments</code>	43
B.8	Table <code>gta_attendance</code>	44
B.9	Table <code>gta_data</code>	45
B.10	Table <code>gta_deficiencies</code>	47
B.11	Table <code>gta_gre</code>	48
B.12	Table <code>gta_prefs</code>	49
B.13	Table <code>gta_prefs_ts</code>	50
B.14	Table <code>gta_speak</code>	51
B.15	Table <code>gta_survey</code>	52
B.16	Table <code>classes</code>	55
B.17	Table <code>class_confinements</code>	58
B.18	Table <code>consider_application_semesters</code>	59
B.19	Table <code>facEvals</code>	60
B.20	Table <code>hired</code>	62
B.21	Table <code>hired_info</code>	63
B.22	Table <code>jassignments</code>	64
B.23	Table <code>preassignment</code>	65
B.24	Table <code>same_ta</code>	65
B.25	Table <code>deficiencies</code>	67
B.26	Table <code>duty</code>	67
B.27	Table <code>faculty</code>	67
B.28	Table <code>grad_program</code>	68
B.29	Table <code>ita</code>	69
B.30	Table <code>semesters</code>	69

Chapter 1

Introduction

This report describes a system we have designed and implemented for the management of Graduate Teaching Assistants (GTAs) in the Department of Computer Science & Engineering (CSE) at the University of Nebraska-Lincoln. The task at hand is to assign GTAs based on their qualifications, availability, and preferences, to academic tasks such as grading, supervising laboratory sessions, and conducting recitations and lectures for introductory courses. This project originated from an idea listed on the web-page of Professor Rina Dechter of the University of California, Irvine. The project was started in Fall 2001 by Robert Glaubius as an undergraduate honors thesis [2001]. This application is a critical and demanding responsibility that the department's administration has to drudge through each semester. We set ourselves the challenge of building and testing a system that allows the students to make and revise their applications online and that effectively supports the department manager in making consistent decisions.

In this chapter, we state our motivation for this project, briefly outline the constraint-model, discuss our previous work, summarize our contributions, and state the organization of this report.

1.1 Motivation

The task of assigning GTAs to tasks is particularly difficult for humans because of all of the hard constraints about students' availability, qualification, and load that one has to keep track of. In our department, it is acknowledged to be the “most difficult duty of the department chair” [Reichenbach, 2001].

In a typical semester, CSE hires between 25 to 40 GTAs to be assigned approximately 70 different academic tasks. These tasks include teaching in lecturer courses and recitations, lab supervision, and grading. The problem is often tight (i.e., has few solutions in a large search space) and sometimes over-constrained (i.e., has no solution that satisfies all constraints). Historically, this task was performed by hand first by the department chair, then by the secretary of the Graduate Program, and currently it is carried out by a faculty member. Preliminary schedules were iteratively refined and updated based on the feedback from supervising faculty members and the GTAs themselves in a tedious and error-prone process that lasted over the first 3 weeks of the semester. These preliminary assignments often contained a number of conflicts and inconsistencies that negatively affected the quality of the CSE program. For example, when a course is assigned a GTA with little knowledge of the subject matter, the GTA has to invest significant effort to adjust to the situation. When the effort invested by the GTA is not sufficient to provide the adequate support, the course instructor has to perform a large portion of the GTA's duties. Moreover, students enrolled in the course may receive unfair grading or feedback.

Before this project was started, the entire process was carried out manually and on paper. It was obvious that any level of automation to maintain up-to-date data from the applicant students and any level of computer-support to support sound and consistent decision making by the department were badly needed.

1.2 Constraint model

It is not the goal of this report to discuss the model and computational mechanisms built in this project, which are the contributions of other members of the Constraint Systems Laboratory [Glaubius, 2001; Zou, 2003; Guddeti, 2004b; Thota, 2004]. Rather, we focus on the system’s architecture and implementation, which are our own contribution. Nevertheless, we summarize below the adopted model.

Informally speaking, the problem is to assign GTAs to academic tasks under a number of constraints while covering as many courses as possible (as a primary optimization criterion) and maximizing the GTAs’ preferences (as a secondary optimization criterion). As the first task undertaken by Glaubius in his undergraduate honors thesis, Glaubius modeled this combinatorial problem as a Constraint Satisfaction Problem (CSP) [2001]. A CSP is defined as a set of variables with associated domains and a set of constraints that restrict the combination of values that the variables can simultaneously take from their respective domains, the task being to find an assignment of values to variables that satisfies all the constraints. Glaubius modeled the academic tasks¹ as variables in a CSP and the GTAs as the values that the variables may take. The constraints restricting the allowable assignments of GTAs to courses are expressed as either unary, binary, and non-binary relations. Figure 1.1 shows the constraint network associated with a CSP with three variables and three constraints.

In GTAAP, the unary constraints include whether or not the student has the appropriate teaching certifications (i.e., ITA certified); whether or not the student is enrolled in the course in question or (for laboratory sessions, recitations, or lectures tasks) enrolled in another course at the same time; and whether or not the student

¹We sometimes use the generic terms ‘course’ to refer to any academic task carried out by a GTA, such as grading, supervising a laboratory session, and teaching a lecture course or recitations.

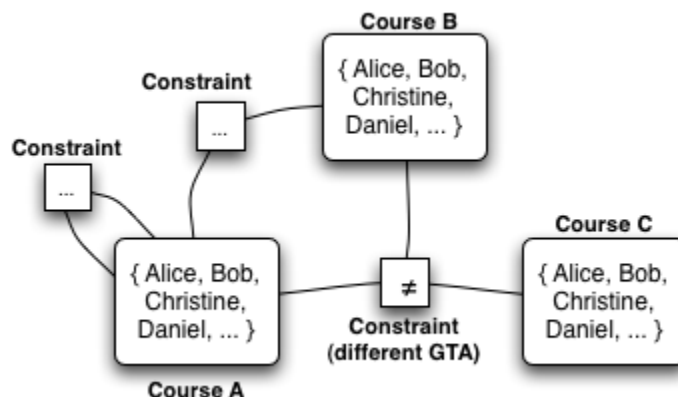


Figure 1.1: Constraint network of a simple problem instance.

has expressed his/her inability to teach the course (represented by a zero-preference constraint). Binary constraints include constraints stating that a student can be assigned to at most one of two courses (i.e., mutex constraint) or to exactly any two courses. Finally, non-binary constraints include constraints stating that the load assigned to any GTA cannot exceed the capacity at which he/she has been hired (i.e., capacity constraint), and constraints restricting the assignment of a GTA to a specific set of courses.

1.3 Previous work

In addition to modeling the problem as a CSP, the Constraint Systems Laboratory has developed a portfolio of automated algorithms for solving this problem. These algorithms include a Environment-Reactive-Agents search (ERA) [Zou, 2003; Zou and Choueiry, 2003a; 2003b], a stochastic local search [Zou and Choueiry, 2003b], backtrack search with various ordering heuristics [Glaubius, 2001; Guddeti, 2004b], and a randomized backtrack search [Guddeti, 2004b; 2004a; Guddeti and Choueiry, 2004; 2005]. All of those algorithms focus on automatically finding consistent solutions

(i.e., all constraints are satisfied) while trying to maximize the quality of the solution in terms of the two criteria listed above. That previous work was able to identify regimes (e.g., over-constrained or under-constrained problems) where a technique outperforms the others. However, all those search techniques are automated and do not allow the user (i.e., manager) to intervene in the decision making once the search has started.

While the above approaches were being pursued, our group remained particularly aware of the limitations of automated search. Indeed, in reality, the manager has plenty of information about the GTAs, the courses, and the teaching faculty at his/her fingertips. It is illusive to attempt to formally model such constraints because they are typically subjective and infinite. Also, they would complicate the problem solving without necessarily improving the quality of the solutions. *Fortunately in our particular setting, computers seem to be most effective in exactly the same tasks where humans fail* (e.g., keeping track of hard constraints), *and vice versa* (e.g., quickly evaluating alternatives and making compromises). In response to these observations, Thota [2004] developed an ‘Interactive Search’ that allows the human user (i.e., the manager) to make decisions interactively while being prompted, at each step, with the set of choices consistent with the manager’s past decisions. The interactive search keeps the human user in the decision loop while removing the need for tedious (and error-prone) manual assignments.

1.4 Contributions

I started working on the project as an undergraduate student in Spring 2003. Our contributions, which we describe in the rest of this report, are as follows:

1. We have designed a MySQL database to store the data of students applications

and class information.

2. We have built web-access interfaces for data-entry by candidate GTAs and the department manager.
3. We have co-developed a facility for collecting evaluations by the supervising instructors of the performance of their assigned GTAs. The set of supervising instructors includes student instructors, teaching staff, instructors from outside CSE, and CSE faculty members.
4. We have rebuilt an interactive solver in Java based on Thota's [2004] web interactive-solver.

The resulting system is a significant step towards streamlining the GTA hiring and management process. The system has already significantly contributed in making decisions accountable and 'traceable,' while archiving the data collected and the decisions made over the semesters. It is only fair to say that the system has contributed to making the management of GTAs in the department far smoother and more effective than it was before the project had started.

1.5 Organization of the report

The rest of this report is organized as follows. Chapter 2 discusses our work on GTAAP. We draw our conclusions and propose future work in Chapter 3. The appendices provide a relatively detailed description of the implementation. The information included in the appendices is useful for using and maintaining the system and for carrying out further developments. Appendix A outlines the tasks for setting up and using the system before the beginning of any given semester. Appendix B is concerned with the database (tables and attributes). Appendix C describes the

content of the files associated with the web-interfaces. Finally, Appendix D has the documentation for the Java API (JavaDoc).

Chapter 2

System Implementation

In this chapter, we present the architecture and components of the system we have implemented. We first discuss the design of the database. Then, we describe the web-interfaces (for the potential GTAs, for department manager, for GTA evaluation by instructors). Finally, we discuss our new interactive solver.

2.1 System architecture

Figure 2.1 shows how these components interact to form the system. In this figure, we mark with a ‘*’ the components of GTAAP that we entirely developed in the course of this project, and with a ‘+’ those to which we have substantially contributed.

The components of the system are currently as follows:

1. A password-protected web-access for applicant students to enter and update their record and another one for the manager to input and manage data. The system is accessible, both for the GTAs and department manager, via the web at <http://cse.unl.edu/~gta/>. Potential GTAs must register and enter their academic data and teaching preferences at the web interface. After a given

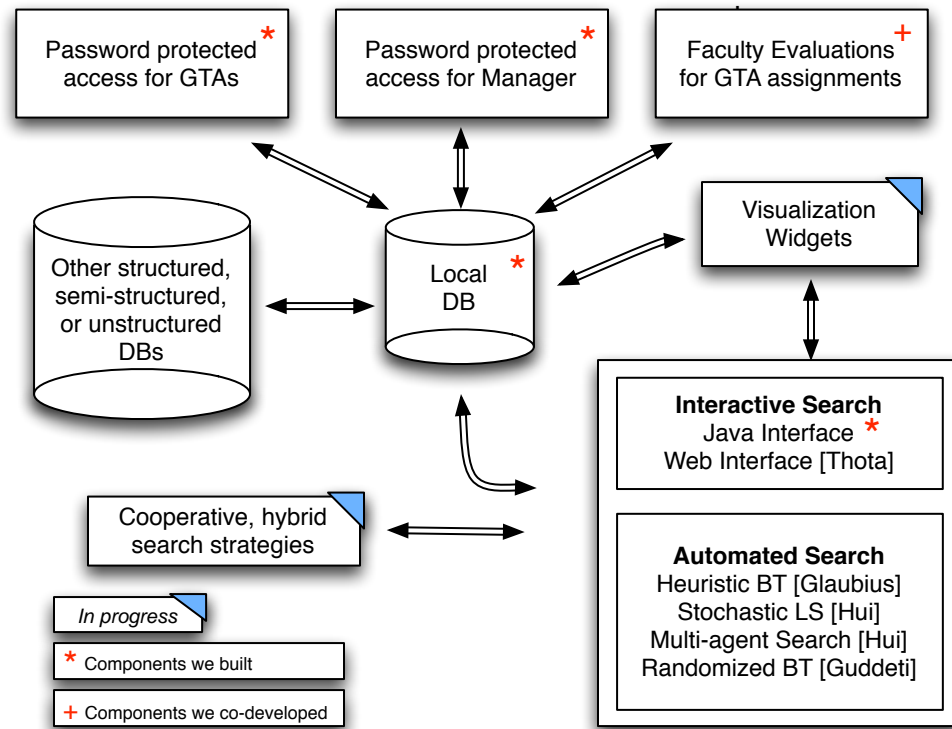


Figure 2.1: System architecture of GTAAP.

deadline, the manager accesses the web interface to hire GTAs. He/She can view data about the candidate GTAs, set up courses, make pre-assignments, perform search and view historical data.

2. A password-protected web-access for instructors to evaluate their assigned GTAs. This part of the system is accessible via the web at

http://cse.unl.edu/~gta/fac_evals/

and uses the CSE authentication mechanism and was co-developed Ms. Traci Fink.

3. A central data store (MySQL database). This database stores information about the candidate GTAs and courses.

4. Problem-solving algorithms, both interactive and automated methods. This category include a ‘Java Interactive Solver’ that we implemented as an improvement of the the solver developed by Thota [2004].
5. An interface to UNL’s central database via the department’s mirror database. In order to retrieve course data for the next semester, we import the course data from other structured, semi-structured, or unstructured databases provided by the university into our MySQL database. These data are obtained from the university class schedule database and the SIS+ database.
6. The following components are still under early stages of development: Cooperative, hybrid search strategies that allow the department manager to compose partial solutions built using the interactive or automated solvers in order to build complete solutions. Also remaining to be addressed are visualization widgets that provide a graphical representation of the solution space and enable the manager to monitor the search process while receiving pertinent statistics about a problem as solutions are being built.

The various components are implemented in different programming languages. In the late 2001, Chris Hammack wrote the web-interface for GTAs in JSP (Java Server Pages). We rewrote all the web-access interfaces in PHP4 (www.php.net). PHP is the language of choice for the interface because it is the dominant web programming-language and contains a vast variety of built-in functions. Further, it has a built-in MySQL driver for access to the MySQL database. The automated search, its data structures and supporting propagation algorithms are written in Common LISP. Thota [2004] wrote the initial Interactive Search as a web-based application running on the departmental web-server. Thota’s propagation algorithms are written in Common LISP and his web-interface in PHP4. These two components communicate via

TCP sockets.

Once all the required data (course data, course setups, GTA applications and teaching preferences) are in the database, the manager is ready to make assignments either using automated search or interactive search. The various tasks for setting up and using the system for a given semester are described in details in Appendix A.

In the rest of the chapter, we discuss each of our contributions in details. These include the database, the password protected web-access for the GTAs and the manager, the instructor evaluation of the GTAs, and the interactive solver.

2.2 Database design

When this system was started in Fall 2001, the data was collected by the department on paper forms, and the data was manually entered and stored in plain-text files. We stored the information of each GTA on separate files. With the numerous files and manual data-entry, there was bound to be inconsistencies in the files. Flat-files do not satisfy the database ACID properties (atomicity, concurrency, isolation, and durability [Garcia-Molina *et al.*, 2002]) that proper databases have. The fields in the flat-files are position dependent, and no attributes may be added between the attributes without making significant changes to the file reading methods. In addition to these limitations, flat-file storage does not scale up well.

In Spring 2003, I decided to transition away from using flat-files to a MySQL database. We chose MySQL because it was a popular and open-source database with APIs available to most programming languages. (If the decision was to be revised today, we recommend using a more feature-rich database system such as Oracle.) Along with the move to MySQL, we also imported data such as course schedules from external databases with the gracious help of the department's system administrator,

Mr. Charles Daniel. We group our MySQL tables into 4 categories:

1. *Authentication and Logging*: These tables are related to user information, authentication, and access logging. The tables are: `auth_log`, `users`.
2. *GTA*: These tables are related to GTA academic data and teaching preferences. The tables are: `exams_comps`, `exams_qualis`, `gta_apply`, `gta_assignments`, `gta_attendance`, `gta_data`, `gta_deficiencies`, `gta_gre`, `gta_prefs`, `gta_prefs_ts`, `gta_speak`, `gta_survey`.
3. *Manager*: These tables are related to courses, the constraints setup, hiring information, and GTA evaluations. The tables are: `classes`, `class_confinements`, `consider_application_semester`, `facEvals`, `hired`, `hired_info`, `preassignment`, `same_ta`.
4. *Auxiliary tables*: These tables are read-only to provide frequently accessed information. The tables are: `deficiencies`, `duty`, `faculty`, `grad_program`, `ita`, `semester`.

In Appendix B, we discuss the above-mentioned tables in details listing their attributes, types, and default values.

2.3 Password protected web-access

The next logical step after transitioning to a MySQL database was to provide web-access for data collection. We developed the web-access for GTAs and the department manager using PHP4. With this web-access, GTAs and the manager input data directly into our MySQL database, eliminating inconsistency resulting from the manual data-entry. We first developed the GTA web-access for potential GTAs to apply for a position, updating their academic record and specifying their teaching. Next, we

developed the manager web-access for the department manager to setup classes for GTA assignments and view assignment data. Details about the web-interface file locations and the SQL queries in these modes are in Appendix C. Below, we discuss in detail our interfaces for the GTAs and the department manager.

2.3.1 Web-access for the GTAs

The GTA web-access provides access to the potential GTAs to manage their academic data and input their teaching preferences. Accounts are password protected. Graduate students can access this page at any time. However, a deadline for receiving applications can optionally be enforced. It is crucial that the system be not restricted to applicants who already have accounts in the department. Indeed, admitted graduate students from foreign countries need to be able to register on the system and input their academic data and preferences over the Internet prior to arrival. Figure 2.2 shows a screen-capture of the GTA's main web-page. The interface offers nine modes listed in a line on the top of the screen shot: 'Print' the displayed page, access the 'Main Page,' the student's 'Academic Record,' 'Teaching Preference,' 'Login Info,' 'Course Descriptions,' a 'Survey,' 'Email webmaster,' and 'Logout.' The first two modes (i.e., academic record and teaching preferences) display the time-stamp of the last modifications made by the applicant. Importantly, every time an applicant modifies his/her record or preference selections, a message confirming the new data input is sent via email to the applicant for his/her personal record. The 'main page' the student instructs the student to complete the three most important step: updating the academic record, inputting teaching preferences, and, optionally, filling out a survey. It also allows the student to check a semester to be considered for.

GTAs are hired based on their qualifications. It is critical that GTAs maintain an up-to-date academic data information in the system. Figure 2.3 shows the page for

GTAAP

[Print page](#)
[Main page](#)
[Academic record](#)
[Teaching preference](#)
[Login info](#)
[Course descriptions](#)
[Survey](#)
[Email webmaster](#)

[Logout](#)

Welcome test test,

We need some information from you to continue. Please follow these steps.

- [Fill in your Academic Record](#)
- [Tell us your Teaching Preference](#) --Note the teaching preferences for Fall 2006 are not yet available, we will update you when they become available.
- [Fill out a survey \(optional\)](#)

I want to be considered for the following semesters:

- Fall 2006 (08/14/06-12/31/06)
- Spring 2006
- Fall 2005 (08/15/04 - 01/02/05)
- Summer (2nd 5 weeks) 2005
- Summer (1st 5 weeks) 2005
- Summer (8 weeks) 2005
- Summer (Pre-session) 2005
- Spring 2005 (01/03/05 - 05/13/05)
- Fall 2004 (08/16/04 - 01/02/05)

Figure 2.2: GTAAP access for GTAs.

updating one's academic record. The information on this page includes the number

Information

Name	test test	Data last saved	2006-06-20 14:49:17
------	-----------	-----------------	---------------------

General

Program	M.S. (Thesis)	Advisor	- OTHER -
Semester admitted	FALL Year: 2005	Semesters supported	GTA 1 GRA 1
Expected graduation semester	SPRING Year: 2008	I am currently a	GTA 0 GRA 1/2
Deficiencies	CSCE 451 - NONE - - NONE - - NONE - - NONE -	GRE	Score Percent
		Verbal	790 90.00
		Quantitative	800 99.00
		Analytical	800 99.00
		Subject: CSCE	795 96.00
Attendance	Colloquia 3 M.S. Oral 1 Ph.D. Oral 1	Ph.D. qualifiers	Theory Not applicable Systems Not applicable Applications Not applicable Area:
Undergraduate GPA	2.89 (if none, type "none")		

Figure 2.3: The page for updating one's academic record.

of semesters supported so far, the level of support, the current advisor, previous teaching experience, GPA, English proficiency level, ITA qualification¹, the list of course deficiencies, etc.

Figure 2.4 is a screen shot of the page presented to a student to update his/her teaching preference for a given semester. This page the list of courses offered during

Preference for test test

Fall 2006

[Undo changes](#) [Submit](#)

I am applying to be a TA for this semester.

Class	Section	Description	Time	Days	Enrolled?	Preference	Justification for "0"
CSCE 101	001	BASICS OF COMPUTING	1230 - 1320	M W F	<input type="checkbox"/>	3) Qualified	
CSCE 101	001G	BASICS OF COMPUTING	1230 - 1320	M W F	<input type="checkbox"/>	0) Cannot handle	
CSCE 101L	001	FUND COMPUTING LAB	1830 - 2020	R	<input type="checkbox"/>	3) Qualified	
CSCE 101L	004	FUND COMPUTING LAB	1130 - 1320	R	<input type="checkbox"/>	3) Qualified	
CSCE 105	010	PRBLM SOLVNG:COMPUTR	1430 - 1520	M W F	<input type="checkbox"/>	3) Qualified	
CSCE 105	010g	PRBLM SOLVNG:COMPUTR	1430 - 1520	M W F	<input type="checkbox"/>	0) Cannot handle	

5) Best choice You would love to teach this class	4) Favorite You are very capable of teaching this class	3) Qualified You don't mind this assignment.	2) Able to handle You will survive ...	1) Avoid if possible Your last choice before giving up your assistantship.	0) Cannot handle You are either enrolled in the class or not able to teach it. Either way you must produce a legitimate reason.
---	---	--	--	--	---

Figure 2.4: The page for input teaching preferences.

the semester and their meeting times. For each course, an applicant can specify his/her preferences: '5 Best choice,' '4 Favorite,' '3 Qualified,' '2 Able to handle,' '1 Avoid if possible,' '0 Cannot handle.' The student can check a box indicating that he/she is enrolled in given course, which automatically sets up the preference value to 0. In all other cases where the applicant specifies a preference value of 0, the system prompts the applicant to provide a justification for the inability to handle the course. A preference value is set to 3 by default to reduce the applicants' burden in

¹International Teaching Assistant (ITA) is a university monitored training that foreign students are required to complete to qualify for classroom instruction.

specifying preferences. At the bottom of the page, a permanent legend reminds the students of the meaning of the different preference values.

The third mode provides students with direct access to the official course descriptions at CSE, which is useful for incoming students who have not yet joined the department.

The ‘Survey’ button allows students to express their opinion about the site. The questions asked are about:

1. Navigation: Ease, flexibility, robustness.
2. Data entry: Ease, flexibility, robustness.
3. Other aspects: Whether the online form or paper form is preferable; The number of times the student revised his/her application; Whether any question was superfluous or missed; and the student had a constructive comment.

Figure 2.5 shows the satisfaction of the candidate GTAs using our system. We always address the feedback provided by the students through the survey as soon as we receive it in order to enhance the system.

2.3.2 Web-access for the manager

The interface for the manager permanently displays, in a horizontal frame on the top of the page, five buttons representing the *operational modes* (see Figure 2.6). These modes are: ‘GTAs,’ ‘Classes,’ ‘Interactive selections’ (for interactive problem solving), ‘Search’ (for automatic problem solving), ‘Server console’ (for system level commands), and ‘Logout.’ These modes are controlled by a semester selector (left-most in Figure 2.6), set up by default to the most recent semester in the database. When any of these 5 modes is selected, additional buttons for selecting *functionalities*

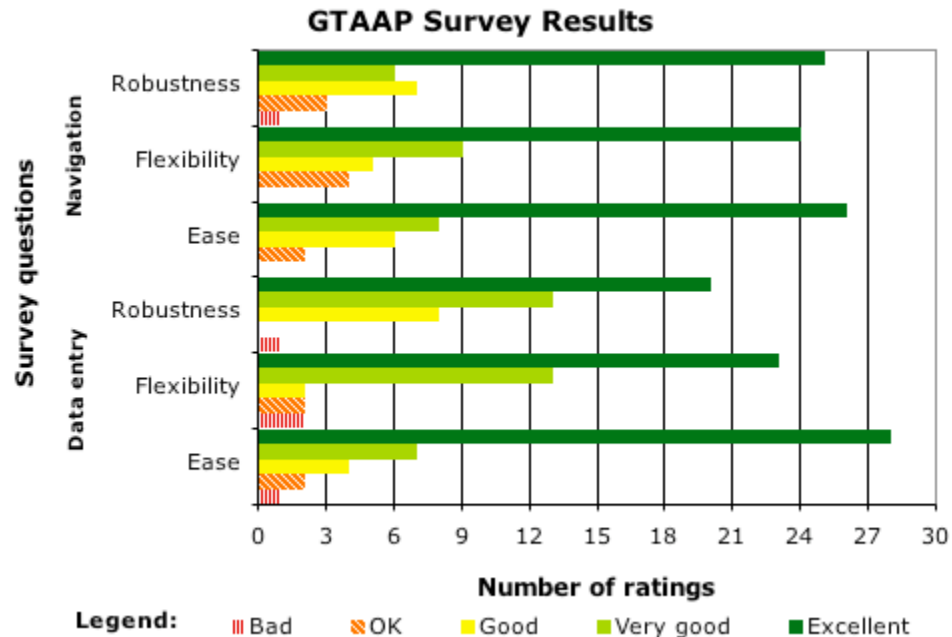


Figure 2.5: Results of the optional online survey of GTAAP by applicants.



Figure 2.6: The modes available to the manager.

specific to the mode appear in a vertical frame in the left margin. These buttons are displayed as long as the mode is active. The MySQL queries that implement the functionalities in both modes are given in Appendix C.2. Below, we discuss the following modes: GTAs, Classes, Interactive selections, Search, and Server console.

2.3.2.1 GTAs

The first mode allows the manager to access information about GTAs in the system. Selecting this mode displays an empty page with several functionalities listed in the left margin of the page (see Figure 2.7). These functionalities are clustered in two main groups: ‘GTAs in Selected Semester’ and ‘GTA All Semesters Information.’ The

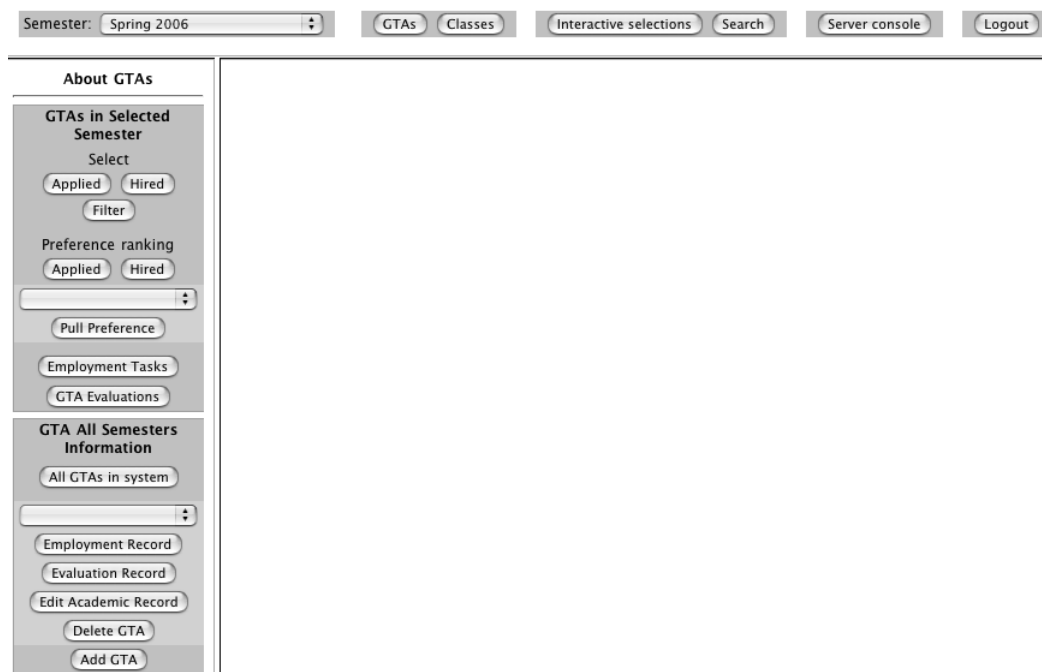



Figure 2.7: The GTA mode in the manager's web-interface.

functionalities included in the former group allow the manager to work on a particular semester and is discussed below. The ones included in the latter allow the manager to view, for a particular student, the employment, evaluation, and academic records for all semesters, and to add or remove a GTA in the database. The functionalities under 'GTAs in Selected Semester' are the most important for managing a given semester. They allow the manager to view the list of students who have applied for the semester ('Applied'), those that have been hired for the semester ('Hired'), those that fit a set of criteria ('Filter'), etc.

The 'Applied' button, which displays the page shown in Figure 2.8, gives the manager has the ability to hire, release, and specify the hiring percentage (i.e., GTA capacity) of a particular applicant, and to view and print reports about any number of the listed applicants. Further, the manager can send an email message to any subset of these applicants. A similar page, restricted to hired GTAs, is displayed by

GTAs in semester: Applied Customize View 

Spring 2006

Hire	Capacity	Name	Semesters supported as		Acad. record	Preference		Email
			GRA	GTA		Updated	Updated	
<input checked="" type="checkbox"/>	1/2	Abbey, Talia	2	2	<input type="checkbox"/>	06/03/27	<input type="checkbox"/>	05/11/22
<input checked="" type="checkbox"/>	1.0	Aguon, Ted	6	2	<input type="checkbox"/>	05/11/10	<input type="checkbox"/>	05/11/10
<input type="checkbox"/>	1.0	Bathke, Clarence	1	1	<input type="checkbox"/>	06/03/28	<input type="checkbox"/>	- ??? -
<input checked="" type="checkbox"/>	1.0	Behrenwald, Coleman	0	3	<input type="checkbox"/>	05/11/12	<input type="checkbox"/>	05/11/18
<input checked="" type="checkbox"/>	2/3	Belliston, Hallie	4	5	<input type="checkbox"/>	06/03/28	<input type="checkbox"/>	05/11/18
<input checked="" type="checkbox"/>	1/2	Bettters, Lisandra	2	2	<input type="checkbox"/>	06/04/13	<input type="checkbox"/>	05/11/18
<input type="checkbox"/>	1.0	Boshell, Candra	0	6	<input type="checkbox"/>	05/11/21	<input type="checkbox"/>	05/11/12
<input checked="" type="checkbox"/>	1.0	Castoreno, Matilde	4	4	<input type="checkbox"/>	05/11/10	<input type="checkbox"/>	05/11/10
<input checked="" type="checkbox"/>	1/3	Cohill, Clemencia	0	0	<input type="checkbox"/>	06/03/05	<input type="checkbox"/>	06/03/05
<input checked="" type="checkbox"/>	1/2	Critelli, Debra	0	3	<input type="checkbox"/>	05/11/16	<input type="checkbox"/>	05/11/16
<input checked="" type="checkbox"/>	1/2	Deel, Salina	0	0	<input type="checkbox"/>	06/04/05	<input type="checkbox"/>	00/00/00
<input type="checkbox"/>	1.0	Dilbeck, Bridgett	2	0	<input type="checkbox"/>	06/04/10	<input type="checkbox"/>	05/11/16
<input checked="" type="checkbox"/>	1.0	Dreher, Bari	0	1	<input type="checkbox"/>	05/11/16	<input type="checkbox"/>	05/11/16
<input checked="" type="checkbox"/>	2/3	Fazzino, Felix	0	0	<input type="checkbox"/>	06/02/23	<input type="checkbox"/>	06/03/03
<input type="checkbox"/>	1.0	Finnell, Gregg	5	3	<input type="checkbox"/>	05/11/23	<input type="checkbox"/>	05/11/23
<input type="checkbox"/>	1.0	Fragoso, Delia	0	0	<input type="checkbox"/>	05/11/22	<input type="checkbox"/>	05/11/22
<input type="checkbox"/>	1.0	Giesler, Corinne	2	1	<input type="checkbox"/>	06/04/15	<input type="checkbox"/>	05/11/30

Figure 2.8: Manager's view of the GTAs who applied.

the 'Hired' button.

The 'Filter' allows the manager to select a subset of the applicants based on their performance history and according to a set of pre-defined criteria with user-selectable values. The 'Filter' functionality is shown in Figure 2.9.

GTAs: Filter

Program ITA Past TA support

GPA Current support Past RA support

Applied for consideration for:

Figure 2.9: Selective queries used to filter GTAs based on a number of criteria.

Similar to the 'GTAs' mode, the remaining functionalities display pre-set queries from the database to support the user in decision making.

2.3.2.2 Classes

The second mode of the manager's web interface allows the manager to manipulate information about the classes in a semester, specify course loads, course types (i.e., grading, lab, recitation, or lecture), and set-up various dependencies among courses. Figure 2.10 shows a screen shot of a page of the second mode. The 'Preassignment'

Semester: Spring 2006 | GTAs | Classes | Interactive selections | Search | Server console | Logout

About Classes

Manage Classes

Setup | Preassignment

Constraints

Confinement | Same-TA

Reports

Preference ranking

Class: Load & Type Print Page

Spring 2006

Cancel	Class	Section	Description	Time	Days	Course load	Type	Duplicate	Status
<input type="checkbox"/>	CSCE 101		FUND OF COMPUTING	1400 - 1525	TR	0.0	Lecture	<input type="checkbox"/>	
<input type="checkbox"/>	CSCE 101	001	FUND OF COMPUTING	1400 - 1525	TR	1/2	Lecture	<input type="checkbox"/>	
<input type="checkbox"/>	CSCE 101L	001	FUND COMPUTING LAB	1030 - 1220	R	1.0	Lab	<input type="checkbox"/>	
<input type="checkbox"/>	CSCE 101L	002	FUND COMPUTING LAB	1330 - 1520	W	0.0	Lab	<input type="checkbox"/>	
<input type="checkbox"/>	CSCE 105		PRBLM SOLVNG:COMPUTR	1330 - 1420	MWF	1/4	Lecture	<input type="checkbox"/>	
<input type="checkbox"/>	CSCE 105	150	PRBLM SOLVNG:COMPUTR	1330 - 1420	MWF	2/3	Grading	<input type="checkbox"/>	
<input type="checkbox"/>	CSCE 105	151	Laboratory	0830 - 0920	W	1/4	Lab	<input type="checkbox"/>	
<input type="checkbox"/>	CSCE 105	152	Laboratory	1130 - 1220	T	1/4	Lab	<input type="checkbox"/>	
<input type="checkbox"/>	CSCE 105	153	Laboratory	1230 - 1320	T	1/4	Lab	<input type="checkbox"/>	
<input type="checkbox"/>	CSCE 105	154	Laboratory	1030 - 1120	W	0.0	Lab	<input type="checkbox"/>	

Canceled
 Preassigned
 New (Unsaved)
 New (Saved)

Figure 2.10: GTAAP access for the department manager.

functionality in this mode allows the manager to enforce the assignments of GTAs to classes to satisfy some requirement external to the system. For each class, a pull-down menu lists all hired GTAs (regardless of whether they are available, appropriate for the task, etc.) from which manager can select. Such pre-assignments are considered fixed, can be enforced only through this specific page, and cannot be undone by the interactive or automated search facilities.

The 'Confinement' functionality allows the manager to specify a subset of courses such a GTA assigned to course in the subset cannot be assigned to a course outside the subset. The 'Same-TA' functionality allows the manager to specify a subset of courses to be serviced by the same GTA.

Similar to the ‘GTAs’ mode, the remaining functionalities display pre-set queries from the database to support the user in decision making.

2.3.2.3 Interactive selections

The ‘Interactive selections’ is the web-based interactive solver developed by Thota [2004]. This mode is currently thought to be the most useful by our users because it assists the manager in interactively making individual or group assignments manually in an efficient way. The manager remains in total control of the decisions, but is relieved of the burden of keeping track of the consistency among decisions.

The manager can adopt one of two dual perspectives on the problem and switch between them: assigning GTAs for courses or vice versa. We use constraint propagation to determine the list of available and busy GTAs (alternatively, courses), and to propagate the effects of the manager decisions on the remaining open choices.

Figure 2.11 shows how the appropriate GTAs (i.e., those that have passed node consistency) are listed to the user. The upper portion of the pull-down displays the GTAs that can be assigned to the course without reservation. In terms of the CSP, these are the values in the current domain of the variable. The lower portion lists the GTAs who could potentially be assigned but are ‘busy’ in assignments from which they should be first relieved. These GTAs correspond to the values eliminated from the domain of the variable by constraint propagation. In each portion, the GTAs are listed in decreasing preference order, as a primary criterion, then in increasing lexicographical of their last name, as a secondary criterion. Next to the name, the ‘current’ capacity of each GTA is displayed (which is the hired capacity of the GTA discounted by the load of his/her other assignments). Every time a new assignment is made, an efficient consistency algorithm (a full arc-consistency) is executed to filter the domains of the unassigned variables. When assignments are undone or changed,

MWF	1	----- Nobody -----
MWF	0.66	----- Nobody -----
MWF	0.66	Available GTAs
MWF	1	3 -- Ahlborn Charissa (1)
MWF	0.5	3 -- Dreher Bari (1)
TR	0.33	3 -- Huitzacua Thaddeus (1)
TR	0.5	3 -- Kaczor Lorenza (1)
MWF	0.25	3 -- Rothenberger Torri (1)
MWF	0.75	3 -- Telega Elsie (1)
MWF	0.75	-----
MWF	0.75	Filtered out GTAs
MWF	0.75	4 -- Lipszyc Aleisha (0.75)
MWF	0.75	4 -- Makris Gilbert (0.25)
MWF	0.75	3 -- Aguon Ted (0)
MWF	0.75	3 -- Behrenwald Coleman (0.5)
MWF	0.75	3 -- Boshell Candra (0.67)
MWF	0.75	3 -- Castoreno Matilde (0.34)
MWF	0.75	3 -- Critelli Debra (0.34)
RTY		

Figure 2.11: List of available and unavailable GTAs in the web-based interactive-selections.

the domains of all the variables are first re-filtered from scratch while maintaining the intermediate selections.

Based on this web-based interactive solver, we have developed a new interactive solver as a Java application. We discuss our implementation in Section 2.5.

2.3.2.4 Search

The fourth mode of the manager’s web interface offers several search algorithms for automatically solving the problem. The algorithms currently available are an ‘Environment-Reactive-Agents’ search (ERA) [Zou, 2003; Zou and Choueiry, 2003a; 2003b], a stochastic local search [Zou and Choueiry, 2003b], backtrack search with various ordering heuristics [Glaubius, 2001; Guddeti, 2004b], and a randomized backtrack search [Guddeti, 2004b; Guddeti and Choueiry, 2005; 2004; Guddeti, 2004a]. These algorithms consider the pre-assignments made in the second mode of the manager’s web interface as hard constraints, but do not (yet) take into account the interactive

decisions made in the third mode. These algorithms run independently whenever selected by the user. Their integration among themselves and with the interactive solver should be addressed in the future.

2.3.2.5 Server console

This mode allows the manager to perform server-level commands including restarting the LISP-PHP communication socket and viewing the LISP-PHP daemon information.

2.4 Instructors evaluations of GTAs

At the completion of a semester, supervising instructors evaluate the performance of their assigned GTAs based on a number of criteria set by the department and provide their recommendations about re-hiring their assigned GTAs. These evaluations are important for the department in the selection process for the following semesters. The evaluator is asked questions pertaining to the GTA's knowledge of the course material, communication skills, quality of work, organization and planning skills, overall effectiveness, students' feedback, evaluator's feedback, and the evaluator's overall recommendation for the GTA. The evaluation page was initially done by Traci Fink, but we re-designed it in Spring 2005 to enhance flexibility.

We summarize the results of these evaluations and make available through the manager's web-page as shown in Figure 2.12.

2.5 Interactive solver

The task addressed in GTAAP is ideally suited to an interactive approach. Interactivity keeps the manager in control of the decision-making process. In our context,

Spring 2006																			
Name		Program	Advisor	GPA	ITA	History	Current employment			Evaluation									
last	first	Type	Started			TA	RA	FTE	Course	Role	A	B	C	D	E	Avg	Rec	Rater	
Abbey	Talia	MS	F 2005	Lue Broderson	3.11	X	2	2	0.50	340	Grader	5	5	5	4	4.80	RG	Giovanna Parkos	
Aguon	Ted	PhD	S 2003	Kym Karnish	3.83	√	2	6	1.00	155	Grader	6	5	4	4	5	4.80	SR	Sandee Fuhrmann
										155	Lab instructor	6	5	4	5	5	5.00	SR	Sandee Fuhrmann
										340	Grader	6	6	6	6	6.00	SR	Giovanna Parkos	
Behrenwald	Coleman	MS	F 2004	Carlita Ebesu	3.61	n/a	3	0	1.00	101	Course instructor							- NONE -	
Belliston	Hallie	PhD	F 2003	Cierra Brazel	3.60	√	5	4	0.66	150	Lab instructor	5	5	5	4	5	4.80	SR	Giovanna Parkos

Evaluation legend: Larger numbers are better.

A Knowledge of material (1-6)	B Communication skills (1-6)	C Quality of work (1-6)	D Organizational skills (1-6)
E Overall effectiveness (1-6)	Avg Average of A-E (1-6)	F Recommendation (SR, RD, RG, NO)	
SR strongly recommend	RD recommend	RG recommend only for grading	NO do not recommend

Figure 2.12: Summarized results of the instructor evaluations of their assigned GTAs.

an interactive solver must fulfill the following criteria:

- Keep the user in-control of any decisions.
- Maintain at all times a consistent state even when the assignment is partial.

In order to fulfill these criteria, the interactive solver may not make any assignments but just maintain the list of possible (i.e., consistent) values for each variable.

Below, we discuss our methodology, the features and benefits of our new interactive solver, and the interface for our interactive solver.

2.5.1 Methodology

Thota wrote the initial interactive solver (Section 2.3.2.3) in Common LISP, and the web interface for this solver in PHP4. For these two vastly different languages to communicate to support interactivity, the languages had to communicate via network sockets. This solution was an ingenious way to enable Common LISP to interact with PHP4 (vice-versa), but it was not robust. Because Thota's interactive solver was

server-side, the network socket(s) had to be recreated each time the server rebooted or when the LISP-PHP socket server crashed, which significantly affected the reliability of the interactive solver in practice. In order to make the interactive-solver more robust, we rewrote the interactive-solving component entirely in Java. This component now runs independently of the web interface and accesses the MySQL database directly to collect the relevant information about GTAs, courses, and constraints.

The algorithms used in the Interactive Solver were adapted from Thota's MS Project (see Chapter 2 in [Thota, 2004]). Our interactive solver has the two dual perspectives on the problem (i.e., task and resource perspectives) displayed side-by-side. In addition to all the functionalities of Thota's interface, this new interface has the features described below that firmly support interactivity.

2.5.2 Features and benefits

Rewriting the Interactive Solver has yielded many features and benefits. This new solver:

- eliminates the need of recreating the socket server when the server is rebooted or the socket server has crashed;
- provides a better interactive experience by securing a quicker response time;
- provides cross-platform support in a full fledged Java application that runs on the client-side, and is not affected by CSE's web-server downtime;
- provides secured and authenticated access (via transparent SSH tunneling) to the MySQL database;
- supports multi-user and simultaneous access and assignments;
- provides a Save/Delete/Restore assignments feature for the user; and

- allows to print the output of the GTA assignment on a single sheet.

2.5.3 Interface of the interactive solver

Figures 2.13, 2.14, and 2.15 are screen captures of the Java interactive solver. While implementing this new interactive solver, we strived to keep our interface as similar as possible to the Excel sheet used by the faculty member in charge of the assignment while integrating all functionalities provided by Thota's web-based interactive-solver [2004].

The main window, shown in Figure 2.13, displays both the task perspective and the resource perspective. As these two perspectives use the same underlying CSP model, the user is able to make assignments in either perspective and these decisions are immediately reflected on the other perspective, which is not possible in the Excel sheet.

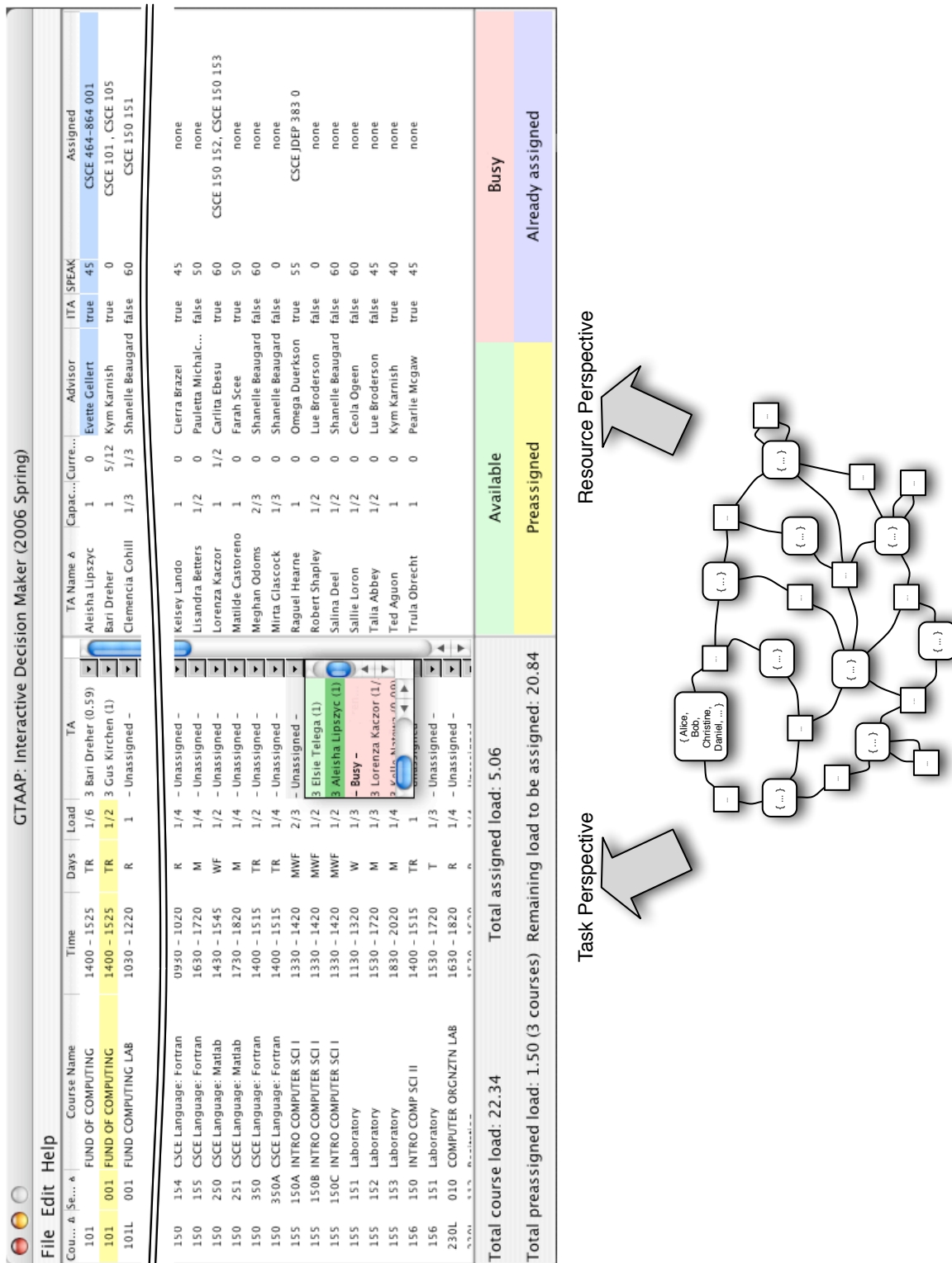


Figure 2.13: Interactive solver main screen. Two perspectives are visible in the same window pane.

Figure 2.14 shows a class being assigned a GTA and Figure 2.15 shows that GTA is assigned to class. Assignments may be made in either the task-perspective or the resource-perspective because they use the same model. Details about the (Java)

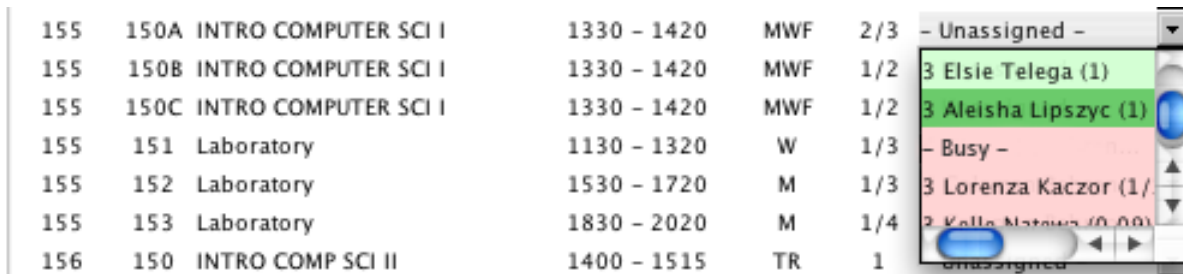


Figure 2.14: Interactive solver doing a task assignment.

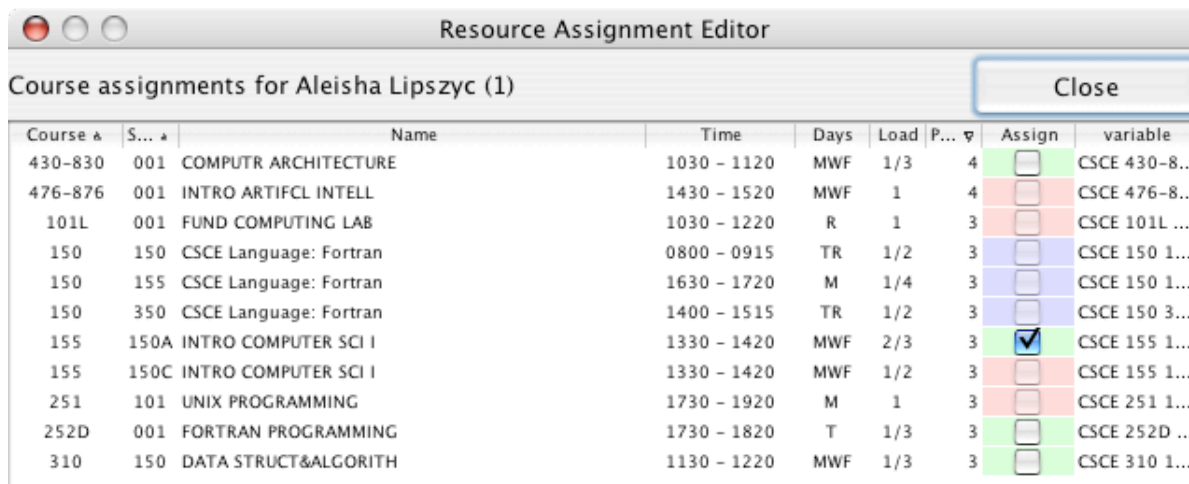


Figure 2.15: Interactive solver doing a resource assignment.

classes, its associated attributes, and methods are in Appendix D.

Summary

In this chapter we reviewed the general system architecture and discussed each of the components that we have implemented or co-developed. These components include a MySQL database to store data pertaining to GTAAP, a GTA web-page and a manager web-page for data entry, and a facility for course instructors to evaluate their assigned GTA. We have also redesigned and implemented a new interactive solver, which runs as a Java graphical interface on the manager's local computer.

Chapter 3

Conclusions and future work

We have designed and built the main components of a prototype system for the management and hiring of GTAs at CSE. Our contributions comprise a relational database, web-interfaces for GTAs and the department manager, a facility for GTA evaluations, and an interactive-problem solver. We have also deployed the system since its inception and supported its use in the department by the students and faculty, providing reliable and continuous support under tight working conditions.

Our approach to system development has been pragmatic, and we have not carry out a formal longitudinal study of usability and usefulness. However, anecdotal evidence of the value of the system is demonstrated in the form of financial support by CSE, satisfaction of the Graduate Secretary and the manager (relieved from handling application forms and massive paperwork), and enthusiastic anonymous on-line reviews from the students (see Figure 2.5). Invariably, assignments are now made quickly (from a 3 week duration after the beginning of the semester down to a day or two before the beginning of the semester), they are more stable, and, above all, they are more satisfactory to the course instructor, the GTAs, and the students in the classrooms.

This system has constantly evolved and changed over time. We have made the conscious decision to solicit users' feedback at every step of the development of the prototype and immediately addressed users' concerns communicated to us by email or via the online survey. We also systematically and immediately deployed the incremental improvements and made them available to the users. We maintained the system in use even through major transitions, preserving the data consistency throughout the changes.

If we were to redesign this system from scratch today, we would make significant changes. First, the web-interfaces available to the GTAs and the manager would have to be redesigned and reorganized to reduce clutter and facilitate navigation. Second, the database would have to be reorganized and redesigned. As we have been collecting data attributes for new functionalities, new fields are often added to the database tables and existing ones modified making the table layout 'messy.' Reorganizing the tables by decomposing or normalizing them may allow the manager to compute queries that are more complex than available in the current system. In addition to this, we recommend migrating away from the MySQL database to a database that has more powerful features such table views, transactions, and triggers.

GTAAP opens new directions for research in problem solving (e.g., collaborative strategies among the human manager and the various automated solvers), and in human-computer interaction (e.g., supporting the human manager in navigating the solution space). It also gives us an opportunity to push the envelop in system design and implementation in terms of integrating advanced scientific techniques such as Constraint Processing with technological solutions such as web-design and database integration.

In conclusion, we think that while the development of GTAAP should continue in the above listed directions, it may be time now to explore using GTAAP beyond

CSE, at UNL and other institutions, world-wide, that have expressed interest in using it.

Appendix A

Process description

Below are the steps that must be followed for using the system every semester. The sequence of these steps may vary depending on how the department chooses to conduct the process.

Before the semester starts,

1. The person maintaining GTAAP imports the data about the classes offered during the semester. This includes CSE classes and some JDEHP classes. The task is achieved by running the script

```
~gta/upload_db.pl.
```

2. The person maintaining GTAAP sets up the courses in the manager's web-interface by setting course loads and canceling courses. A link for the GTA is also provided in the GTA page

```
~gta/public_html/gta/gta_classes_instructions.php
```

to enable the GTA to enter his/her teaching preferences.

3. The secretary of the Graduate Program at CSE sends an email requesting graduate students to apply for GTA position by updating their academic record and/or expressing their teaching preferences for the courses listed for the semester. This message can be either sent to the mailing list `grads@cse.unl.edu` or directly through the GTAAP interface to the students currently employed.
4. The Graduate Committee reviews the applications to the GTA positions (the selective query facility discussed in Section 2.3.2.1 and shown in Figure 2.9 can be very useful to this end) in light of the applicants past performance, and choose the students to be hired for the semester.
5. The manager responsible for the assignment must specify/adjust the load of each course depending on the number of students enrolled in the course. He/she may cancel or add courses.

In parallel, the manager and/or the secretary of the Graduate Program at CSE must specify the students hired and their hiring capacity. Further, they must ensure that all students hired have entered correctly and completely their data. This last step must be repeated for every new students hired later on in the process.

6. The manager may choose to run the interactive solver to build solutions or the automatic solver to come up with seed solutions that he/she can improve. (Currently, assignments done interactively can be copied to, and saved in, the database).

One month before the end of the semester,

1. The manager must specify the final employment of each GTA along with the name of the supervising instructor. This information can be entered by the

manager by selecting the functionality labeled 'Employment Tasks' under the mode labeled 'GTAs.'

2. One month before the end of the semester, the secretary of the graduate program at CSE must send an email requesting all teaching instructors for the current semester (including student instructors, teaching staff, instructors from outside CSE, and CSE faculty members) to evaluate the performance of their respective assigned GTAs. Further, he/she must monitor that all evaluations have been entered before the end of the semester.

Appendix B

Database Documentation

This appendix chapter contains the MySQL table names, their attributes and types, and default values in GTAAP

We group the tables into 4 categories: Authentication and logging, GTA, Manager, and Auxiliary tables. In each section, we describe the function of each table and then provide the details of each tables.

B.1 Authentication and logging

These tables are related to user information, authentication, and access logging.

1. `auth_log`: The log of database accesses from the web interface (see Table B.1).
2. `users`: This table contains user, contact, authentication and authorization information. Each GTA and manager must be listed in this table (see Table B.2).

B.1.1 Detail description

Below, we provide the low-level details about the tables in this category.

B.1.1.1 auth_log

Table B.1: Table `auth_log`

Field	Type	Null	Default
<i>eventId</i>	int(12)	No	
date_time	datetime	No	0000-00-00 00:00:00
userId	varchar(15)	No	
ip_addr	varchar(20)	No	
url	varchar(200)	No	

eventId: This is the unique event identifier for the records in the `auth_log` table.

date_time: This is the date and time of the event in the format *YYYY-MM-DD hh:mm:ss* where the *YYYY* is the 4 digit representation of the year, *MM* is the 2 digit representation of the months, *DD* is the 2 digit representation of the day of the month, *hh* is the 24-hour representation of the hour of the day, *mm* is the 2 digit representation of the minute and *ss* is the 2 digit representation of the second.

userId: This is the `userId` (refer to Table B.2) that caused this event to be logged.

ip_addr: This is the Internet Protocol address (IPv4) used by the user that caused this event to be logged.

url: This is the Uniform Resource Locator (page) that was accessed by the user.

B.1.1.2 users

Table B.2: Table `users`

Field	Type	Null	Default
<i>id</i>	int(11)	No	
userId	varchar(15)	No	
userPw	varchar(15)	No	
acl_level	tinyint(4)	No	0
status	tinyint(4)	No	0
firstName	varchar(20)	No	
lastName	varchar(20)	No	
cse_login	varchar(10)	Yes	NULL
email	varchar(50)	No	

id: This is the unique identifier for this user.

userId: This is the username used for identification and login purposes.

userPw This is the plain-text password for this user.

acl_level: This is the user authorization level. These values should be positive integers (negative integers denote a disabled account). The values below are the authorization levels used in GTAAP.

- 1: manager
- 2: instructor
- 3: GTA

status: This is the account status. These values should be positive integers (negative integers denote a disabled account). The values below are the account status values used in GTAAP.

- 1: manager

- 2: instructor
- 3: GTA

firstName: This is the user's first name.

lastName: This is the user's last name.

cse_login: This is the user's departmental (CSE) UNIX username.

B.2 GTA

These tables are related to GTA academic data and their teaching preferences.

1. **consider_application:** This table contains the GTAs who want to be considered for positions in the upcoming semesters. (see Table B.3).
2. **exams_comps:** This table stores comprehensive exam records for Ph.D. seeking GTAs (see Table B.4).
3. **exams_qual:** This table stores qualifiers exam records for Ph.D. seeking GTAs (see Table B.5).
4. **gta_apply:** This table stores the list of GTAs and the semesters they have applied to (see Table B.6).
5. **gta_assignments:** This table stores the previous GTA assignments saved by the GTA in their academic record page (see Table B.7).
6. **gta_attendance:** This table stores the colloquia/department talks saved by the GTA in their academic record page (see Table B.8).
7. **gta_data:** This table stores the GTA academic record data (see Table B.9).

8. `gta_deficiencies`: For GTAs who have yet to completed deficiency courses, the GTAs are listed in this table along with the deficiency courses to be completed (see Table B.10).
9. `gta_gre`: The GTAs' GRE records (see also Table B.9) (see Table B.11).
10. `gta_prefs`: The GTAs' preference values for the courses offered (see Table B.12).
11. `gta_prefs_ts`: The time stamp values for when the GTAs updated their preference values. Updates to each semester are stored in different entries (see also Table B.12) (see Table B.13).
12. `gta_speak`: Table for non-citizen GTAs who are required to complete the SPEAK test (see Table B.14).
13. `gta_survey`: Data collection (GTAs' web-interface). evaluation by the GTAs (see Table B.15).

B.2.1 Detail description

Below, we provide the low-level details about the tables in this category.

B.2.1.1 `consider_application`

Table B.3: Table `consider_application`

Field	Type	Null	Default
<code>id</code>	<code>int(11)</code>	No	0
<code>c_year</code>	<code>year(4)</code>	No	0000
<code>c_semester</code>	<code>tinyint(4)</code>	No	0

id: This is the identifier for the GTA (refer to Table B.2) who wants to be considered for the semester `c_year` and `c_semester`.

c_year: This is the year the GTA would like to be considered for.

c_semester: This is the semester the GTA would like to be considered for.

B.2.1.2 exams_comps

Table B.4: Table `exams_comps`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
done	enum('true', 'false')	Yes	NULL
yr	year(4)	No	0000
mo	tinyint(3)	No	0
da	tinyint(3)	No	0

id: This is the identifier for the GTA (refer to Table B.2) who has taken the comprehensive exam.

done: This attribute is set to `true` if the GTA has completed the comprehensive exam, or `false` otherwise.

yr: This is the year in 4-digit format the GTA took the comprehensive exam.

mo: This is the month in 2-digit format the GTA took the comprehensive exam.

day: This is the day in 2-digit format the GTA took the comprehensive exam.

B.2.1.3 exams_qualis

Table B.5: Table `exams_qualis`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
a_theory	enum('na', 'hp', 'p', 'f')	No	na
a_systems	enum('na', 'hp', 'p', 'f')	No	na
a_applications	enum('na', 'hp', 'p', 'f')	No	na
a_applications_s	varchar(50)	Yes	NULL
yr	year(4)	No	0000
mo	tinyint(3)	No	0
da	tinyint(3)	No	0

id: This is the identifier for the GTA (refer to Table B.2) who has taken the qualifiers exam.

a_theory: This attribute is set to `na` if it is not applicable, `hp` if the GTA received a high pass, `p` if the GTA passed, `f` if the GTA failed the theory section for the qualifiers exam.

a_systems: This attribute is set to `na` if it is not applicable, `hp` if the GTA received a high pass, `p` if the GTA passed, `f` if the GTA failed the systems section for the qualifiers exam.

a_applications: This attribute is set to `na` if it is not applicable, `hp` if the GTA received a high pass, `p` if the GTA passed, `f` if the GTA failed the applications section for the qualifiers exam.

a_applications_s: This is a text value for the application subject area the GTA took for the qualifiers exam.

yr: This is the year in 4-digit format the GTA took the qualifiers exam.

mo: This is the month in 2-digit format the GTA took the qualifiers exam.

day: This is the day in 2-digit format the GTA took the qualifiers exam.

B.2.1.4 gta_apply

Table B.6: Table `gta_apply`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
<i>c_semester</i>	tinyint(4)	No	0
<i>c_year</i>	year(4)	No	0000
apply	enum('true', 'false')	Yes	NULL

id: This is the is identifier for the GTA (refer to Table B.2) who has applied for a position in *c_semester* and *c_year*.

c_semester: This is the semester the GTA applied for.

c_year: This is the year the GTA applied for.

apply: This attribute is set to `true` if the the GTA applied for this semester and year, or `false` otherwise.

B.2.1.5 gta_assignments

Table B.7: Table `gta_assignments`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
<i>course</i>	varchar(10)	No	
<i>a_semester</i>	enum('FALL', 'SPRING', 'SUMMER')	No	FALL
<i>a_year</i>	year(4)	No	0000
instructor	varchar(50)	Yes	NULL
duties	tinyint(3)	Yes	NULL

id: This is the identifier for the GTA (refer to Table B.2) whose teaching assignments (from the Academic Record page) is stored.

course: This is the course identifier (refer to Table B.16) of the course this GTA was previously assigned to.

a_semester: This is the semester the GTA was assigned the course *course*.

a_year: This is the year the GTA was assigned the course *course*.

instructor: This is the instructor identifier (refer to Table B.27) who supervised this GTA in the course *course*.

duties: This is the task the GTA performed (refer to Table B.26) in the course *course*.

B.2.1.6 gta_attendance

Table B.8: Table `gta_attendance`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
colloquia	tinyint(3)	No	0
ms	tinyint(3)	No	0
phd	tinyint(3)	No	0

id: This is the GTA identifier (refer to Table B.2) for which this attendance record pertains to.

colloquia: This is the number of department colloquia talks this GTA has attended.

ms: This is the number of department M.S. Thesis defenses this GTA has attended.

phd: This is the number of department Ph.D. Dissertation defenses this GTA has attended.

B.2.1.7 gta_data

Table B.9: Table `gta_data`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
lastmodified	datetime	Yes	NULL
advisor	int(11)	No	0
program	tinyint(3)	No	0
admitted_semester	enum('FALL', 'SPRING', 'SUMMER')	No	FALL
admitted_year	year(4)	No	0000
graduation_semester	enum('FALL', 'SPRING', 'SUMMER')	Yes	NULL
graduation_year	year(4)	Yes	NULL
support_gta	tinyint(3)	Yes	NULL
support_gra	tinyint(3)	Yes	NULL
support_amount	int(10)	Yes	NULL
ugrd_GPA	float	Yes	NULL
grad_GPA	float	Yes	NULL
ita	tinyint(3)	Yes	NULL
ta_ship	enum('true', 'false')	Yes	NULL
ra_ship	enum('true', 'false')	Yes	NULL
ta_ship_load	smallint(3)	No	999
ra_ship_load	smallint(3)	No	999
thesis_project_title	varchar(100)	Yes	NULL
thesis_project_advisor	varchar(100)	Yes	NULL
research_experience	text	Yes	NULL
npublications	tinyint(3)	No	0
foreign_student	tinyint(1)	No	1

id: This is the identifier for the GTA (refer to Table B.2) this academic record pertains to.

lastmodified: This is the date and time this record was last updated in the format `YYYY-MM-DD hh:mm:ss` where the `YYYY` is the 4 digit representation of the year, `MM` is the 2 digit representation of the months, `DD` is the 2 digit representation of the day of the month, `hh` is the 24-hour representation of the

hour of the day, *mm* is the 2 digit representation of the minute and *ss* is the 2 digit representation of the second.

advisor: This is the identifier for the advisor (refer to Table B.27) of this GTA.

program: This is the program identifier (refer to Table B.28) this GTA is enrolled as.

admitted_semester: This is the semester this GTA was admitted into the program.

admitted_year: This is the year this GTA was admitted into the program.

graduation_semester: This is the expected graduation semester for this GTA.

graduation_year: This is the expected graduation year for this GTA.

support_gta: This is the number of semesters this GTA has received support as a teaching assistant from the department.

support_gra: This is the number of semesters this GTA has received support as a research assistant from the department.

support_amount: This is the most recent dollar amount of support received by this GTA (This field is deprecated and should not be used).

ugrd_GPA: This is this GTA's undergraduate grade-point average.

grad_GPA: This is this GTA's graduate grade-point average.

ita: This is this GTA's ITA (International Teaching Assistants) qualification status (refer to Table B.29).

ta_ship: This attribute is set to **true** if this GTA is currently supported as a teaching assistant, or **false** otherwise.

ra_ship: This attribute is set to `true` if this GTA is currently supported as a research assistant, or `false` otherwise.

ta_ship_load: This is the expected current workload of this GTA as a teaching assistant (values 0.00, 0.16, 0.25, 0.33, 0.50, 0.66, 0.75, 0.83, 1.00).

ra_ship_load: This is the expected current workload of this GTA as a research assistant (values 0.00, 0.16, 0.25, 0.33, 0.50, 0.66, 0.75, 0.83, 1.00).

thesis_project_title: This is the text value for this GTA's thesis/project title.

thesis_project_advisor: This is is the text value for the candidate GTA's advisor (this should be the same as the advisor attribute; this field is deprecated and should not be used).

research_experience: This is a variable length text value for this GTA's research experience.

npublications: This is the number of publications this GTA has published to-date.

foreign_student: This attribute is set to 1 if the GTA is a foreign student, or 0 otherwise.

B.2.1.8 gta_deficiencies

Table B.10: Table `gta_deficiencies`

Field	Type	Null	Default
id	int(11)	No	0
deficiency	int(11)	No	0

id: This is the identifier for the GTA (refer to Table B.2) who has yet to complete deficiency courses.

deficiency: This is the deficiency course (this list of deficiency courses are in Table B.25).

B.2.1.9 gta_gre

Table B.11: Table `gta_gre`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
<i>greType</i>	enum('VERBAL', 'QUANTITATIVE', 'ANALYTICAL', 'SUBJECT')	No	VERBAL
subjectArea	varchar(50)	Yes	NULL
score	int(11)	No	0
percent	float(10,2)	No	0.00

***id*:** This is the identifier for GTA (refer to Table B.2) for which this GRE record pertains to.

***greType*:** This attribute is either VERBAL, QUANTITATIVE, ANALYTICAL, or SUBJECT depending on the GRE area.

subjectArea: This attribute is the text value for the subject area (if *greType* is SUBJECT).

score: This is the GTA's score (this may be different depending on the subject area) on the specific area of GRE.

percent: This is the score percentile this GTA achieved.

B.2.1.10 gta_prefs

Table B.12: Table `gta_prefs`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
<i>c_semester</i>	tinyint(4)	No	0
<i>c_year</i>	year(4)	No	0000
<i>classId</i>	int(11)	No	0
enrolled	enum('true', 'false')	No	true
preference	tinyint(3)	No	0
justification	varchar(50)	Yes	NULL

id: This is the identifier for the GTA (refer to Table B.2) this preference value pertains to.

c_semester: This is the semester this preference is valid for.

c_year: This is the year this preference is valid for.

classId: This is the identifier for the course (refer to Table B.16) this preference is for.

enrolled: This attribute is set to `true` if this GTA is enrolled in the course corresponding to the *classId*, or `false` otherwise.

preference: This is the preference value ranging from 0 (unable to handle) to 5 (best choice) for the course-preference pair for this GTA.

justification: This is a text value justifying why this GTA is unable to serve this course (if a preference value 0 is specified).

B.2.1.11 gta_prefs_ts

Table B.13: Table `gta_prefs.ts`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
<i>c_semester</i>	tinyint(4)	No	0
<i>c_year</i>	year(4)	No	0000
lastmodified	datetime	No	0000-00-00 00:00:00

id: This is identifier for the GTA (refer to Table B.2) this preference timestamp is for.

c_semester: This is the semester that this preference timestamp is for.

c_year: This is the year that this preference timestamp is for.

lastmodified: This is the date and time of this preference timestamp in the format `YYYY-MM-DD hh:mm:ss` where the `YYYY` is the 4 digit representation of the year, `MM` is the 2 digit representation of the months, `DD` is the 2 digit representation of the day of the month, `hh` is the 24-hour representation of the hour of the day, `mm` is the 2 digit representation of the minute and `ss` is the 2 digit representation of the second.

B.2.1.12 gta_speak

Table B.14: Table `gta_speak`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
yr	year(4)	No	0000
mo	tinyint(3)	No	0
da	tinyint(3)	No	0
score	tinyint(3)	No	0

id: This is the identifier for the GTA (refer to Table B.2) that pertains to this SPEAK test.

yr: This is the year that this GTA took the SPEAK test.

mo: This is the month that this GTA took the SPEAK test.

da: This is the day that this GTA took the SPEAK test.

score: This is the GTA's score (maximum 60) for the SPEAK test.

B.2.1.13 gta_survey

Table B.15: Table `gta_survey`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
<i>survey_id</i>	int(11)	No	0
<code>navigate_ease</code>	tinyint(4)	Yes	NULL
<code>navigate_flex</code>	tinyint(4)	Yes	NULL
<code>navigate_rbst</code>	tinyint(4)	Yes	NULL
<code>dataentry_ease</code>	tinyint(4)	Yes	NULL
<code>dataentry_flex</code>	tinyint(4)	Yes	NULL
<code>dataentry_rbst</code>	tinyint(4)	Yes	NULL
<code>online_paper</code>	enum('online', 'paper')	Yes	NULL
<code>revised</code>	tinyint(4)	Yes	NULL
<code>superfluous_p</code>	enum('true', 'false')	Yes	NULL
<code>superfluous_c</code>	text	Yes	NULL
<code>missinginfo_p</code>	enum('true', 'false')	Yes	NULL
<code>missinginfo_c</code>	text	Yes	NULL
<code>other_comments</code>	text	Yes	NULL

id: This is the identifier for the GTA (refer to Table B.2) associated with this survey result.

survey_id: This is the survey identifier number. Each semester should have an incremented *survey_id* from the previous semesters.

`navigate_ease`: This is the response for the navigation ease ranging from 0 (bad) to 5 (very good).

`navigate_flex`: This is the response for the navigation flexibility ranging from 0 (bad) to 5 (very good).

`navigate_rbst`: This is the response for the navigation robustness ranging from 0 (bad) to 5 (very good).

dataentry_ease: This is the response for the data entry ease ranging from 0 (bad) to 5 (very good).

dataentry_flex: This is the response for the data entry flexibility ranging from 0 (bad) to 5 (very good).

dataentry_rbst: This is the response for the data entry robustness ranging from 0 (bad) to 5 (very good).

online_paper: This attribute is set to `online` if the GTA prefers to submit an online GTA application, or `paper` if the GTA prefers to submit a paper copy of the GTA application.

revised: This is the number of times that this GTA revised his/her application.

superfluous_p: This attribute is set to `true` if the GTA felt there were any superfluous questions, or `false` otherwise.

superfluous_c: This is a text value explanation if *superfluous_p* was set to `true`.

missinginfo_p: This attribute is set to `true` if the GTA felt there were any missing questions, or `false` otherwise.

missinginfo_c: This is a text value explanation if *missinginfo_p* was set to `true`.

other_comments: This is a text value attribute for the GTA to submit additional comments about their experience using GTAAP.

B.3 Manager

These tables are related to courses, the constraints setup, hiring information and GTA evaluations.

1. **classes**: All courses offered by the department (since Spring 2002). are stored in this table. (see Table B.16).
2. **class_confinements**: Lists courses that must have the GTA assigned to them be exclusive to all courses in the set (associated by a unique identifier). i.e., GTAs cannot be also assigned to courses outside this set (see Table B.17).
3. **consider_application_semesters**: This table holds all semesters that can be applied for, or have been filled out in the past. This table is only read and not written to (see Table B.18).
4. **facEvals**: Faculty evaluation of GTAs for each semester (see Table B.19).
5. **hired**: General hired information (hired, capacity, salary) about the GTAs (see also Table B.21) (see Table B.20).
6. **hired_info**: Specific hired information (course assigned to, task, task load, supervising instructor) (see also Table B.20). (see Table B.21).
7. **jassignments**: The Java Interactive Solver uses this table to store the saved scenarios. (see Table B.22).
8. **preassignment**: Preassignments are made and stored here. The constraint solvers must honor these preassignments (see Table B.23).
9. **same_ta**: The same-TA table. Courses listed here must have the GTA assigned to all courses in the set (associated by a unique identifier) (see Table B.24).

B.3.1 Detail description

Below, we provide the low-level details about the tables in this category.

B.3.1.1 classes

Table B.16: Table `classes`

Field	Type	Null	Default
<i>c_semester</i>	tinyint(4)	No	0
<i>c_year</i>	year(4)	No	0000
<i>id</i>	int(11)	No	
parent	int(11)	No	-1
classId	varchar(10)	No	
name	varchar(60)	No	
section	varchar(5)	No	
startTime	varchar(4)	No	
endTime	varchar(4)	No	
day_m	enum('true', 'false')	No	true
day_t	enum('true', 'false')	No	true
day_w	enum('true', 'false')	No	true
day_r	enum('true', 'false')	No	true
day_f	enum('true', 'false')	No	true
day_s	enum('true', 'false')	No	true
courseLoad	float	No	1
canceled	enum('true', 'false')	No	false
startDate	varchar(8)	No	00000000
endDate	varchar(8)	No	00000000
class_type	int(11)	No	0
type_lec	enum('true', 'false')	Yes	NULL
type_grad	enum('true', 'false')	Yes	NULL
type_lab	enum('true', 'false')	Yes	NULL
type_rec	enum('true', 'false')	Yes	NULL
type_s_lec	enum('true', 'false')	Yes	NULL
type_s_grad	enum('true', 'false')	Yes	NULL
type_s_lab	enum('true', 'false')	Yes	NULL
type_s_rec	enum('true', 'false')	Yes	NULL
instructor_id	int(11)	Yes	99999

c_semester: This is the semester this course is offered.

c_year: This is the year this course is offered.

id: This is the course identifier used across the database to identify this specific course.

parent: This is the parent course for this course. A parent course is the lecture course associated with a lab or recitation course. If this course has no parent, a value of '-1' is set.

classId: This is the course number for this course.

name: This is the descriptive name for this course.

section: This is the section number for this course.

startTime: This is the start time for this course in *HHMM* format. *HH* is the 24-hour representation of the hour and *MM* is the minute of the hour (2-digit) representation of minute.

endTime: This is the end time for this course in *HHMM* format. *HH* is the 24-hour representation of the hour and *MM* is the minute of the hour (2-digit) representation of minute.

day_m: This attribute is set to **true** if it meets on Monday, or **false** otherwise.

day_t: This attribute is set to **true** if it meets on Tuesday, or **false** otherwise.

day_w: This attribute is set to **true** if it meets on Wednesday, or **false** otherwise.

day_r: This attribute is set to **true** if it meets on Thursday, or **false** otherwise.

day_f: This attribute is set to **true** if it meets on Friday, or **false** otherwise.

day_s: This attribute is set to **true** if it meets on Saturday, or **false** otherwise.

courseLoad: This is the commitment level expected of the GTA for serving this course. The course load is defined by the double values 0.00, 0.16, 0.25, 0.33, 0.50, 0.66, 0.75, 0.83, 1.00.

canceled: This attribute is set to **true** if the course is cancelled, or **false** otherwise.

startDate: This is the start date for courses that do not begin at the beginning of the semester. The format is *YYYYMMDD* where *YYYY* is the 4 digit representation of the year, *MM* is the 2 digit representation of the months, and *DD* is the 2 digit representation of the day of the month.

endDate: This is the end date for courses that do not complete at the end of the semester. The format is *YYYYMMDD* where *YYYY* is the 4 digit representation of the year, *MM* is the 2 digit representation of the months, and *DD* is the 2 digit representation of the day of the month.

class_type: This is the course type. (This field is deprecated and should not be used).

type_lec: This attribute is set to **true** if this course is a lecture course, or **false** otherwise.

type_grad: This attribute is set to **true** if this course is a grading course, or **false** otherwise.

type_lab: This attribute is set to **true** if this course is a lab course, or **false** otherwise.

type_rec: This attribute is set to **true** if this course is a recitation course, or **false** otherwise.

type_s_lec: This attribute is set to `true` if this course is a short lecture course, or `false` otherwise. (This field is deprecated and should not be used).

type_s_grad: This attribute is set to `true` if this course is a short grading course, or `false` otherwise. (This field is deprecated and should not be used).

type_s_lab: This attribute is set to `true` if this course is a short lab course, or `false` otherwise. (This field is deprecated and should not be used).

type_s_rec: This attribute is set to `true` if this course is a recitation course, or `false` otherwise. (This field is deprecated and should not be used).

instructor_id: is identifier used to identify the instructor from Table B.27.

B.3.1.2 class_confinements

Table B.17: Table `class_confinements`

Field	Type	Null	Default
<code>c_semester</code>	<code>tinyint(4)</code>	No	0
<code>c_year</code>	<code>year(4)</code>	No	0000
<code>class_id</code>	<code>int(11)</code>	No	0
<code>confinement</code>	<code>int(10)</code>	No	0

c_semester: This is the semester which this confinement applies to.

c_year: This is the year which this confinement applies to.

class_id: This is the `class_id` (refer to Table B.16) of the confined course.

confinement: This is the (semester) unique confinement identifier. A group of confined courses must have the same confinement identifier number.

B.3.1.3 consider_application_semesters

Table B.18: Table `consider_application_semesters`

Field	Type	Null	Default
<code>c_year</code>	<code>year(4)</code>	No	0000
<code>c_semester</code>	<code>tinyint(4)</code>	No	0
<code>dates</code>	<code>varchar(50)</code>	Yes	NULL
<code>comments</code>	<code>varchar(50)</code>	Yes	NULL

c_year: This is the year in *YYYY* 4 digit format.

c_semester: This is the semester.

dates: This is a text field that represents the start and end date. The convention used is “*MM/DD/YY - mm/dd/yy*” where *MM* is the start month, *DD* is the start day, *YY* is the start year, *mm* is the end month, *dd* is the end month, and *yy* is the end year.

comments: This is a text field for comments.

B.3.1.4 facEvals

Table B.19: Table facEvals

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
<i>instructor</i>	int(11)	No	0
<i>courseID</i>	int(11)	No	0
<i>task</i>	tinyint(3)	No	0
<i>c_semester</i>	tinyint(4)	No	0
<i>c_year</i>	year(4)	No	0000
knowledge	tinyint(4)	Yes	NULL
communicate	tinyint(4)	Yes	NULL
effort	tinyint(4)	Yes	NULL
plan	tinyint(4)	Yes	NULL
overall	tinyint(4)	Yes	NULL
witness	int(11)	Yes	NULL
comments	text	Yes	NULL
studentComments	text	Yes	NULL
recommend	tinyint(4)	Yes	NULL
date_time	datetime	Yes	0000-00-00 00:00:00

id: This is the identifier for the GTA (refer to Table B.2) this evaluation pertains to.

instructor: This is the identifier for the instructor (refer to Table B.27) who evaluated this GTA.

courseID: This is the course identifier (refer to Table B.16) this evaluation pertains to.

task: This is the task identifier (refer to Table B.26) this evaluation pertains to.

c_semester: This is the semester this evaluation pertains to.

c_year: This is the year in 4-digit representation that this evaluation pertains to.

knowledge: This is the rating for the GTA's knowledge on the course material ranging from 1 (little/bad) to 6 (very good). A value 0 represents an unknown value.

communicate: This is the rating for the GTA's communication skills ranging from 1 (little/bad) to 6 (very good). A value 0 represents an unknown value.

effort: This is the rating for the GTA's quality of work ranging from 1 (little/bad) to 6 (very good). A value 0 represents an unknown value.

plan: This is the rating for the GTA's planning skills ranging from 1 (little/bad) to 6 (very good). A value 0 represents an unknown value.

overall: This is the rating for the GTA's overall evaluation ranging from 1 (little/bad) to 6 (very good). A value 0 represents an unknown value.

witness: This is the number of times the course instructor witnessed/supervised the grading/lab/recitation instructor (presence/grading).

comments: This is a text field (variable length) for the instructor's comments on the GTA.

studentComments: This is a text field (variable length) for the students' comments on the GTA.

recommend: This is the overall recommendation for the GTA. A value 4 represents a strong recommend, a 3 represents a recommend for another course, a 2 represents a recommend but only for grading, a 1 represents do not recommend, and a 0 represents an unknown value.

date_time: This is the date and time of that this evaluation was submitted in the format *YYYY-MM-DD hh:mm:ss* where the *YYYY* is the 4 digit representation of the year, *MM* is the 2 digit representation of the months, *DD* is the 2 digit

representation of the day of the month, *hh* is the 24-hour representation of the hour of the day, *mm* is the 2 digit representation of the minute and *ss* is the 2 digit representation of the second.

B.3.1.5 hired

Table B.20: Table `hired`

Field	Type	Null	Default
<code>id</code>	<code>int(11)</code>	No	0
<code>c_semester</code>	<code>tinyint(4)</code>	No	0
<code>c_year</code>	<code>year(4)</code>	No	0000
<code>hired</code>	<code>enum('true', 'false')</code>	No	<code>true</code>
<code>capacity</code>	<code>float</code>	No	0
<code>salary</code>	<code>float</code>	Yes	0

id: This is the identifier for the GTA (refer to Table B.2).

c_semester: This is the semester.

c_year: This is the year.

hired: This is set to `true` if the GTA is hired for this semester and year, `false` otherwise.

capacity: This is the capacity which the GTA is expected to work at (0.00, 0.16, 0.25, 0.33, 0.50, 0.66, 0.75, 0.83, 1.00).

salary: This is the salary the GTA receives per semester.

B.3.1.6 hired_info

Table B.21: Table `hired_info`

Field	Type	Null	Default
<code>id</code>	<code>int(11)</code>	No	0
<code>c_semester</code>	<code>tinyint(4)</code>	No	0
<code>c_year</code>	<code>year(4)</code>	No	0000
<code>courseId</code>	<code>int(11)</code>	No	0
<code>task</code>	<code>tinyint(3)</code>	No	0
<code>task_load</code>	<code>float</code>	No	0
<code>instructor</code>	<code>int(11)</code>	Yes	99999

id: This is the identifier for the GTA (refer to Table B.2).

c_semester: This is the semester.

c_year: This is the year.

courseId: This is the course identifier (refer to Table B.16)

task: This is the task the GTA performed (refer to Table B.26).

task_load: This is the GTA's expected load with values 0.00, 0.16, 0.25, 0.33, 0.50, 0.66, 0.75, 0.83, 1.00.

instructor: This is the instructor identifier (refer to Table B.27) who supervises this class.

B.3.1.7 jassignments

Table B.22: Table `jassignments`

Field	Type	Null	Default
<code>c_semester</code>	<code>tinyint(4)</code>	No	0
<code>savedId</code>	<code>int(11)</code>	No	0
<code>savedName</code>	<code>varchar(50)</code>	No	0
<code>c_year</code>	<code>year(4)</code>	No	0000
<code>c_semester</code>	<code>tinyint(4)</code>	No	0
<code>classId</code>	<code>int(11)</code>	No	0
<code>gtaId</code>	<code>int(11)</code>	No	0

savedId: This is the saved scenario's identifier. All GTA assignments in this scenario have the same `savedId`.

savedName: This is the saved scenario's name. All GTA assignments in this scenario have the same `savedName`.

c_semester: This is the semester the saved scenario is for.

c_year: This is the year the saved scenario is for.

classId: This is the course identifier (refer to Table B.16) of the course saved in this scenario.

gtaId: This is the GTA identifier (refer to Table B.2) of the GTA saved in this scenario.

B.3.1.8 preassignment

Table B.23: Table `preassignment`

Field	Type	Null	Default
<code>gta_id</code>	<code>int(11)</code>	Yes	NULL
<i><code>class_id</code></i>	<code>int(11)</code>	No	0
<code>c_semester</code>	<code>tinyint(4)</code>	No	0
<code>c_year</code>	<code>year(4)</code>	No	0000

`gta_id`: This is the identifier for the GTA (see Table B.2).

***`class_id`*:** This is the class identifier (see Table B.16).

`c_semester`: This is the semester.

`c_year`: This is the year.

B.3.1.9 same_ta

Table B.24: Table `same_ta`

Field	Type	Null	Default
<code>c_semester</code>	<code>tinyint(4)</code>	No	0
<code>c_year</code>	<code>year(4)</code>	No	0000
<code>class_id</code>	<code>int(11)</code>	No	0
<code>sameta</code>	<code>int(10)</code>	No	0

`c_semester`: This is the semester the same-TA constraint applies to.

`c_year`: This is the year the same-TA constraint applies to.

`class_id`: This is the course identifier (refer to Table B.16) of the course with the same-TA constraint.

`sameta`: This is the (semester) unique same-TA identifier. A group of same-TA courses must have the same `sameta` identifier number.

B.4 Auxiliary tables

These tables are read-only to provide frequently accessed information.

1. **deficiencies**: The list of all possible deficiency courses. Deficiencies are courses that a GTA must complete before completing the program (see also Table B.10). This table is only read and not written to (see Table B.25).
2. **duty**: A list of duties a GTA may have (course instructor, lab instructor, grader, recitation instructor, other). This table is only read and not written to (see Table B.26).
3. **faculty**: List of faculty members, staff, lecturers and other instructors who may complete a GTA evaluation (see Table B.27).
4. **grad_program**: List of graduate programs. This table is only read and not written to (see Table B.28).
5. **ita**: International Teaching-Assistant Association qualifications values. This table is only read not written to (see Table B.29).
6. **semesters**: The list of semesters (Fall, Spring, Summer).. This table is only read and not written to (see Table B.30).

B.4.1 Detail description

Below, we provide the low-level details about the tables in this category.

B.4.1.1 deficiencies

Table B.25: Table `deficiencies`

Field	Type	Null	Default
id	tinyint(3)	No	0
value	varchar(30)	No	

id: This is the identifier for the deficiency course.

value: This is the text value for the deficiency course (the course name and number, e.g. CSCE 322).

B.4.1.2 duty

Table B.26: Table `duty`

Field	Type	Null	Default
id	tinyint(3)	No	0
value	varchar(30)	No	

id: This is the identifier for the task.

value: This is the text value for the task (the task name, e.g. Course instructor).

B.4.1.3 faculty

Table B.27: Table `faculty`

Field	Type	Null	Default
<i>id</i>	int(11)	No	0
rank	tinyint(3)	No	0
firstName	varchar(20)	Yes	NULL
lastName	varchar(20)	Yes	NULL
cse.login	varchar(15)	Yes	NULL

id: This is the unique identifier for the instructor.

rank: This is the title/rank associated with this instructor.

- 0: professor (tenure)
- 1: associate professor (tenure)
- 2: assistant professor (tenure)
- 3: professor (research)
- 4: associate professor (research)
- 5: assistant professor (research)
- 6: Senior lecturer
- 7: Lecturer
- 8: Instructor
- 9: GTA Instructor

firstName: This is the instructor's first name.

lastName: This is the instructor's last name.

cse_login: This is the instructor's departmental (CSE) UNIX username.

B.4.1.4 grad_program

Table B.28: Table `grad_program`

Field	Type	Null	Default
<i>id</i>	tinyint(3)	No	0
program	varchar(20)	Yes	NULL

id: This is the identifier for the program.

program: This is the text value for the programs offered at the department.

B.4.1.5 ita

Table B.29: Table `ita`

Field	Type	Null	Default
id	tinyint(3)	No	0
value	varchar(30)	No	

id: This is the identifier for the ITA value.

value: This is the text value for the ITA qualification status (e.g., passed, failed).

B.4.1.6 semesters

Table B.30: Table `semesters`

Field	Type	Null	Default
semester	varchar(10)	Yes	NULL

semesters: This is the text value for the available semesters (name).

Appendix C

Documentation for the Web-interface and SQL Queries

This appendix shows the web-interface file structure and hierarchy. We also list the MySQL query statements and explains their meaning and significance in the system.

C.1 Web-interface file structure and hierarchy

Below we give a brief summary of the directory structures listed:

- The `configs` directory contains configuration files used by the system. These files provide predefined values for the system's default value for the semester, define the paths to the search algorithms, and define the authentication credentials required to access the MySQL database.
- The `gta` directory contains files for the GTA web-interface. These files provide the functionalities described in Section 2.3.1.
- The `manager` directory contains files for the manager web-interface. These files provide the functionalities described in Section 2.3.2.

- The `fac_evals` directory contains files used for instructors evaluations of GTAs as described in Section 2.4.
- The `include` directories (at the root or nested below the `gta` or `manager` directories) are PHP4 functions that support the features for the GTA or manager web-access built into our system.
- The `perl_modules` directory contains Perl modules used by GTAAP that CSE does not have installed on the server.
- The `scripts` directory contain Perl scripts that assists us in the database queries.
- The `db` directory contain the MySQL table structures and the SQL queries to build the database tables.

Below is the file structure and hierarchy for the web-interface.

```
public_html
|-- admin
|   |-- frame_layout
|   |   |-- class_opts.php
|   |   |-- gta_opts.php
|   |   |-- main.php
|   |   |-- main_opts.php
|   |   |-- search_opts.php
|   |-- index.php
|-- array.php
|-- blank.html
|-- check_acl.inc.php
|-- config.inc.php
|-- configs
|   |-- CONFIG -> CONFIG-live
|   |-- CONFIG-demo
|   |-- CONFIG-dev
|   |-- CONFIG-live
|   |-- Makefile
|   |-- _htaccess_gta_template
```



```

| |-- _htaccess_manager_template
| |-- _htaccess_template
| |-- change.pl
| |-- config.inc.php
| |-- config.lisp
| |-- config_template.lisp
| |-- config_template.php
| |-- config_template.pl
| |-- configure-load.lisp
| |-- copy.pl
| |-- header_template.php
| |-- make_template.lisp
| |-- manager_index_template.php
| |-- ownnits_template.lisp
| |-- read-data_global-var_template.lisp
| |-- replicate_db.pl
| |-- set-config.pl
| '-- setup_tmp.pl
|-- contact_us.php
|-- db
| |-- README
| |-- accessdb.sh
| |-- auth_log.sql
| |-- backup-20031116.sql.gz
| |-- class_confinements.sql
| |-- classes
| | |-- classes_populate_fall03.sql
| | |-- classes_populate_spring03.sql
| | |-- summer03
| | | |-- 1st 5wks.sql
| | | |-- 2nd 5wks.sql
| | | |-- 8weeks.sql
| | | |-- parse.pl
| | | |-- presession.txt
| | | '-- summer-files
| | | |-- 1st 5wks.txt
| | | |-- 2nd 5wks.txt
| | | |-- 8wk.txt
| | | '-- presession.txt
| | '-- summer04
| | |-- 1st 5wks.sql
| | |-- 2nd 5wks.sql
| | |-- 8weeks.sql

```

```
| |         |-- parse.pl
| |         |-- presession.txt
| |         '-- summer-files
| |             |-- 1st 5wks.txt
| |             |-- 2nd 5wks.txt
| |             |-- 8wk.txt
| |         '-- presession.txt
|-- classes.sql
|-- consider_application.sql
|-- dump
| |     '-- database.sql
|-- exams_.sql
|-- facEvals.sql
|-- faculty.sql
|-- faculty_populate.sql
|-- gta_apply.sql
|-- gta_assignments.sql
|-- gta_attendance.sql
|-- gta_data.sql
|-- gta_deficiencies.sql
|-- gta_gre.sql
|-- gta_prefs.sql
|-- gta_prefs_ts.sql
|-- gta_speak.sql
|-- gta_survey.sql
|-- hired.sql
|-- hired_info.sql
|-- jassignments.sql
|-- misc.sql
|-- preassignment.sql
|-- proposals.sql
|-- users.sql
|-- users_populate.sql
|-- fac_evals
| |     |-- auth_user.php
| |     |-- evaluate.php
| |     |-- form.php
| |     |-- index.php
|-- find.sh
|-- footer.inc.php
|-- functions.inc.php
|-- gta
| |     |-- email_webmaster.php
```

```

| |-- gta_classes.php
| |-- gta_classes_frames.php
| |-- gta_classes_help.html
| |-- gta_classes_instructions.php
| |-- gta_data.php
| |-- gta_include
| | |-- db
| | | |-- insert_gta_courses.inc.php
| | | '-- insert_gta_dta.inc.depreciated.php
| | |-- forms
| | | |-- consider_application.inc.php
| | | |-- get_survey_data.inc.php
| | | |-- include.php
| | | |-- list_courses_form.inc.php
| | | '-- survey_form.inc.php
| | |-- function_template.INC.php
| | |-- html
| | | |-- gta_email_data.inc.php
| | | '-- gta_navigator.inc.php
| | |-- include.php
| | |-- make.pl
| | '-- register
| | '-- register_form1.inc.php
| |-- index.php
| |-- new_register.php
| |-- register.php
| |-- survey.php
| '-- update_info.php
|-- gta-evals
| |-- FacEval.php
| '-- index.php
|-- header.inc.php
|-- images
| |-- emailbutton.jpg
| |-- prettyprintpagebutton.jpg
| |-- printbutton.jpg
| |-- printer.jpg
| |-- printpagebutton.jpg
| '-- qmark.gif
|-- include
| |-- auth
| | |-- auth.inc.php
| | '-- check_acl.inc.php

```

```

| |-- db
| | |-- db_query.inc.php
| | |-- insert_gta_data.inc.php
| | |-- insert_new_user.inc.php
| | |-- uname2uid.inc.php
| | |-- unique_userId.inc.php
| |-- forms
| | |-- duty_menu.inc.php
| | |-- gta_data.inc.php
| | |-- login.inc.php
| | |-- register.inc.php
| | |-- select_course.inc.php
| | |-- select_deficiencies.inc.php
| | |-- select_faculty.inc.php
| | |-- select_ita.inc.php
| | |-- select_phd_qualifications.inc.php
| | |-- select_program.inc.php
| | |-- select_ra_ship.inc.php
| | |-- select_semester.inc.php
| | |-- select_ta_ship.inc.php
| | |-- update_info.inc.php
| |-- function_template.INC.php
| |-- html
| | |-- gta_show_data.inc.php
| |-- include.php
| |-- make.pl
| |-- masks
| | |-- get_className.inc.php
| | |-- get_class_name.inc.php
| | |-- get_class_name_working.inc.php
| | |-- get_deficiency_name.inc.php
| | |-- get_dutyName.inc.php
| | |-- get_email.inc.php
| | |-- get_fac_name.inc.php
| | |-- get_flName.inc.php
| | |-- get_gpa.inc.php
| | |-- get_gta_data.inc.php
| | |-- get_ita_name.inc.php
| | |-- get_lfName.inc.php
| | |-- get_myId.inc.php
| | |-- get_program_name.inc.php
| | |-- get_ra_ship_load.inc.php
| | |-- get_ta_ship_load.inc.php

```

```
| |   '-- get_usersLastId.inc.php
| |-- misc
| |   |-- check_validEmail.inc.php
| |   |-- check_valid_gta_data.inc.php
| |   |-- class_id2name.inc.php
| |   |-- duty_name.inc.php
| |   |-- invalid_user.inc.php
| |   '-- semester_name.inc.php
| |-- session
| |   |-- end_session.inc.php
| |   '-- reg_session.inc.php
| '-- test.php
|-- index.php
|-- information
| |-- ita.html
| |-- semesters_supported.html
| '-- speak.html
|-- login.php
|-- logout.php
|-- lost_pw.php
|-- make.pl
|-- manager
| |-- alisp_iface.sh
| |-- checkall.js
| |-- class_add_rm.php
| |-- class_add_rm_choose_semester.php
| |-- class_confinement.php
| |-- class_confinement_choose_semester.php
| |-- class_dependencies.php
| |-- class_dependencies_choose_semester.php
| |-- class_edit.php
| |-- class_edit_ajax.js
| |-- class_edit_choose_semester.php
| |-- class_group_confinement.php
| |-- class_group_confinement_choose_semester.php
| |-- class_group_sameta.php
| |-- class_group_sameta_choose_semester.php
| |-- class_preassign.php
| |-- class_preassign_choose_semester.php
| |-- class_preference_ranking.php
| |-- compose_rater_email.php
| |-- frame_layout
| |   |-- advanced_options.php
```

```
| | |-- blank.html
| | |-- blank.php
| | |-- class_opts.php
| | |-- courses.html
| | |-- edit_constraints_opts.php
| | |-- employment_opts.php
| | |-- evals_legend.php
| | |-- gta_opts.php
| | |-- gtas.html
| | |-- interactive_selections_opts.php
| | |-- lispfuns.inc.php
| | |-- main.php
| | |-- main_opts.php
| | |-- search_opts.php
| | |-- sys_admin_opts.php
| | '-- test.php
|-- gta-search
| | |-- results.txt
| | |-- search_era.pl
| | |-- search_lisp
| | | |-- era.lisp
| | | |-- systematic_dsbt.lisp
| | | '-- systematic_rrbt.lisp
| | |-- search_local.pl
| | |-- search_rl.pl
| | |-- search_systematic_dsbt.pl
| | '-- search_systematic_rrbt.pl
|-- gta_add.php
|-- gta_change_view.php
|-- gta_del.php
|-- gta_edit.php
|-- gta_edit_employment_history.php
|-- gta_edit_multiple_employment_history.php
|-- gta_employment_history.php
|-- gta_evaluations.php
|-- gta_evaluations_new.php
|-- gta_historical_decider.php
|-- gta_list.php
|-- gta_list.php-dumb
|-- gta_list_consider.php
|-- gta_list_filter.php
|-- gta_list_filter_query.php
|-- gta_list_semester.php
```

```
| |-- gta_list_semester_action.php
| |-- gta_multiple_edit_ajax.js
| |-- gta_preassign.php
| |-- gta_preference_ranking.php
| |-- gta_pull.php
| |-- gta_user_all_evaluations.php
| |-- gta_view_prefs.php
| |-- i_selections_gta_list.php
| |-- index.php
| |-- lisp-interface
| | |-- blank.html
| | |-- class_course.inc.php
| | |-- class_gta.inc.php
| | |-- commonfuncs.inc.php
| | |-- config.pl
| | |-- configvars.inc.php
| | |-- courses-new.php
| | |-- courses-tst.php
| | |-- courses.php
| | |-- courses1.php
| | |-- courses2.php
| | |-- debugpage.php
| | |-- dotask.php
| | |-- dotask.php-old
| | |-- dotask.pl
| | |-- exp
| | |-- gtas.php
| | |-- gtas.phpt
| | |-- gtayearsem.php
| | |-- lispfuns.inc.php
| | |-- messages.php
| | |-- misc-funcs.php
| | |-- restartDaemon.exp
| | |-- sort.inc.php
| | |-- startDaemon.exp
| | |-- stopDaemon.exp
| | |-- style.css
| | |-- syslogin.php
| | |-- test.exp
| | |-- test.php
| |-- manager_include
| | |-- GTA
| | | |-- manager_gta_edit_all_employment_history.inc.php
```

```

| | | |-- manager_gta_edit_employment_history.inc.php
| | | |-- manager_gta_employment_history.inc.php
| | | |-- manager_gta_evaluations.inc.php
| | | |-- manager_gta_evaluations_summary.inc.php
| | | |-- manager_gta_evaluations_summary_new.inc.php
| | | |-- manager_gta_evaluations_view.inc.php
| | | |-- manager_gta_list.inc.php
| | | |-- manager_gta_list_semester.inc.php
| | | |-- manager_gta_list_semester_summary.inc.php
| | | '-- manager_gta_view_prefs.inc.php
| | |-- class
| | | |-- course_menu.inc.php
| | | |-- manager_add_rm_courses_form.inc.php
| | | |-- manager_class_dependencies.inc.php
| | | |-- manager_class_dependencies_edit.inc.php
| | | |-- manager_class_preassign_form.inc.php
| | | |-- manager_edit_class_group.inc.php
| | | |-- manager_edit_courses_form.inc.php
| | | |-- manager_new_class_group.inc.php
| | | '-- manager_view_class_groups.inc.php
| | |-- function_template.inc.php
| | |-- include.php
| | |-- interactive_selections
| | | |-- interactive_selections_class_confinement_form.inc.php
| | | |-- interactive_selections_gta_form.inc.php
| | | '-- interactive_selections_gta_list.inc.php
| | |-- make.pl
| | |-- misc
| | | |-- all_semesters_menu.inc.php
| | | |-- class_type_menu.inc.php
| | | |-- compose_email.inc.php
| | | |-- doHire.inc.php
| | | |-- gta_list_menu.inc.php
| | | |-- hired_sem_menu.inc.php
| | | |-- lispfuncs.inc.php
| | | |-- load_menu.inc.php
| | | |-- print_all_evaluations.inc.php
| | | |-- print_gta.inc.php
| | | |-- print_gta_evals.inc.php
| | | '-- print_prefs.inc.php
| | |-- pdf
| | | |-- class.ezpdf.php
| | | |-- class.pdf.php

```



```

| | | |-- data.txt
| | | |-- fonts
| | | | |-- Courier-Bold.afm
| | | | |-- Courier-BoldOblique.afm
| | | | |-- Courier-Oblique.afm
| | | | |-- Courier.afm
| | | | |-- Helvetica-Bold.afm
| | | | |-- Helvetica-BoldOblique.afm
| | | | |-- Helvetica-Oblique.afm
| | | | |-- Helvetica.afm
| | | | |-- Symbol.afm
| | | | |-- Times-Bold.afm
| | | | |-- Times-BoldItalic.afm
| | | | |-- Times-Italic.afm
| | | | |-- Times-Roman.afm
| | | | |-- ZapfDingbats.afm
| | | | |-- php_Courier-Bold.afm
| | | | |-- php_Courier-BoldOblique.afm
| | | | |-- php_Courier-Oblique.afm
| | | | |-- php_Courier.afm
| | | | |-- php_Helvetica-Bold.afm
| | | | |-- php_Helvetica-BoldOblique.afm
| | | | |-- php_Helvetica-Oblique.afm
| | | | |-- php_Helvetica.afm
| | | | |-- php_Symbol.afm
| | | | |-- php_Times-Bold.afm
| | | | |-- php_Times-BoldItalic.afm
| | | | |-- php_Times-Italic.afm
| | | | |-- php_Times-Roman.afm
| | | | |-- php_ZapfDingbats.afm
| | | | |-- php_a0100131.afm
| | | | '--- php_a0100131.afm
| | | |-- pdf_print.inc.php
| | | |-- print_gta_info.php
| | | |-- readme.pdf
| | | |-- readme.php
| | | '--- ros.jpg
| | |-- reports
| | | |-- class_report.inc.php
| | | '--- gta_report.inc.php
| | '--- search
|-- old
| | |-- class_group_confinement.php

```

```

| | |-- class_group_confinement_choose_semester.php
| | |-- i_selections_class.php -> class_preassign.php
| | '-- i_selections_class_choose_semester.php
| |     -> class_preassign_choose_semester.php
| |-- pdf_files
| |-- phpinfo.php
| |-- print.php
| |-- print_gta_info.php
| |-- search
| | |-- config.pl
| | |-- index.php
| | |-- lisp
| | | |-- era.lisp
| | | |-- heuristic_bt.lisp
| | | |-- local.lisp
| | | '-- rbt.lisp
| | |-- out-no-subcourses
| | |-- out-with-subcourses
| | |-- results.txt
| | |-- search.pl
| | |-- search_era.pl
| | |-- search_heuristic_bt.pl
| | |-- search_local.pl
| | |-- search_query
| | |-- search_rbt.pl
| | '-- semester_config
| |-- search.php
| |-- search_fc-bound.pl
| |-- search_lisp
| | |-- era.lisp
| | |-- local.lisp
| | |-- systematic_dsbt.lisp
| | '-- systematic_rrbt.lisp
| |-- search_test.php
| |-- sel_all.png
| |-- template.php
| '-- x.php
|-- perl_modules
| |-- Exporter
| | '-- Lite.pm
| |-- PHP
| | |-- Session
| | | '-- Serializer

```

```

|   |   |           '-- PHP.pm
|   |   '-- Session.pm
|   |-- Test
|   |   |-- Builder.pm
|   |   |-- Harness
|   |   |   |-- Assert.pm
|   |   |   |-- Iterator.pm
|   |   |   '-- Straps.pm
|   |   |-- Harness.pm
|   |   |-- More.pm
|   |   |-- Simple.pm
|   |   '-- Tutorial.pod
|   |-- UNIVERSAL
|   |   |-- exports.pm
|   |   '-- require.pm
|   |-- auto
|   |   |-- Exporter
|   |   |   '-- Lite
|   |   |-- PHP
|   |   |   '-- Session
|   |   |-- Test
|   |   |   '-- Simple
|   |   '-- UNIVERSAL
|   |       '-- exports
|   '-- sources.tar
|-- phpinfo.php
|-- postlogin.php
|-- prepend.php
|-- print_gta_info.php
|-- scripts
|   |-- anonymizer
|   |   |-- anonymizer.pl
|   |   |-- dist.all.first
|   |   |-- dist.all.last
|   |   |-- dist.female.first
|   |   |-- dist.male.first
|   |   |-- first
|   |   |-- get_first.pl
|   |   |-- get_last.pl
|   |   '-- last
|   |-- config.pl
|   |-- deleteUser.pl
|   |-- fix_gtaName_cases.pl

```

```

| |-- gta_apply.pl
| |-- insertPref.pl
| |-- log_clean.pl
| |-- log_view.cgi
| |-- log_view.pl
| |-- survey_results.pl
| |-- svn-add-unversioned
| |-- svn-clean
| |-- userEmailList.pl
| '-- userList.pl
|-- session_files
'-- template.php

```

C.2 MySQL queries

In this section, we specify all the MySQL queries executed when pushing a button in the system. They are divided into two groups: one for the ‘GTA’ mode and the other for the ‘Classes’ mode. We use the variables denoted by \$semester and \$year to represent the selected semester and year respectively.

C.2.1 Queries posted from the GTA mode

Below are the SQL queries used in the GTA mode:

- In the GTA mode, to show data in the “Select Applied” button, we execute this query:

```

SELECT * FROM users WHERE id IN
(SELECT id FROM gta_apply WHERE c_year='$year'
AND c_semester='$semester' AND apply='true')
ORDER BY lastName;

```

Then, for each user, the academic data information is obtained from the `gta_data` table (each user is denoted by \$id).

```
SELECT support_gta, support_gra, lastmodified FROM gta_data
WHERE id = '$id';
```

For each user, the timestamp for the preference is obtained from the `gta_prefs_ts` table (each user is denoted by `$id`).

```
SELECT lastmodified FROM gta_prefs_ts WHERE id = '$id'
AND c_year = '$year' AND c_semester = '$semester';
```

The hired checkbox state and capacity information is then obtained from the `hired` table.

```
SELECT hired, capacity FROM hired WHERE id = '$id'
AND c_year = '$year' AND c_semester = '$semester';
```

- In the GTA mode, to show data in the “Select Hired” button, we execute this query:

```
SELECT * FROM users WHERE id IN
(SELECT id FROM hired c_year = '$year' AND
c_semester = '$semester' AND hired = 'true')
ORDER BY lastName;
```

Then, for each user, the academic data information is obtained from the `gta_data` table (each user is denoted by `$id`).

```
SELECT support_gta, support_gra, lastmodified FROM gta_data
WHERE id = '$id';
```

For each user, the timestamp for the preference is obtained from the `gta_prefs_ts` table (each user is denoted by `$id`).

```
SELECT lastmodified FROM gta_prefs_ts WHERE id = '$id'
AND c_year = '$year' AND c_semester = '$semester';
```

The hired checkbox state and capacity information is then obtained from the hired table.

```
SELECT hired, capacity FROM hired WHERE id = '$id'
AND c_year = '$year' AND c_semester = '$semester';
```

- In order to filter the GTAs (the “Select Filter” button), we execute this query:

```
SELECT * FROM users WHERE id IN
(SELECT id FROM gta_data, consider_application WHERE
program='$program' AND gta_data.ita='$ita'
AND gta_data.grad_GPA = '$gpa'
AND gta_data.ta_ship = '$ta_ship' AND gta_data.ra_ship = '$ra_ship'
AND gta_data.support_gta = '$support_gta'
AND gta_data.support_gra = '$support_gra'
AND consider_application.c_year = '$year'
AND consider_application.c_semester = '$semester'
AND consider_application.id = gta_data.id) ORDER BY lastName;
```

Then, for each user, the academic data information is obtained from the `gta_data` table (each user is denoted by `$id`).

```
SELECT support_gta, support_gra, lastmodified FROM gta_data
WHERE id = '$id';
```

For each user, the timestamp for the preference is obtained from the `gta_prefs_ts` table (each user is denoted by `$id`).

```
SELECT lastmodified FROM gta_prefs_ts WHERE id = '$id'
AND c_year = '$year' AND c_semester = '$semester';
```

The hired checkbox state and capacity information is then obtained from the hired table.

```
SELECT hired, capacity FROM hired WHERE id = '$id'
AND c_year = '$year' AND c_semester = '$semester';
```

- In the GTA mode, to show data in the “Preference Ranking Applied” button, we execute this query:

```
SELECT * FROM users WHERE id IN
(SELECT id FROM gta_apply WHERE c_year='$year'
AND c_semester='$semester' AND apply='true')
ORDER BY lastName;
```

To get the advisor name, we execute this query (each user is denoted by \$id):

```
SELECT faculty.lastName, faculty.firstName FROM faculty, gta_data
WHERE gta_data.id = '$id' AND gta_data.advisor = faculty.id;
```

To obtain the top preferences (at least a preference value of 4), we execute this query (each user is denoted by \$id):

```
SELECT classId, section FROM classes WHERE id IN
(SELECT classId FROM gta_prefs WHERE id = '$id'
AND c_semester = '$semester' AND c_year = '$year'
AND (preference = 4 OR preference = 5));
```

- In the GTA mode, to show data in the “Preference Ranking Hired” button, we execute this query:

```
SELECT * FROM users WHERE id IN
(SELECT id FROM hired WHERE id = '$id'
AND c_year='$year' AND c_semester='$semester'
AND hired = 'true')
ORDER BY lastName;
```

To get the advisor name, we execute this query (each user is denoted by \$id):

```
SELECT faculty.lastName, faculty.firstName FROM faculty, gta_data
WHERE gta_data.id = '$id' AND gta_data.advisor = faculty.id;
```

To obtain the top preferences (at least a preference value of 4), we execute this query (each user is denoted by \$id):

```
SELECT classId, section FROM classes WHERE id IN
(SELECT classId FROM gta_prefs WHERE id = '$id'
AND c_semester = '$semester' AND c_year = '$year'
AND (preference = 4 OR preference = 5));
```

- In the GTA mode, to show data in the “Pull Preference” button, we execute this query (the user is denoted by \$id):

```
SELECT * FROM gta_prefs WHERE id = '$id';
```

- In the GTA mode, to show data in the “Employment Tasks” button, we execute this query (the user is denoted by \$id):


```

SELECT * FROM users WHERE id IN
(SELECT id FROM hired WHERE id = '$id' AND c_year='$year'
AND c_semester='$semester' AND hired = 'true')
ORDER BY lastName;

```

The hired capacity and salary information is then obtained from the `hired` table (each user is denoted by `$id`).

```

SELECT capacity, salary FROM hired WHERE id = '$id'
AND c_year = '$year' AND c_semester = '$semester';

```

The specific hired information is then obtained from the `hired_info` table (each user is denoted by `$id`).

```

SELECT courseId, task, task_load, instructor FROM hired_info
WHERE id = '$id' AND c_year = '$year' AND c_semester = '$semester';

```

- In the GTA mode, to show data in the “GTA Evaluations” button, we execute this query (the user is denoted by `$id`):

```

SELECT * FROM users WHERE id IN
(SELECT id FROM hired WHERE id = '$id' AND
c_year='$year' AND c_semester='$semester' AND hired = 'true')
ORDER BY lastName;

```

Then, for each user, the academic data information is obtained from the `gta_data` table (each user is denoted by `$id`).

```

SELECT program, admitted_semester, admitted_year, advisor,
grad_gpa, ita, support_gta, support_gra
FROM gta_data WHERE id = '$id'

```

To translate the program id obtained above:

```
SELECT program FROM grad_program WHERE id = '$program'
```

To translate the advisor id obtained above:

```
SELECT lastName, firstName FROM faculty WHERE id = '$faculty'
```

The hired capacity information is then obtained from the hired table.

```
SELECT hired, capacity FROM hired WHERE id = '$id'
AND c_year = '$year' AND c_semester = '$semester';
```

The courses the GTA was assigned to:

```
SELECT courseId, task FROM hired_info WHERE id = '$id'
AND c_semester = '$semester' AND c_year = '$year';
```

The evaluations:

```
SELECT knowledge, communicate, effort, plan, overall, instructor
FROM facEvals WHERE id = '$id' AND courseID = '$courseId'
AND c_semester = '$semester' AND c_year = '$year';
```

The rater:

```
SELECT lastName, firstName FROM faculty WHERE id = '$instructor';
```

- In the GTA mode, to show “All GTAs in system”, we execute this query:

```
SELECT lastName, firstName, id FROM users ORDER BY lastName;
```

For each user (denoted as \$id) to obtain the data on the table, we execute the following queries:

```
SELECT lastmodified FROM gta_data;
```

```
SELECT lastmodified FROM gta_prefs_ts WHERE c_semester = '1'  
AND c_year = '$year' AND id = '$id';
```

```
SELECT apply FROM gta_apply WHERE c_semester = '1'  
AND c_year = '$year' AND id = '$id';
```

```
SELECT lastmodified FROM gta_prefs_ts WHERE c_semester = '2'  
AND c_year = '$year' AND id = '$id';
```

```
SELECT apply FROM gta_apply WHERE c_semester = '2'  
AND c_year = '$year' AND id = '$id';
```

```
SELECT lastmodified FROM gta_prefs_ts WHERE c_semester = '11'  
AND c_year = '$year' AND id = '$id';
```

```
SELECT apply FROM gta_apply WHERE c_semester = '11'  
AND c_year = '$year' AND id = '$id';
```

```
SELECT lastmodified FROM gta_prefs_ts WHERE c_semester = '12'  
AND c_year = '$year' AND id = '$id';
```

```
SELECT apply FROM gta_apply WHERE c_semester = '12'
```

```
AND c_year = '$year' AND id = '$id';
```

```
SELECT lastmodified FROM gta_prefs_ts WHERE c_semester = '13'
AND c_year = '$year' AND id = '$id';
```

```
SELECT apply FROM gta_apply WHERE c_semester = '13'
AND c_year = '$year' AND id = '$id';
```

```
SELECT lastmodified FROM gta_prefs_ts WHERE c_semester = '14'
AND c_year = '$year';
```

```
SELECT apply FROM gta_apply WHERE c_semester = '14'
AND c_year = '$year' AND id = '$id';
```

- In the GTA mode, to show data in the “Employment Record” button, we execute this query:

```
SELECT * FROM users WHERE id = '$id';
```

The hired capacity and salary information is then obtained from the hired table.

```
SELECT capacity, salary FROM hired WHERE id = '$id'
AND c_year = '$year' AND c_semester = '$semester';
```

The specific hired information is then obtained from the hired_info table.

```
SELECT courseId, task, task_load, instructor FROM hired_info
WHERE id = '$id' AND c_year = '$year' AND c_semester = '$semester';
```

- In the GTA mode, to show the data from the “Evaluation Record” button, we execute this query:

```
SELECT instructor, courseID, task, knowledge, communicate,
effort, plan, overall, witness, comments, studentComments,
recommend, date_time FROM facEvals WHERE id = '$id'
AND c_semester = '$semester' AND c_year = '$year';
```

In order to obtain the instructor’s name:

```
SELECT lastName, firstName FROM faculty WHERE id = '$instructor';
```

- In the GTA mode, to show the data from the “Edit Academic Record” button, we execute the following queries:

```
SELECT program, advisor, admitted_year, admitted_semester,
graduation_year, graduation_semester, support_gta,
support_gra, ta_ship_load, ra_ship_load, ugrad_GPA,
grad_GPA, thesis_project_title, research_experience,
extra_experience FROM gta_data WHERE id = '$id';
```

```
SELECT score, percent FROM gta_gre WHERE id = '$id'
AND greType = "VERBAL";
```

```
SELECT score, percent FROM gta_gre WHERE id = '$id'
AND greType = "QUANTITATIVE";
```

```
SELECT score, percent FROM gta_gre WHERE id = '$id'
```

```
AND greType = "ANALYTICAL";
```

```
SELECT score, percent, subjectArea FROM gta_gre  
WHERE id = '$id' AND greType = "SUBJECT";
```

```
SELECT value FROM deficiencies WHERE id IN  
(SELECT deficiency FROM gta_deficiencies WHERE id = '$id');
```

```
SELECT colloquia, ms, phd FROM gta_attendance WHERE id = '$id';
```

```
SELECT done, yr, mo, da FROM exams_comps WHERE id = '$id';
```

```
SELECT a_theory, a_systems, a_applications, a_applications_s,  
yr, mo, da FROM exams_qualifications WHERE id = '$id';
```

```
SELECT course, instructor, duties, a_semester, a_year  
FROM gta_assignments WHERE id = '$id';
```

```
SELECT classId FROM classes WHERE id = '$course';
```

```
SELECT lastName, firstName FROM faculty WHERE id = '$instructor';
```

```
SELECT value FROM duty WHERE id = '$duties';
```

- In order to delete a GTA, this query is performed:

```
UPDATE users SET acl_level = '-3', status = '-3' WHERE id = '$id';
```

C.2.2 Queries posted from the Classes mode

Below are the SQL queries used in the Classes mode:

- In the Classes mode, to show data in the “Setup” button, we execute this query:

```
SELECT canceled, classId, section, name, startTime, endTime,
day_m, day_t, day_w, day_r, day_f, courseLoad, type_lec,
type_grad, type_lab, type_rec FROM classes
WHERE c_semester = '$semester' AND c_year = '$year';
```

- In the Classes mode, to show data in the “Preassignment” button, we execute this query:

```
SELECT gta_id, class_id FROM preassignment WHERE
c_semester = '$semester' AND c_year = '$year';
```

For each \$class_id, we obtain the class number, section, name, and course load:

```
SELECT classId, section, name, courseLoad FROM classes
WHERE id = '$class_id';
```

For each \$gta_id, we obtain the GTA’s name:

```
SELECT lastName, firstName FROM users WHERE id = '$gta_id';
```

- In the Classes mode, to show data in the “Confinement” button, we execute this query:

```
SELECT DISTINCT(confinement) FROM class_confinements
WHERE c_semester = '$semester' AND c_year = '$year';
```

For each \$confinement group, we find out the members of the group.

```
SELECT classId, section FROM classes WHERE id IN
(SELECT class_id FROM class_confinements
WHERE c_semester = '$semester' AND c_year = '$year' AND
confinement = '$confinement');
```

- In the Classes mode, to show data in the “Same-TA” button, we execute this query:

```
SELECT DISTINCT(sameta) FROM same_ta
WHERE c_semester = '$semester' AND c_year = '$year';
```

For each \$sameta group, we find out the members of the group.

```
SELECT classId, section FROM classes WHERE id IN
(SELECT class_id FROM same_ta WHERE c_semester = '$semester'
AND c_year = '$year' AND sameta = '$sameta');
```

- In the Classes mode, to show data in the “Preference ranking” button, we execute this query:

```
SELECT id, classId, section FROM classes
WHERE c_semester = '$semester' AND c_year = '$year';
```

For each \$id, we find GTAs who have preference value 5 or value 4.

```
SELECT lastName, firstName FROM users IN
(SELECT id FROM gta_prefs WHERE classId = '$id' AND preference = '5');
```

```
SELECT lastName, firstName FROM users IN
(SELECT id FROM gta_prefs WHERE classId = '$id' AND preference = '4');
```


Appendix D

Interactive Solver: Java API Documentation

This appendix contains the details of the Java application programming interface (API). Further Java code may be developed that reuses these classes, data structures, and methods. Below is the listing of the Java source code for the Java Interactive Solver.

```
edu/  
'-- unl  
    '-- consystlab  
        |-- CSPConstraint.java  
        |-- CSPPProblem.java  
        |-- CSPUtils.java  
        |-- CSPVVP.java  
        |-- CSPValue.java  
        |-- CSPVariable.java  
        |-- EVector.java  
        |-- FC_Solver.java  
        |-- LabelParams.java  
        |-- Log.java  
        |-- Setup.java  
        |-- Utils.java  
        |-- gtaap
```

```

| |-- Course.java
| |-- GTA.java
| |-- Preference.java
| '-- constraints
|     |-- CapacityConstraint.java
|     |-- CertificationConstraint.java
|     |-- EqualityConstraint.java
|     |-- MutexConstraint.java
|     |-- NilPrefConstraint.java
|     |-- OverlapConstraint.java
|     '-- TakingCourseConstraint.java
|-- gtajava
| |-- GTAAPConfig.java
| |-- Main.java
| |-- RandomGTAAPGenerator.java
| '-- YearSemester.java
|-- gui
| |-- AboutBox.java
| |-- CanEnable.java
| |-- ComboItem.java
| |-- ComboListener.java
| |-- DisplayPreferences.java
| |-- GTAComboBox.java
| |-- GTAComboBoxEditor.java
| |-- GTAComboBoxRenderer.java
| |-- Interactive.java
| |-- JCheckBoxRenderer.java
| |-- JComboBoxEditor.java
| |-- JComboBoxRenderer.java
| |-- OpenFileDialog.java
| |-- PortForwardingL.java
| |-- PrintPage.java
| |-- RAButton.java
| |-- ResourceAssignmentCellEditor.java
| |-- ResourceAssignmentCheckBox.java
| |-- ResourceAssignmentCourseCellEditor.java
| |-- ResourceAssignmentDialog.java
| |-- ResourceAssignments.java
| |-- SaveDialog.java
| |-- SemesterYearSelector.java
| |-- StringRenderer.java
| '-- TunnelAuthenticationDialog.java
|-- jAssignment.java

```

```
|-- java
|-- javac
'-- tests
    |-- AC.java
    |-- BTNode.java
    |-- TestCapacityConstraint.java
    '-- TestCertificationConstraint.java
```

Following are the generated JavaDoc source code documentation for the Java Interactive Solver.

D.1 Package edu.unl.consystlab.gui

Package Contents

Page

Interfaces

CanEnable	102
<i>This is an interface for the JComboBox to show enabled/disabled items in the JComboBox.</i>	

Classes

AboutBox	103
<i>The About box.</i>	
ComboItem	104
<i>This class forms the object that is used in JComboBox so that we have items that are enabled, and items that are disabled.</i>	
ComboListener	106
<i>This class is the combo box listener that handles actions.</i>	
DisplayPreferences	107
<i>This class forms the dialog box for the display preferences.</i>	
GTACombobox	108
<i>This is an extended JComboBox used to display GTAs.</i>	
GTACombobox.horizontalScrollComboUI	111
<i>This method enables horizontal scroll bars on the combobox.</i>	
GTAComboboxEditor	111
<i>This class is the JComboBox editor for GTAs in the resource perspective.</i>	
GTAComboboxRenderer	113

	100
<i>This class is the renderer for the elements for the GTAComboBox.</i>	
Interactive	114
<i>This class is primarily for the display of the Interactive Selections.</i>	
JCheckBoxRenderer	127
<i>This class is the JCheckBox renderer.</i>	
JComboBoxEditor	129
<i>This class is the JCheckBox editor.</i>	
JComboBoxRenderer	130
<i>This class is the table cell renderer for JComboBox.</i>	
OpenDialog	131
<i>This class generates the “Open” dialog box.</i>	
PortForwardingL	133
<i>This class enables and sets up SSH port forwarding for the MySQL access.</i>	
PortForwardingL.MyUserInfo	134
<i>Retrieves the user login credentials.</i>	
PrintPage	135
<i>This class enables the printing of both the resource and task perspective as one.</i>	
RAButton	137
<i>This class is the resource assignment button.</i>	
ResourceAssignmentCellEditor	139
<i>The cell editor for the resource perspective.</i>	
ResourceAssignmentCheckBox	142
<i>This class extends from JCheckBox by adding additional attributed that are kept track of – namely the CSPVariable and ResourceAssignments variables.</i>	
ResourceAssignmentCourseCellEditor	145

<i>This class is the checkbox editor for the resource assignment dialog (ResourceAssignmentDialog) box.</i>	
ResourceAssignmentDialog	147
<i>This class generates the graphical interface for the resource perspective allocation.</i>	
ResourceAssignments	151
<i>This class is used for the resource perspective.</i>	
SaveDialog	154
<i>This class generates the "Save" dialog box.</i>	
SemesterYearSelector	157
<i>This class is the dialog box that prompts the user to select a semester and year.</i>	
StringRenderer	161
<i>This class just is an extended cell renderer.</i>	
TunnelAuthenticationDialog	162
<i>This class prompts the user for their CSE username and password.</i>	

D.2 Interfaces

D.2.1 INTERFACE `CanEnable`

This is an interface for the `JComboBox` to show enabled/disabled items in the `JComboBox`.

D.2.1.1 DECLARATION

```
public interface CanEnable
```

D.2.1.2 METHODS

- *isEnabled*

```
public boolean isEnabled( )
```

- **Usage**

- * If an item is enabled, this method returns `true`, otherwise it is `false`.

- **Returns** - `true` if it is enabled, `false` otherwise.

- *setEnabled*

```
public void setEnabled( boolean isEnabled )
```

- **Usage**

- * Enable/Disable this item.

- **Parameters**

- * `isEnabled` - `true` if this item is enabled, `false` otherwise.

D.3 Classes

D.3.1 CLASS AboutBox

The About box.

D.3.1.1 DECLARATION

```
public class AboutBox
    extends javax.swing.JDialog
    implements java.awt.event.ActionListener
```

D.3.1.2 SERIALIZABLE FIELDS

- private JPanel About
 - This about box.
- private JLabel TitleText
 - The title text.
- private JButton buttonOK
 - The OK button.
- private int width
 - The window height.
- private int height

- The window height.
- private final String OK
 - The constant used for the method `actionPerformed(...)`.
- private JLabel CopyrightInformation
 - Copyright information.

D.3.1.3 CONSTRUCTORS

- *AboutBox*

```
public AboutBox( )
```

 - **Usage**
 - * Constructor.

D.3.1.4 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

D.3.2 CLASS ComboItem

This class forms the object that is used in JComboBox so that we have items that are enabled, and items that are disabled.

D.3.2.1 DECLARATION

```
public class ComboItem
    extends java.lang.Object
    implements CanEnable, java.lang.Comparable
```

D.3.2.2 FIELDS

- public Object obj
 - The object (Course/GTA) treated as the JComboBox item.
- public boolean isEnabled
 - Is the object enabled?

D.3.2.3 METHODS

- *compareTo*

```
public int compareTo( java.lang.Object nCompare )
```

- *hashCode*

```
public int hashCode( )
```

- *isEnabled*

```
public boolean isEnabled( )
```

 - Usage

* If an item is enabled, this method returns `true`, otherwise it is `false`.

– **Returns** - `true` if it is enabled, `false` otherwise.

- *setEnabled*

```
public void setEnabled( boolean isEnabled )
```

– **Usage**

* Enable/Disable this item.

– **Parameters**

* `isEnabled` - `true` if this item is enabled, `false` otherwise.

- *toString*

```
public String toString( )
```

D.3.3 CLASS ComboListener

This class is the combo box listener that handles actions.

D.3.3.1 DECLARATION

```
public class ComboListener
    extends java.lang.Object
    implements java.awt.event.ActionListener
```

D.3.3.2 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

D.3.4 CLASS DisplayPreferences

This class forms the dialog box for the display preferences. Here we allow turning on/off table columns such the section name, number, etc.

D.3.4.1 DECLARATION

```
public class DisplayPreferences
    extends java.lang.Object
    implements java.awt.event.ActionListener
```

D.3.4.2 CONSTRUCTORS

- *DisplayPreferences*

```
public DisplayPreferences( edu.unl.consyslab.gui.Interactive
    parent )
```

- Usage

- * Constructor.

- Parameters

- * parent - Our parent window, Interactive.

D.3.4.3 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

- *setVisible*

```
public void setVisible( boolean b )
```

- **Usage**

- * Shows or hides the display preferences.

- **Parameters**

- * **b** - true to set as visible, false otherwise.

D.3.5 CLASS GTAComboBox

This is an extended JComboBox used to display GTAs.

D.3.5.1 DECLARATION

```
public class GTAComboBox
    extends javax.swing.JComboBox
```

D.3.5.2 SERIALIZABLE FIELDS

- private int filteredOutIndex

- The position of the filtered-out item separator.

D.3.5.3 CONSTRUCTORS

- *GTAComboBox*

```
public GTAComboBox( )
```

- **Usage**

- * Constructor.

- *GTAComboBox*

```
public GTAComboBox( javax.swing.JComboBox jcb )
```

- **Usage**

- * Constructor.

- **Parameters**

- * *jcb* - Items from this is copied over to our new GTAComboBox.

D.3.5.4 METHODS

- *getComboBox*

```
public JComboBox getComboBox( )
```

- **Usage**

- * Returns the JComboBox representation of myself.

- **Returns** - The JComboBox representation of myself.
-

- *getFilteredOutIndex*

```
public int getFilteredOutIndex( )
```

- **Usage**

- * This method returns the filtered out index location.

- **Returns** - The index.

- *getVariable*

```
public CSPVariable getVariable( )
```

- **Usage**

- * Gets the CSPVariable for this combo box.

- **Returns** - Returns the variable.

- *setFilteredOutIndex*

```
public void setFilteredOutIndex( int index )
```

- **Usage**

- * This method sets the index for the "– Filtered Out –" separator so the renderer will know where to change the color.

- **Parameters**

- * index -

- *setVariable*

```
public void setVariable( edu.unl.consyslab.CSPVariable variable  
)
```

- **Usage**

* Sets the `CSPVariable` for this combo box.

– **Parameters**

* `variable` - The variable to set.

D.3.6 CLASS `GTAComboBox.horizontalScrollComboUI`

This method enables horizontal scroll bars on the combobox.

D.3.6.1 DECLARATION

```
public class GTAComboBox.horizontalScrollComboUI
    extends javax.swing.plaf.basic.BasicComboBoxUI
```

D.3.6.2 CONSTRUCTORS

- *GTAComboBox.horizontalScrollComboUI*

```
public GTAComboBox.horizontalScrollComboUI( )
```

D.3.6.3 METHODS

- *createPopup*

```
protected ComboPopup createPopup( )
```

D.3.7 CLASS `GTAComboBoxEditor`

This class is the `JComboBox` editor for GTAs in the resource perspective.

D.3.7.1 DECLARATION

```
public class GTAComboBoxEditor
    extends javax.swing.AbstractCellEditor
    implements javax.swing.table.TableCellEditor, java.awt.event.ActionListener
```

D.3.7.2 SERIALIZABLE FIELDS

- private `GTAComboBox combobox`
 - The internal combobox used by this class.

D.3.7.3 CONSTRUCTORS

- *GTAComboBoxEditor*

```
public GTAComboBoxEditor( edu.unl.consyslab.gui.GTAComboBox
    gcb )
```

 - **Usage**
 - * Default constructor.
 - **Parameters**
 - * `gcb` - Sets the internal combo box value to this.

D.3.7.4 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

- *getCellEditorValue*

```
public Object getCellEditorValue( )
```

- *getTableCellEditorComponent*

```
public Component getTableCellEditorComponent(
    javax.swing.JTable table, java.lang.Object value, boolean isSelected,
    int row, int column )
```

D.3.8 CLASS GTAComboBoxRenderer

This class is the renderer for the elements for the GTAComboBox.

D.3.8.1 DECLARATION

```
public class GTAComboBoxRenderer
    extends javax.swing.JLabel
    implements javax.swing.ListCellRenderer
```

D.3.8.2 SERIALIZABLE FIELDS

- private GTAComboBox cb
 - The internal combobox used by this class.

D.3.8.3 CONSTRUCTORS

- *GTAComboBoxRenderer*

```
public GTAComboBoxRenderer( )
```

– Usage

* Constructor.

- *GTAComboBoxRenderer*

```
public GTAComboBoxRenderer( edu.unl.consystlab.gui.GTAComboBox
gcb )
```

– Usage

* Constructor.

– Parameters

* *gcb* - Sets the internal combo box value to this.

D.3.8.4 METHODS

- *getListCellRendererComponent*

```
public Component getListCellRendererComponent(
javax.swing.JList list, java.lang.Object value, int index,
boolean isSelected, boolean cellHasFocus )
```

D.3.9 CLASS Interactive

This class is primarily for the display of the Interactive Selections.

D.3.9.1 DECLARATION

```
public class Interactive
{
    extends javax.swing.JFrame
    implements java.awt.event.ActionListener,
    javax.swing.event.TableModelListener
}
```

D.3.9.2 SERIALIZABLE FIELDS

- private boolean fromDatabase
 - If the problem data is loaded from the database, this is set to `true`. Otherwise, it is set to `false`
- private JPanel jContentPane
 - The `JPanel` for the content pane.
- private JMenuBar jJMenuBar
 - The `JMenuBar` that is displayed for the file, edit, and help menu.
- private JMenu fileMenu
 - The `JMenu` for the file menu.
- private JMenu editMenu
 - The `JMenu` for the edit menu.
- private JMenu helpMenu
 - The `JMenu` for the help menu.

- private JMenuItem exitMenuItem
 - The JMenuItem to exit/quit the program.
- private JMenuItem aboutMenuItem
 - The JMenuItem to call the about box.
- private JMenuItem displayPreferenceMenuItem
 - The JMenuItem to display and set preferences.
- private DisplayPreferences displayPreferences
 - The DisplayPreferences object for setting the preferences
- private JMenuItem openMenuItem
 - The JMenuItem to call the open box.
- private JMenuItem saveMenuItem
 - The JMenuItem to call the save box.
- private JMenuItem refetchDataMenuItem
 - The JMenuItem to call the refetch-data box.
- private JMenuItem printMenuItem
 - The JMenuItem to print.
- private JScrollPane taskPerspectiveScroller
 - The JScrollPane for the task perspective.
- private JScrollPane taskTallyScroller
 - The JScrollPane for the task tally.

- private JScrollPane resourcePerspectiveScroller
 - The JScrollPane for the resource perspective.
- private JScrollPane resourceTallyScroller
 - The JScrollPane for the resource tally.
- private JTable taskPerspective
 - The JTable for the task perspective.
- private JPanel taskTally
 - The JPanel for the task tally.
- private JTable resourcePerspective
 - The JTable for the resource perspective.
- private JPanel colorLegend
 - The JPanel for the color legend.
- private CSPPProblem problem
 - The CSPPProblem for this problem.
- private boolean takeNoAction
 - If set to `true`, no action is taken when a table change happens.
- private boolean doAssign
 - If set to `true`, then assignments are made, otherwise nothing happens.
- private boolean hasPreassignments

- When there are preassignments (preassignments are loaded and there is at least one preassignment) from the database, this is set to `true`, otherwise `false`.
- private double totalNeeded
 - The tally for the total needed capacity from the GTAs.
- private double totalAssigned
 - The total assigned load made so far.
- private double totalPreassignedLoad
 - The total preassigned load.
- private int totalPreassigned
 - The total number of courses preassigned.
- private YearSemester ys
 - This is used to store the selected year and semester.
- private boolean firstTime
 - If this is the first time the Interactive window is displayed, it is `true`.
- private boolean doRefetch
 - If we want to refetch the data, then this is `true`.
- private int width
 - The width of this Interactive window. This is used to determine the width of the window and set the position on screen.
- private int height

- The height of this `Interactive` window. This is used to determine the height of the window and set the position on screen.
- `private Interactive self`
 - Just a reference to this `Interactive` object so we can pass it to other classes.

D.3.9.3 CONSTRUCTORS

- *Interactive*

```
public Interactive( )
```

 - **Usage**
 - * This is the default constructor

D.3.9.4 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

 - **See Also**
 - * `java.awt.event.ActionListener.actionPerformed(java.awt.event.ActionEvent)`

- *Assign*

```
public void Assign( edu.unl.consyslab.CSPVariable c,
edu.unl.consyslab.CSPValue g )
```


– **Usage**

- * Makes an assignment of a value (GTA) to a variable (course). This method first does an unassignment before making an assignment to ensure that an already assigned course is not assigned twice and propagation is done properly.

– **Parameters**

- * `c` - The variable to assign with value `g`.
- * `g` - The value to assign to variable `c`.

• *calculateTotalAssigned*

```
public double calculateTotalAssigned( )
```

– **Usage**

- * Calculates the total assigned courses load.

– **Returns** - The double value for the total assigned courses load.

• *getCourse*

```
public Course getCourse( edu.unl.consyslab.CSPVariable var )
```

– **Usage**

- * Returns the `Course` object from a `CSPVariable`.

– **Parameters**

- * `var` - The `CSPVariable`.

– **Returns** - The `Course`.

- *getGTA*

```
public GTA getGTA( edu.unl.consyslab.CSPValue val )
```

- **Usage**

- * Returns the GTA object from a CSPValue.

- **Parameters**

- * val - The CSPValue.

- **Returns** - The GTA.

- *getResourcePerspective*

```
public JTable getResourcePerspective( )
```

- **Usage**

- * This method returns resourcePerspective. If it was not previously initialized, it is initialized, then returned.

- **Returns** - javax.swing.JTable The initialized resourcePerspective

- *getTaskPerspective*

```
public JTable getTaskPerspective( )
```

- **Usage**

- * This method returns taskPerspective. If it was not previously initialized, it is initialized, then returned.

- **Returns** - javax.swing.JTable The initialized taskPerspective

- *getTotalAssigned*

```
public double getTotalAssigned( )
```

– **Usage**

* Returns the total assigned courses load by calling
`calculateTotalAssigned()`

– **Returns** - The double value for the total assigned courses load.

– **See Also**

* `edu.unl.consyslab.gui.Interactive.calculateTotalAssigned()`

• *getTotalNeeded*

`public double getTotalNeeded()`

– **Usage**

* Returns the totalNeeded.

– **Returns** - Returns the totalNeeded.

• *getTotalPreassigned*

`public int getTotalPreassigned()`

– **Usage**

* Returns the totalPreassigned.

– **Returns** - Returns the totalPreassigned.

• *getTotalPreassignedLoad*

`public double getTotalPreassignedLoad()`

– **Usage**

* Returns the totalPreassignedLoad.

– **Returns** - Returns the totalPreassignedLoad.

- *main*

```
public static void main( java.lang.String [] args )
```

- **Usage**

- * Launches this application
-

- *restoreAssignments*

```
public void restoreAssignments( java.util.Vector save )
```

- **Usage**

- * Restores an assignment from the save Vector. The save Vector must contain elements of type jAssignment.

- **Parameters**

- * save - Vector of elements jAssignment
-

- *restoreAssignmentsFrom*

```
public void restoreAssignmentsFrom( java.lang.String saved-  
Name )
```

- **Usage**

- * Restores an assignment from the saveName parameter.

- **Parameters**

- * savedName - String containing the name of the saved assignments to restore from.
-

- *saveAssignments*

```
public void saveAssignments( edu.unl.consyslab.gui.SaveDialog  
sd )
```

- **Usage**

- * Saves the current assignments to the database. The name to save as is obtained from the SaveDialog parameter.

- **Parameters**

- * sd -
-

- *setDoRefetch*

```
public void setDoRefetch( boolean doRefetch )
```

- **Usage**

- * Sets the boolean value doRefetch.

- **Parameters**

- * doRefetch - The boolean value to be set for the class variable doRefetch.
-

- *setTotalAssigned*

```
public void setTotalAssigned( double totalAssigned )
```

- **Usage**

- * Sets the total assigned value.

- **Parameters**

- * totalAssigned - The totalAssigned to set.
-

- *setTotalNeeded*

```
public void setTotalNeeded( double totalNeeded )
```

- **Usage**

- * The totalNeeded to set.

- **Parameters**

- * totalNeeded - The totalNeeded to set.

- *setTotalPreassigned*

```
public void setTotalPreassigned( int totalPreassigned )
```

- **Usage**

- * The totalPreassigned to set.

- **Parameters**

- * totalPreassigned - The totalPreassigned to set.

- *setTotalPreassignedLoad*

```
public void setTotalPreassignedLoad( double totalPreassigned-  
Load )
```

- **Usage**

- * The totalPreassignedLoad to set.

- **Parameters**

- * totalPreassignedLoad - The totalPreassignedLoad to set.

- *tableChanged*

```
public void tableChanged( javax.swing.event.TableModelEvent  
e )
```

– See Also

* `javax.swing.event.TableModelListener.tableChanged(
 javax.swing.event.TableModelEvent)`

• *Unassign*

`public void Unassign(edu.unl.consyslab.CSPVariable c)`

– Usage

* Makes an unassignment of a variable `c`.

– Parameters

* `c` - The variable to unassign.

• *updateResourcePerspective*

`public void updateResourcePerspective()`

– Usage

* This method will update the GUI resource perspective to reflect the assignment.

• *updateTaskPerspective*

`public void updateTaskPerspective()`

– Usage

* This method will update the GUI task perspective to reflect the assignment.

• *updateTaskTally_totalAssigned*

`public void updateTaskTally_totalAssigned()`

– Usage

- * Updates the task tally by calculating the total assigned load.

- *updateTaskTally_totalAssigned*

```
public void updateTaskTally_totalAssigned( double total )
```

– Usage

- * Updates the task tally by using the parameter total.

– Parameters

- * `total` - The value we update the task tally with.

D.3.10 CLASS JCheckBoxRenderer

This class is the JCheckBox renderer.

D.3.10.1 DECLARATION

```
public class JCheckBoxRenderer
    extends javax.swing.JCheckBox
    implements javax.swing.table.TableCellRenderer
```

D.3.10.2 SERIALIZABLE FIELDS

- private TableSorter sorter
 - The table sorter.

D.3.10.3 CONSTRUCTORS

- *JCheckBoxRenderer*

```
public JCheckBoxRenderer( )
```

- **Usage**

- * Constructor.

- *JCheckBoxRenderer*

```
public JCheckBoxRenderer( javax.swing.JCheckBox cb )
```

- **Usage**

- * Constructor.

- **Parameters**

- * `cb` - This is copied to this class' checkbox value.

- *JCheckBoxRenderer*

```
public JCheckBoxRenderer( thirdparty.TableSorter sorter )
```

- **Usage**

- * Constructor.

- **Parameters**

- * `sorter` - The local sorter is set to this value.

D.3.10.4 METHODS

- *getTableCellRendererComponent*

```
public Component getTableCellRendererComponent(
    javax.swing.JTable table, java.lang.Object value, boolean isSelected,
    boolean hasFocus, int row, int column )
```

D.3.11 CLASS JComboBoxEditor

This class is the JCheckBox editor.

D.3.11.1 DECLARATION

```
public class JComboBoxEditor
    extends javax.swing.DefaultCellEditor
```

D.3.11.2 CONSTRUCTORS

- *JComboBoxEditor*

```
public JComboBoxEditor( edu.unl.consyslab.gui.GTAComboBox
    gcb )
```

– Usage

* Constructor.

– Parameters

* *gcb* - The GTAComboBox that the super class is instantiated with.

D.3.12 CLASS JComboBoxRenderer

This class is the table cell renderer for JComboBox.

D.3.12.1 DECLARATION

```
public class JComboBoxRenderer
    extends edu.unl.consyslab.gui.GTAComboBox
    implements javax.swing.table.TableCellRenderer
```

D.3.12.2 CONSTRUCTORS

- *JComboBoxRenderer*

```
public JComboBoxRenderer( )
```

– Usage

* Constructor.

- *JComboBoxRenderer*

```
public JComboBoxRenderer( edu.unl.consyslab.gui.GTAComboBox
    cb )
```

– Usage

* Constructor.

– Parameters

* *cb* - The *GTAComboBox* that the super class' is initialized to.

D.3.12.3 METHODS

- *getTableCellRendererComponent*

```
public Component getTableCellRendererComponent(
    javax.swing.JTable table, java.lang.Object valueo, boolean isS-
    elected, boolean hasFocus, int row, int column )
```

D.3.13 CLASS OpenFileDialog

This class generates the “Open” dialog box.

D.3.13.1 DECLARATION

```
public class OpenFileDialog
    extends javax.swing.JDialog
    implements java.awt.event.ActionListener
```

D.3.13.2 SERIALIZABLE FIELDS

- private JPanel jContentPane
 - The content pane.
- private JComboBox openAsName
 - The JComboBox where we can select previous saved assignments.
- private JButton okButton

- The OK button.
- private JButton cancelButton
 - The Cancel button.
- private JLabel openLabel
 - The label for “Open”
- private Interactive parent
 - The parent window.
- private JLabel openWarningLabel
 - Warning label.

D.3.13.3 CONSTRUCTORS

- *OpenDialog*

```
public OpenDialog( edu.unl.consyslab.gui.Interactive parent
)
```

 - **Usage**
 - * Constructor.
 - **Parameters**
 - * **parent** - The parent window.

D.3.13.4 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

D.3.14 CLASS PortForwardingL

This class enables and sets up SSH port forwarding for the MySQL access.

D.3.14.1 DECLARATION

```
public class PortForwardingL
    extends java.lang.Object
```

D.3.14.2 CONSTRUCTORS

- *PortForwardingL*

```
public PortForwardingL( )
```

– Usage

* Constructor.

D.3.14.3 METHODS

- *close*

```
public void close( )
```

– Usage

* Close the port.

- *open*

```
public void open( )
```

- **Usage**

* Open up the port.

- **Exceptions**

* `com.jcraft.jsch.JSchException` - This happens when the port is currently unavailable either because it is being used or a user tries to open a privileged port (less than 1024).

D.3.15 CLASS `PortForwardingL.MyUserInfo`

Retrieves the user login credentials.

D.3.15.1 DECLARATION

```
public static class PortForwardingL.MyUserInfo
    extends java.lang.Object
    implements com.jcraft.jsch.UserInfo
```

D.3.15.2 CONSTRUCTORS

- *PortForwardingL.MyUserInfo*

```
public PortForwardingL.MyUserInfo( )
```

D.3.15.3 METHODS

- *getPassphrase*

```
public String getPassphrase( )
```

- *getPassword*

```
public String getPassword( )
```

- *promptPassphrase*

```
public boolean promptPassphrase( java.lang.String message )
```

- *promptPassword*

```
public boolean promptPassword( java.lang.String message )
```

- *promptYesNo*

```
public boolean promptYesNo( java.lang.String message )
```

- *setPassword*

```
public void setPassword( java.lang.String password )
```

- *showMessage*

```
public void showMessage( java.lang.String message )
```

D.3.16 CLASS **PrintPage**

This class enables the printing of both the resource and task perspective as one. Since we cannot print both perspectives together, we have to reconstruct a new table with both perspectives on it.

D.3.16.1 DECLARATION

```
public class PrintPage
    extends java.lang.Object
    implements java.awt.event.ActionListener, java.awt.print.Printable
```

D.3.16.2 CONSTRUCTORS

- *PrintPage*

```
public PrintPage( edu.unl.consyslab.gui.Interactive parent
    )
```

- **Usage**

- * Constructor.

- **Parameters**

- * **parent** - The parent window.

D.3.16.3 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

- *getPreviewTable*

```
public JTable getPreviewTable( )
```

- **Usage**

- * Returns the preview table (the reconstructed, merged table).

- **Returns** - The preview table.

- *print*

```
public int print( java.awt.Graphics g, java.awt.print.PageFormat
pf, int pi )
```

- **Usage**

- * This print method will print the entire table out and scale it to fit the paper.

- **Parameters**

- * *g* - The graphics.

- * *pf* - The page format.

- * *pi* - The page index.

D.3.17 CLASS RAButton

This class is the resource assignment button.

D.3.17.1 DECLARATION

```
public class RAButton
extends javax.swing.JButton
```

D.3.17.2 SERIALIZABLE FIELDS

- private `ResourceAssignments` rassignment
 - The `ResourceAssignments` for this button.

D.3.17.3 CONSTRUCTORS

- *RAButton*
`public RAButton()`

- Usage
 - * Constructor.
-

- *RAButton*
`public RAButton(edu.unl.consyslab.gui.ResourceAssignments
ra, java.lang.String label)`

- Usage
 - * COnstructor.
- Parameters
 - * ra - The `ResourceAssignments` to set this to.
 - * label - The label for this button.

D.3.17.4 METHODS

- *getRassignment*

```
public ResourceAssignments getRassignment( )
```

- **Usage**

- * Gets the ResourceAssignments for this class.

- **Returns** - The ResourceAssignments for this class.

- *setRassignment*

```
public void setRassignment(
edu.unl.consyslab.gui.ResourceAssignments rassignment )
```

- **Usage**

- * Sets the ResourceAssignments for this class.

- **Parameters**

- * rassignment - The ResourceAssignments for this class.

D.3.18 CLASS ResourceAssignmentCellEditor

The cell editor for the resource perspective. This is displayed on the main Interactivewindow and when the user clicks on this cell, a dialog box will open up so he/she can perform assignments.

D.3.18.1 DECLARATION

```
public class ResourceAssignmentCellEditor
extends javax.swing.AbstractCellEditor
implements javax.swing.table.TableCellEditor, java.awt.event.ActionListener
```

D.3.18.2 SERIALIZABLE FIELDS

- private Interactive parent
 - The parent window.
- private CSPPProblem problem
 - The problem.

D.3.18.3 CONSTRUCTORS

- *ResourceAssignmentCellEditor*

```
public ResourceAssignmentCellEditor(
    edu.unl.consyslab.gui.Interactive parent )
```

- Usage

- * Constructor.

- Parameters

- * parent - The parent window.

- *ResourceAssignmentCellEditor*

```
public ResourceAssignmentCellEditor(
    edu.unl.consyslab.gui.Interactive parent,
    edu.unl.consyslab.gui.ResourceAssignments ra,
    edu.unl.consyslab.CSPPProblem problem )
```

- Usage

* Constructor.

– **Parameters**

* `parent` - The parent window.

* `ra` - The assignment data.

* `problem` - The problem.

• *ResourceAssignmentCellEditor*

```
public ResourceAssignmentCellEditor(
    edu.unl.consyslab.gui.Interactive parent,
    edu.unl.consyslab.gui.ResourceAssignments
    ra, edu.unl.consyslab.CSPPProblem problem,
    edu.unl.consyslab.gtaap.GTA gta )
```

– **Usage**

* Constructor.

– **Parameters**

* `parent` - The parent window.

* `ra` - The assignment data.

* `problem` - The problem.

* `gta` - The GTA for these assignments.

D.3.18.4 METHODS

• *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

- *getCellEditorValue*

```
public Object getCellEditorValue( )
```

- *getTableCellEditorComponent*

```
public Component getTableCellEditorComponent(
    javax.swing.JTable table, java.lang.Object value, boolean isSe-
    lected, int row, int column )
```

D.3.19 CLASS ResourceAssignmentCheckBox

This class extends from `JCheckBox` by adding additional attributed that are kept track of – namely the `CSPVariable` and `ResourceAssignments` variables.

D.3.19.1 DECLARATION

```
public class ResourceAssignmentCheckBox
    extends javax.swing.JCheckBox
```

D.3.19.2 SERIALIZABLE FIELDS

- `private CSPVariable variable`
 - The variable.
- `private ResourceAssignments rassignment`
 - The assignment data.

D.3.19.3 CONSTRUCTORS

- *ResourceAssignmentCheckBox*

```
public ResourceAssignmentCheckBox( )
```

- Usage

- * Constructor.

- *ResourceAssignmentCheckBox*

```
public ResourceAssignmentCheckBox( edu.unl.consyslab.CSPVariable
variable, edu.unl.consyslab.gui.ResourceAssignments rassign-
ment, boolean bool )
```

- Usage

- * Constructor.

- Parameters

- * `variable` - The variable.

- * `rassignment` - The assignment data.

- * `bool` - If the checkbox is checked, this is `true`; `false` otherwise.

- *ResourceAssignmentCheckBox*

```
public ResourceAssignmentCheckBox( java.lang.String
label, edu.unl.consyslab.CSPVariable variable,
edu.unl.consyslab.gui.ResourceAssignments rassignment, boolean
bool )
```

- Usage

* Constructor.

– **Parameters**

* `label` - The label associated with this checkbox.

* `variable` - The variable.

* `rassignment` - The assignment data.

* `bool` - If the checkbox is checked, this is `true`; `false` otherwise.

D.3.19.4 METHODS

- *getRassignment*

```
public ResourceAssignments getRassignment( )
```

– **Usage**

* Gets the assignment data.

– **Returns** - The assignment data.

- *getVariable*

```
public CSPVariable getVariable( )
```

– **Usage**

* Gets the variable.

– **Returns** - The variable.

- *setRassignment*

```
public void setRassignment(
edu.unl.consyslab.gui.ResourceAssignments rassignment )
```

– **Usage**

* Sets the assignment data.

– **Parameters**

* `rassignment` - The assignment data.

• *setVariable*

```
public void setVariable( edu.unl.consyslab.CSPVariable variable
)
```

– **Usage**

* Sets the variable.

– **Parameters**

* `variable` - The variable.

D.3.20 CLASS ResourceAssignmentCourseCellEditor

This class is the checkbox editor for the resource assignment dialog (`ResourceAssignmentDialog`) box.

D.3.20.1 DECLARATION

```
public class ResourceAssignmentCourseCellEditor
    extends javax.swing.AbstractCellEditor
    implements javax.swing.table.TableCellEditor, java.awt.event.ActionListener
```

D.3.20.2 SERIALIZABLE FIELDS

- private JCheckBox checkbox
 - The checkbox for editing.

D.3.20.3 CONSTRUCTORS

- *ResourceAssignmentCourseCellEditor*

```
public ResourceAssignmentCourseCellEditor(
    javax.swing.JCheckBox cb, boolean assignBool )
```

- Usage

- * Constructor.

- Parameters

- * `cb` - The checkbox that we set as our internal checkbox attribute.

- * `assignBool` - If this is true then the internal checkbox' state is checked; otherwise it is unchecked.

-
- *ResourceAssignmentCourseCellEditor*

```
public ResourceAssignmentCourseCellEditor(
    edu.unl.consyslab.gui.ResourceAssignmentDialog
    rad, javax.swing.JCheckBox cb, boolean as-
    signBool, edu.unl.consyslab.CSPProblem prob,
    edu.unl.consyslab.CSPVariable var )
```

- Parameters

- * `rad` - The ResourceAssignmentDialog that holds this cell editor.

- * `cb` - The checkbox that we set as our internal checkbox attribute.

- * `assignBool` - If this is `true` then the internal checkbox' state is checked; otherwise it is unchecked.
- * `prob` - The problem.
- * `var` - The variable.

D.3.20.4 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

- *getCellEditorValue*

```
public Object getCellEditorValue( )
```

- *getTableCellEditorComponent*

```
public Component getTableCellEditorComponent(
    javax.swing.JTable table, java.lang.Object value, boolean isSelected,
    int row, int column )
```

D.3.21 CLASS ResourceAssignmentDialog

This class generates the graphical interface for the resource perspective allocation.

D.3.21.1 DECLARATION

```
public class ResourceAssignmentDialog
    extends javax.swing.JDialog
    implements java.awt.event.ActionListener, javax.swing.event.TableModelListener
```

D.3.21.2 SERIALIZABLE FIELDS

- private JScrollPane courseTableScroller
 -
- private JTable courseTable
 - The course table.
- private Interactive parent
 - The parent window of this dialog box.
- private CSPValue value
 - The value this dialog box is for.
- private GTA gta
 - The GTA this dialog box is for.
- private ResourceAssignments rassignment
 - The `ResourceAssignments` attribute for this class.
- private JPanel jContentPane
 - The content pane.
- private CSPProblem problem
 - The problem.
- private boolean takeNoAction

– If this is set to `true`, then no action will be taken by the `actionPerformed(...)` method.

- `private int width`
 - The window's width.
- `private int height`
 - The window's height.

D.3.21.3 CONSTRUCTORS

- *ResourceAssignmentDialog*

```
public ResourceAssignmentDialog( edu.unl.consyslab.gui.Interactive
parent, edu.unl.consyslab.CSPProblem problem,
edu.unl.consyslab.gui.ResourceAssignments ra )
```

– **Usage**

* Constructor.

– **Parameters**

* `parent` - The parent window.

* `problem` - The problem.

* `ra` - The assignments made so far.

D.3.21.4 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

- *getAllReversedDomain*

```
public Vector getAllReversedDomain( )
```

- **Usage**

- * Returns the reversed domain for the GTA (i.e., the available courses for the GTA in this class) based on the savedDomain Vector attribute.

- **Returns** - Vector of CSPVariables that are available for this GTA.

- *getCourse*

```
public Course getCourse( edu.unl.consyslab.CSPVariable var )
```

- **Usage**

- * Quick method to obtain the Course from a CSPVariable.

- **Parameters**

- * val - The CSPVariable which we want the Course of.

- **Returns** - Returns the Course value.

- *getGTA*

```
public GTA getGTA( edu.unl.consyslab.CSPValue val )
```

- **Usage**

- * Quick method to obtain the GTA from a CSPValue.

- **Parameters**

- * val - The CSPValue which we want the GTA of.

- **Returns** - Returns the GTA value.

-
- *getReversedDomain*

```
public Vector getReversedDomain( )
```

- **Usage**

- * Returns the reversed domain for the GTA (i.e., the available courses for the GTA in this class) based on the currentDomain Vector attribute.

- **Returns** - Vector of CSPVariables that are available for this GTA.

- *refreshCourseTable*

```
public void refreshCourseTable( )
```

- **Usage**

- * Refreshes the resource perspective for the GTA in this dialog box.

- *tableChanged*

```
public void tableChanged( javax.swing.event.TableModelEvent  
e )
```

D.3.22 CLASS ResourceAssignments

This class is used for the resource perspective. Each CSPValue (GTA) may be assigned to multiple courses. This class keeps track of the multiple assignments.

D.3.22.1 DECLARATION

<pre>public class ResourceAssignments extends java.lang.Object</pre>

D.3.22.2 CONSTRUCTORS

- *ResourceAssignments*

```
public ResourceAssignments( edu.unl.consyslab.CSPValue value
)
```

- Usage

- * Constructor.

- Parameters

- * value - Sets the internal value to this.

D.3.22.3 METHODS

- *addAssignments*

```
public void addAssignments( edu.unl.consyslab.CSPVariable
variable )
```

- Usage

- * Adds a variable to the assignments vector.

- Parameters

- * variable - Adds this variable to the assignments vector. If the variable already exists, it will not re-add it.

- *getAssignments*

```
public Vector getAssignments( )
```

- **Usage**
 - * Returns the assignments.
 - **Returns** - The assignments.
-

- *getValue*

```
public CSPValue getValue( )
```

- **Usage**
 - * Returns the value.
 - **Returns** - The value.
-

- *setAssignments*

```
public void setAssignments( java.util.Vector assignments )
```

- **Usage**
 - * Sets the assignments from the `Vector` assignments.
 - **Parameters**
 - * `assignments` - The assignments.
-

- *setValue*

```
public void setValue( edu.unl.consyslab.CSPValue value )
```

- **Usage**
 - * Sets the value from the `CSPValue` value.
- **Parameters**
 - * `value` - The value.

-
- *toString*

```
public String toString( )
```

- *toString*

```
public String toString( boolean shortFormat )
```

- **Usage**

- * This method returns the string representation of this object.

- **Parameters**

- * **shortFormat** - If this is **true**, then we return the string representation without the course prefix (e.g., CSCE, MATH). Otherwise, the string representation includes it.

- **Returns** - The string representation of this object.

D.3.23 CLASS SaveDialog

This class generates the "Save" dialog box.

D.3.23.1 DECLARATION

```
public class SaveDialog
    extends javax.swing.JDialog
    implements java.awt.event.ActionListener
```

D.3.23.2 SERIALIZABLE FIELDS

- `private JPanel jContentPane`
 - The content pane.
- `private JComboBox saveAsName`
 - The `JComboBox` where we can select previous saved assignments or edit/create new ones.
- `private JButton saveButton`
 - The save button.
- `private JButton cancelButton`
 - The cancel button.
- `private JLabel saveAsLabel`
 - The "Save as" label.
- `private JButton deleteSavedButton`
 - The delete button.
- `private String action`
 - The action.

D.3.23.3 FIELDS

- `public static final String CANCEL`

- The constant used for the method `actionPerformed(...)`.
- `public static final String DELETE`
 - The constant used for the method `actionPerformed(...)`.
- `public static final String SAVE`
 - The constant used for the method `actionPerformed(...)`.

D.3.23.4 CONSTRUCTORS

- *SaveDialog*

```
public SaveDialog( edu.unl.consyslab.gui.Interactive parent
)
```

 - **Usage**
 - * Constructor.
 - **Parameters**
 - * `parent` - The parent window.

D.3.23.5 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

- *getAction*

```
public String getAction( )
```

– **Returns** - Returns the action.

- *getSavedName*

```
public String getSavedName( )
```

– **Usage**

* Returns the saved name from the dialog pulldown.

– **Returns** - The String value of the name we want to save as.

- *setAction*

```
public void setAction( java.lang.String action )
```

– **Parameters**

* `action` - The action to set.

D.3.24 CLASS SemesterYearSelector

This class is the dialog box that prompts the user to select a semester and year. The semester and year options are already in the database (in the “classes” table).

D.3.24.1 DECLARATION

```
public class SemesterYearSelector
    extends javax.swing.JDialog
    implements java.awt.event.ActionListener
```

D.3.24.2 SERIALIZABLE FIELDS

- `private JPanel jContentPane`
 - The content pane.
- `private JLabel textLabel`
 - The label "Please choose semester"
- `private JComboBox semesters`
 - The pull down menu with the years and semesters.
- `private JPanel jPanel`
 - The panel.
- `private JButton buttonOk`
 - The OK button.
- `private JButton buttonQuit`
 - The quit button.
- `private JButton buttonCancel`
 - The cancel button.
- `private final String CLICK_OK`
 - The constant used for the method `actionPerformed(...)`.
- `private final String CLICK_QUIT`
 - The constant used for the method `actionPerformed(...)`.

- private final String CLICK_CANCEL
 - The constant used for the method `actionPerformed(...)`.
- private boolean hasQuit
 - If this dialog box has the quit button, this is set to `true`. The cancel button is not shown then. If this is `false`, then the opposite.
- private int height
 - The dialog box height.
- private int width
 - The dialog box width.
- private Interactive parent
 - The parent window.

D.3.24.3 CONSTRUCTORS

- *SemesterYearSelector*

```
public SemesterYearSelector( edu.unl.consyslab.gui.Interactive
parent )
```

 - **Usage**
 - * Constructor.
 - **Parameters**
 - * `parent` - The parent.

- *SemesterYearSelector*

```
public SemesterYearSelector( edu.unl.consyslab.gui.Interactive
parent, boolean refetch )
```

- **Usage**

- * Constructor.

- **Parameters**

- * *parent* - The parent.

- * *refetch* - If this is `true`, then the cancel button is shown instead of the quit button. Otherwise, the quit button is shown and not the cancel button.

D.3.24.4 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

- *getYearSemester*

```
public YearSemester getYearSemester( )
```

- **Usage**

- * Returns the year and semester.

- **Returns** - The `YearSemester` data structure.

- **See Also**

- * `edu.unl.consyslab.gtajava.YearSemester` (in D.9.4, page 200)

D.3.25 CLASS `StringRenderer`

This class just is an extended cell renderer. This is used for colored cells.

D.3.25.1 DECLARATION

```
public class StringRenderer
    extends javax.swing.table.DefaultTableCellRenderer
    implements javax.swing.table.TableCellRenderer
```

D.3.25.2 CONSTRUCTORS

- *StringRenderer*

```
public StringRenderer( java.lang.String label )
```

- Usage

- * Constructor.

- Parameters

- * `label` - The label to display.

- *StringRenderer*

```
public StringRenderer( java.lang.String label, java.awt.Color
color )
```

- Usage

- * Constructor.

– Parameters

- * `label` - The label to display.
- * `color` - The cell background color.

D.3.25.3 METHODS

- *getTableCellRendererComponent*

```
public Component getTableCellRendererComponent(
    javax.swing.JTable table, java.lang.Object value, boolean isSe-
    lected, boolean hasFocus, int row, int column )
```

- *toString*

```
public String toString( )
```

D.3.26 CLASS TunnelAuthenticationDialog

This class prompts the user for their CSE username and password. These will be used to setup the SSH tunnel for MySQL access.

D.3.26.1 DECLARATION

```
public class TunnelAuthenticationDialog
    extends java.lang.Object
    implements java.awt.event.ActionListener
```

D.3.26.2 CONSTRUCTORS

- *TunnelAuthenticationDialog*

```
public TunnelAuthenticationDialog( )
```

– **Usage**

* Constructor.

D.3.26.3 METHODS

- *actionPerformed*

```
public void actionPerformed( java.awt.event.ActionEvent e )
```

-
- *getPassword*

```
public String getPassword( )
```

– **Usage**

* Returns the password from the password field.

– **Returns** - The password from the password field.

-
- *getUsername*

```
public String getUsername( )
```

– **Usage**

* Returns the username from the username field.

– **Returns** - The username from the username field.

D.4 Package edu.unl.consystlab.gtaap.constraints

Package Contents

Page

Classes

CapacityConstraint	165
<i>The CapacityConstraint ensures that a given GTA is not assigned courses such that the sum of the course loads do not exceed the GTA's defined maxCapacity.</i>	
CertificationConstraint	168
<i>The CertificationConstraint ensures that a given GTA is not assigned a course he/she is not certified to teach.</i>	
EqualityConstraint	170
<i>Equality constraints are binary constraints in place between any two courses that require the GTA to be the same.</i>	
MutexConstraint	173
<i>Mutex constraints are binary constraints in place between any two courses that require GTAs as instructors and meet during overlapping times.</i>	
NilPrefConstraint	176
<i>The NilPrefConstraint ensures that a given GTA is not assigned courses that the GTA specified a preference 0 for.</i>	
OverlapConstraint	178
<i>The OverlapConstraint ensures that a given GTA is not assigned another teaching courses at the same time.</i>	
TakingCourseConstraint	180
<i>The TakingCourseConstraint ensures that a given GTA is not assigned a teaching courses at the same time he/she is enrolled in a course.</i>	

D.5 Classes

D.5.1 CLASS CapacityConstraint

The CapacityConstraint ensures that a given GTA is not assigned courses such that the sum of the course loads do not exceed the GTA's defined maxCapacity.

D.5.1.1 DECLARATION

```
public class CapacityConstraint
extends edu.unl.consyslab.CSPConstraint
```

D.5.1.2 FIELDS

- public Vector courses
 - A Vector of courses this constraint covers.
- public GTA gta
 - The GTA this constraint is for.
- public Vector variables
 - A Vector of CSPVariable that this constraint covers.
- public CSPValue value
 - The CSPValue this constraint is for.
- public double maxCapacity

- The maximum load.
- `public double currLoad`
 - The current load this GTA has.
- `public Vector assignedVariables`
 - A `Vector` of currently assigned variables in the scope of this constraint.

D.5.1.3 CONSTRUCTORS

- *CapacityConstraint*

```
public CapacityConstraint( )
```

 - **Usage**
 - * Creates a new instance of Capacity

D.5.1.4 METHODS

- *check*

```
public boolean check( java.util.Vector vvps )
```

 - **Usage**
 - * Checks the vvp pair list to see if the list is consistent.
 - **Parameters**
 - * `vvps` - The VVP pair.
 - **Returns** - `true` if it is valid, `false` otherwise.

- *check*

```
public boolean check( java.util.Vector  vvps, double  sum )
```

- **Usage**

- * Checks the vvp pair list to see if the list is consistent.

- **Parameters**

- * *vvps* - A Vector of CSPVVP to run the check on.

- * *sum* - the initial start sum

- **Returns** - true if it is valid, false otherwise.

- *isConsistent*

```
public boolean isConsistent( )
```

- **Usage**

- * If the constraint has not been violated, then this method returns **true**.

- **Returns** - true if this constraint remains consistent, false otherwise.

- *print_assignedVariables*

```
public void print_assignedVariables( )
```

- **Usage**

- * Prints the assigned variables

- *set_assignedVariables*

```
public void set_assignedVariables( edu.unl.consyslab.CSPVariable  
[] v )
```


- **Usage**

- * Sets the assigned variables.

- **Parameters**

- * *v* - The array of variables that are assigned.

- *toString*

```
public String toString( )
```

D.5.2 CLASS CertificationConstraint

The CertificationConstraint ensures that a given GTA is not assigned a course he/she is not certified to teach. For example, teaching courses must have GTAs that are ITA qualified.

D.5.2.1 DECLARATION

```
public class CertificationConstraint
    extends edu.unl.consyslab.CSPConstraint
```

D.5.2.2 FIELDS

- public Course course
 - The course that this constraint covers.
- public Vector definition
 - The **Vector** of GTAs that are certified to teach.

D.5.2.3 CONSTRUCTORS

- *CertificationConstraint*

```
public CertificationConstraint( )
```

- **Usage**

- * Constructor.

D.5.2.4 METHODS

- *check*

```
public boolean check( edu.unl.consyslab.CSPVVP vvp )
```

- **Usage**

- * Checks the vvp pair list to see if the list is consistent.

- **Parameters**

- * vvps - The VVP pair.

- **Returns** - true if it is valid, false otherwise.

- *getCourse*

```
public Course getCourse( edu.unl.consyslab.CSPVariable var )
```

- **Usage**

- * A method to quickly obtain the Course from this constraint.

- **Returns** - The Course from this constraint.

- *getGTA*

```
public GTA getGTA( edu.unl.consyslab.CSPValue val )
```

- **Usage**

- * A method to quickly obtain the GTA from this constraint.

- **Returns** - The GTA from this constraint.

- *isValid*

```
public boolean isValid( edu.unl.consyslab.CSPVariable var,
edu.unl.consyslab.CSPValue val )
```

- **Usage**

- * Is var compatible with the value of val? All must answer “true”: - is this course lecture, recitation, or lab? - is the GTA’s ITA qualification status “citizen” (1) or “passed” (5)? - is this GTA contained in the definition vector?

- **Parameters**

- * `var` - The CSPVariable.

- * `val` - The CSPValue.

- **Returns** - true if it is valid, else false.

- *toString*

```
public String toString( )
```

D.5.3 CLASS EqualityConstraint

Equality constraints are binary constraints in place between any two courses that require the GTA to be the same. This is a binary constraint.

D.5.3.1 DECLARATION

```
public class EqualityConstraint
extends edu.unl.consyslab.CSPConstraint
```

D.5.3.2 FIELDS

- public `CSPProblem` `problem`
 - The `CSPProblem` this constraint is for.

D.5.3.3 CONSTRUCTORS

- *EqualityConstraint*

```
public EqualityConstraint( )
```

 - **Usage**
 - * Creates a new instance of `Mutex`

D.5.3.4 METHODS

- *check*

```
public boolean check( edu.unl.consyslab.CSPVVP vvp1,
edu.unl.consyslab.CSPVVP vvp2 )
```

- **Usage**
 - * Checks if vvp1 and vvp2 are compatible.
- **Parameters**
 - * vvp1 - CSPVVP to check againsts vvp2.
 - * vvp2 - CSPVVP to check againsts vvp1.
- **Returns** - true if compatible; false otherwise.

- *getGta*

```
public GTA getGta( )
```

- **Usage**
 - * A method to quickly obtain the GTA from this constraint.
- **Returns** - The GTA from this constraint.

- *getValue*

```
public CSPValue getValue( )
```

- **Returns** - Returns the value.

- *getVariables*

```
public Vector getVariables( )
```

- **Returns** - Returns the variables.

- *setValue*

```
public void setValue( edu.unl.consyslab.CSPValue value )
```

- **Parameters**

* `value` - The value to set.

- *setVariables*

```
public void setVariables( java.util.Vector variables )
```

– **Parameters**

* `variables` - The variables to set.

- *toString*

```
public String toString( )
```

D.5.4 CLASS `MutexConstraint`

`Mutex` constraints 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/she is instructing. This is a binary constraint.

D.5.4.1 DECLARATION

```
public class MutexConstraint
    extends edu.unl.consystlab.CSPConstraint
```

D.5.4.2 FIELDS

- public `CSPProblem` `problem`
 - The `CSPProblem` this constraint is for.

D.5.4.3 CONSTRUCTORS

- *MutexConstraint*

```
public MutexConstraint( )
```

- **Usage**

- * Creates a new instance of `Mutex`

D.5.4.4 METHODS

- *check*

```
public boolean check( edu.unl.consyslab.CSPVVP vvp1,
edu.unl.consyslab.CSPVVP vvp2 )
```

- **Usage**

- * Checks if `vvp1` and `vvp2` are compatible.

- **Parameters**

- * `vvp1` - `CSPVVP` to check againsts `vvp2`.

- * `vvp2` - `CSPVVP` to check againsts `vvp1`.

- **Returns** - `true` if compatible; `false` otherwise.

- *getGta*

```
public GTA getGta( )
```

- **Usage**

- * A method to quickly obtain the `GTA` from this constraint.

- **Returns** - The `GTA` from this constraint.

- *getValue*

```
public CSPValue getValue( )
```

– **Returns** - Returns the value.

- *getVariables*

```
public Vector getVariables( )
```

– **Returns** - Returns the variables.

- *setValue*

```
public void setValue( edu.unl.consyslab.CSPValue value )
```

– **Parameters**

* *value* - The value to set.

- *setVariables*

```
public void setVariables( java.util.Vector variables )
```

– **Parameters**

* *variables* - The variables to set.

- *toString*

```
public String toString( )
```


D.5.5 CLASS NilPrefConstraint

The NilPrefConstraint ensures that a given GTA is not assigned courses that the GTA specified a preference 0 for.

D.5.5.1 DECLARATION

```
public class NilPrefConstraint
    extends edu.unl.consyslab.CSPConstraint
```

D.5.5.2 FIELDS

- public Course course
 - The **Source** that this constraint covers.
- public Vector definition
 - The **Vector** of GTAs that have preference 0 for this course.

D.5.5.3 CONSTRUCTORS

- *NilPrefConstraint*

```
public NilPrefConstraint( )
```

 - **Usage**
 - * Default constructor.

D.5.5.4 METHODS

- *getCourse*

```
public Course getCourse( edu.unl.consyslab.CSPVariable var )
```

- **Usage**

- * A method to quickly obtain the **Course** from this constraint.

- **Returns** - The **Course** from this constraint.

- *getGTA*

```
public GTA getGTA( edu.unl.consyslab.CSPValue val )
```

- **Usage**

- * A method to quickly obtain the **GTA** from this constraint.

- **Returns** - The **GTA** from this constraint.

- *isValid*

```
public boolean isValid( edu.unl.consyslab.CSPVariable var,
edu.unl.consyslab.CSPValue val )
```

- **Usage**

- * Is var compatible with the value of val? If this GTA contained in the definition vector, then it is **false**.

- **Parameters**

- * **var** - The **CSPVariable**.

- * **val** - The **CSPValue**.

- **Returns** - true if it is valid, else false.

-
- *toString*

```
public String toString( )
```

D.5.6 CLASS `OverlapConstraint`

The `OverlapConstraint` ensures that a given GTA is not assigned another teaching courses at the same time.

D.5.6.1 DECLARATION

```
public class OverlapConstraint
    extends edu.unl.consystlab.CSPConstraint
```

D.5.6.2 FIELDS

- public `Course` `course`
 - The `Course` that this constraint covers.
- public `Vector` definition
 - The `Vector` of GTAs that are not assigned another course at the same time..

D.5.6.3 CONSTRUCTORS

- *OverlapConstraint*

```
public OverlapConstraint( )
```

- **Usage**

- * Default constructor.

D.5.6.4 METHODS

- *getCourse*

```
public Course getCourse( edu.unl.consyslab.CSPVariable var )
```

- **Usage**

- * A method to quickly obtain the **Course** from this constraint.

- **Returns** - The **Course** from this constraint.

- *getGTA*

```
public GTA getGTA( edu.unl.consyslab.CSPValue val )
```

- **Usage**

- * A method to quickly obtain the **GTA** from this constraint.

- **Returns** - The **GTA** from this constraint.

- *isValid*

```
public boolean isValid( edu.unl.consyslab.CSPVariable var,  
edu.unl.consyslab.CSPValue val )
```

- **Usage**

* Is var compatible with the value of val? If this GTA contained in the definition vector, then it is "false".

– **Parameters**

* **var** - The CSPVariable.

* **val** - The CSPValue.

– **Returns** - true if it is valid, else false.

• *toString*

```
public String toString( )
```

D.5.7 CLASS TakingCourseConstraint

The TakingCourseConstraint ensures that a given GTA is not assigned a teaching courses at the same time he/she is enrolled in a course.

D.5.7.1 DECLARATION

```
public class TakingCourseConstraint
    extends edu.unl.consystlab.CSPConstraint
```

D.5.7.2 FIELDS

- public Course course
 - The **Source** that this constraint covers.
- public Vector definition

- The Vector of GTAs that are taking another course at the same time..

D.5.7.3 CONSTRUCTORS

- *TakingCourseConstraint*

```
public TakingCourseConstraint( )
```

D.5.7.4 METHODS

- *getCourse*

```
public Course getCourse( edu.unl.consyslab.CSPVariable var )
```

- **Usage**

* A method to quickly obtain the Course from this constraint.

- **Returns** - The Course from this constraint.
-

- *getGTA*

```
public GTA getGTA( edu.unl.consyslab.CSPValue val )
```

- **Usage**

* A method to quickly obtain the GTA from this constraint.

- **Returns** - The GTA from this constraint.
-

- *isValid*

```
public boolean isValid( edu.unl.consyslab.CSPVariable var,
edu.unl.consyslab.CSPValue val )
```

– **Usage**

- * Is var compatible with the value of val? If this GTA contained in the definition vector, then it is "false".

– **Parameters**

- * **var** - The CSPVariable.
- * **val** - The CSPValue.

– **Returns** - true if it is valid, else false.

• *toString*

```
public String toString( )
```

D.6 Package edu.unl.consystlab.gtaap

Package Contents

Page

Classes

Course 184

This class is the data structure for the courses offered in the department.

GTA.....187

This class is the data structure for the candidate GTAs.

Preference..... 190

This class is the data structure for the GTA preferences.

D.7 Classes

D.7.1 CLASS Course

This class is the data structure for the courses offered in the department.

D.7.1.1 DECLARATION

```
public class Course
    extends java.lang.Object
    implements java.lang.Comparable
```

D.7.1.2 FIELDS

- public int dbId
 - The database identifier for this course.
- public String courseDept
 - The department offering this course. For example, CSCE, MATH, ENGL, ENGR, etc.
- public String courseNumber
 - The 3/6 digit course number. For example, 101, 421-821, etc.
- public String section
 - The section number for the course.

- public String title
 - The title for the course.
- public GTA assignedTA
 - The assigned GTA for this course.
- public String startTime
 - The start time for this course in 24 hour format. For example, 3.15pm = 1515, 8.30am = 0830, etc.
- public String endTime
 - The end time for this course in 24 hour format. For example, 3.15pm = 1515, 8.30am = 0830, etc.
- public boolean day_m
 - The class meets on Monday, then this is set to **true**, otherwise **false**.
- public boolean day_t
 - The class meets on Tuesday, then this is set to **true**, otherwise **false**.
- public boolean day_w
 - The class meets on Wednesday, then this is set to **true**, otherwise **false**.
- public boolean day_r
 - The class meets on Thursday, then this is set to **true**, otherwise **false**.
- public boolean day_f
 - The class meets on Friday, then this is set to **true**, otherwise **false**.
- public double load

- The required commitment from the GTA in double format. For example, 0.25, 0.33, 1.0, etc.
- public String type
 - The type of course. Valid values are “lecture”, “grading”, “lab”, “recitation”.
- public String parentCourseNumber
 - The parent course number (for labs, recitations, grading).
- public String parentSection
 - The parent course section (for labs, recitations, grading).
- public int parentDbId
 - The parent course database identifier (for labs, recitations, grading).
- public boolean shortCourse
 - This course is a short course.
- public String shortCourseStartDate
 - The course start date.
- public String shortCourseEndDate
 - The course end date.

D.7.1.3 CONSTRUCTORS

- *Course*
public Course()

– Usage

* Default constructor.

D.7.1.4 METHODS

- *compareTo*

```
public int compareTo( java.lang.Object nCompare )
```

- *equals*

```
public boolean equals( java.lang.Object nCompare )
```

- *hashCode*

```
public int hashCode( )
```

- *showDetails*

```
public String showDetails( )
```

– Usage

* Returns the details for this course.

– **Returns** - The details for this course in **string** format.

- *toString*

```
public String toString( )
```

D.7.2 CLASS GTA

This class is the data structure for the candidate GTAs.

D.7.2.1 DECLARATION

```
public class GTA
extends java.lang.Object
implements java.lang.Comparable
```

D.7.2.2 FIELDS

- public int dbId
 - The identifier given to this GTA corresponding to the ID in the database.
- public String name
 - The GTA's name.
- public int advisorId
 - The GTA's advisor id wrt the database id
- public String advisorName
 - The GTA's advisor name.
- public int ITA
 - The ITA status of the GTA. 1: Passed 2: Failed 3: Did not attend 4: Will take next session 5: Not required (US Citizen)
- public int speak
 - SPEAK score

- `public Vector courseList`
 - A Vector of Preferences.
- `public double capacity`
 - The maximum capacity (load) that this GTA can handle.
- `public double currentCapacity`
 - The current capacity

D.7.2.3 CONSTRUCTORS

- *GTA*

```
public GTA( )
```

 - Usage
 - * The default constructor for the GTA class. This just initializes the Vector fields.

D.7.2.4 METHODS

- *compareTo*

```
public int compareTo( java.lang.Object nCompare )
```

- *equals*

```
public boolean equals( java.lang.Object nCompare )
```

- *hashCode*

```
public int hashCode( )
```

- *itaStatus*

```
public boolean itaStatus( )
```

– **Usage**

* Returns the ITA status, whether the GTA is ITA certified or not.

– **Returns** - true if the GTA is certified, false otherwise.

- *toString*

```
public String toString( )
```

D.7.3 CLASS Preference

This class is the data structure for the GTA preferences.

D.7.3.1 DECLARATION

```
public class Preference
    extends java.lang.Object
```

D.7.3.2 FIELDS

- public Course course
 - The course.

- `public String courseName`
 - The course name.
- `public int preference`
 - The preference value for this course. 0 is the absolute lowest preference while 5 is the highest.
- `public boolean enrolled`
 - `true` if the GTA is enrolled in this course, `false` otherwise.

D.7.3.3 CONSTRUCTORS

- *Preference*
 - `public Preference()`
 - **Usage**
 - * Default constructor.

D.8 Package edu.unl.consyslab.gtajava

Package Contents

Page

Classes

GTAAPConfig	193
<i>Configuration file for the GTAAP problem.</i>	
Main	197
<i>Forward-check test.</i>	
RandomGTAAPGenerator	198
<i>Randomly generates GTAs and Courses.</i>	
YearSemester	200
<i>This class is a data structure to “transport” the year and semesters from class to class.</i>	

D.9 Classes

D.9.1 CLASS GTAAPConfig

Configuration file for the GTAAP problem. In this file, we declare all the configurations that are needed to run the program.

D.9.1.1 DECLARATION

```
public class GTAAPConfig
    extends java.lang.Object
```

D.9.1.2 FIELDS

- public static int verbosity
 - Defines the level of verbosity to be displayed. 0 = silent 1 = warnings 2 = above + more messages 5 = regular debug messages 10 = everything
- public static String dbServerHost
 - The database server host.
- public static String dbPort
 - The port to connect to dbServerHost.
- public static String dbName
 - The database name.

- `public static String dbUser`
 - The database user to connect to `dbName` on `dbServerHost`.
- `public static String dbPassword`
 - The database password for `dbUser`.
- `public static PortForwardingL portForwarding`
 - Prepare for SSH tunneling
- `public static TunnelAuthenticationDialog tunnelAuthenticationDialog`
 - Tunnel authentication dialog.
- `public static String tunnelUsername`
 - The SSH tunnel username.
- `public static String tunnelPassword`
 - The SSH tunnel user's password.
- `public static int currentYear`
 - The current year.
- `public static int currentSemester`
 - The current semester.
- `public static int randSeed`
 - Constant random seed
- `public static int uuidMax`
 - Maximum value for a universally unique identifier.

- `public static final int NONINTERACTIVE`
 - The modes available for this program. We either run it as non-interactive or interactive.
- `public static final int INTERACTIVE`
 - The modes available for this program. We either run it as non-interactive or interactive.
- `public static int mode`
 - Which mode are we running in now?
- `public static int AC1_maxLoops`
 - ArcConsistency maximum number of loops
- `public static Color gtaColorBusy`
 - GTA pulldown menu colors for a busy GTA.
- `public static Color gtaColorBusySelected`
 - GTA pulldown menu colors for a selected busy GTA.
- `public static Color gtaColorAvailable`
 - GTA pulldown menu colors for an available GTA.
- `public static Color gtaColorAvailableSelected`
 - GTA pulldown menu colors for a selected available GTA.
- `public static Color gtaColorAlreadyAssigned`
 - GTA pulldown menu colors for an already assigned GTA.
- `public static Color gtaColorPreassigned`
 - GTA pulldown menu colors for a GTA who is preassigned.

D.9.1.3 CONSTRUCTORS

- *GTAAPConfig*

```
public GTAAPConfig( )
```

D.9.1.4 METHODS

- *generateUuid*

```
public static int generateUuid( )
```

- **Usage**

- * Generates a (hopefully) universally unique identifier for all instances in this problem.

- **Returns** - The universally unique identifier.

- *getLocalMySQLPort*

```
public static int getLocalMySQLPort( )
```

- **Usage**

- * Returns the local MySQL port. If it is not yet initialized, it will be initialized to a random port number.

- *getTunnelAuthenticationDialog*

```
public static TunnelAuthenticationDialog getTunnelAuthenticati-
onDialog( )
```

- **Usage**

* This method will ensure that the TunnelAuthenticationDialog is only called once. To recall this dialog, set `tunnelAuthenticationDialog = null` and recall this method.

– **Returns** - The single (and only) instance of TunnelAuthenticationDialog.

D.9.2 CLASS Main

Forward-check test.

D.9.2.1 DECLARATION

```
public class Main
    extends java.lang.Object
```

D.9.2.2 CONSTRUCTORS

- *Main*

```
public Main( )
```

D.9.2.3 METHODS

- *main*

```
public static void main( java.lang.String [] args )
```

D.9.3 CLASS RandomGTAAPGenerator

Randomly generates GTAs and Courses. Access the Courses by the `Vector` `vec_course`, the GTAs by `Vector` `vec_gta`, the variables by `Vector` `vec_variables`, and the values by `Vector` `vec_values`.

D.9.3.1 DECLARATION

```
public class RandomGTAAPGenerator
    extends java.lang.Object
```

D.9.3.2 FIELDS

- `public Vector vec_course`
 - Stores the course entries.
- `public Vector vec_gta`
 - Stores the GTA entries.
- `public Vector vec_variables`
 - Stores the variable entries.
- `public Vector vec_values`
 - Stores the value entries.
- `public Hashtable hash_constraints`
 - Stores the constraints in a hashtable

D.9.3.3 CONSTRUCTORS

- *RandomGTAAPGenerator*

```
public RandomGTAAPGenerator( long seed )
```

– Usage

- * Constructor for RandomGTAAPGenerator. This initializes the load table and the vectors involved for courses and the GTAs.

D.9.3.4 METHODS

- *generateRandomCourses*

```
public void generateRandomCourses( int n )
```

– Usage

- * Generates random Courses

– Parameters

- * **n** - The number of courses to randomly generate.
-

- *generateRandomGTAs*

```
public void generateRandomGTAs( int n )
```

– Usage

- * Generates random GTAs

– Parameters

- * **n** - The number of GTAs to randomly generate.
-

- *makeRandom*

```
public void makeRandom( int nGta, int nCourse )
```

– **Usage**

- * Generates random Courses and GTAs.

– **Parameters**

- * **nGta** - The number of GTAs to randomly create.
- * **nCourse** - The number of Courses to randomly create.

D.9.4 CLASS YearSemester

This class is a data structure to “transport” the year and semesters from class to class.

D.9.4.1 DECLARATION

```
public class YearSemester
    extends java.lang.Object
```

D.9.4.2 CONSTRUCTORS

- *YearSemester*

```
public YearSemester( int year, int semester )
```

D.9.4.3 METHODS

- *getSemester*

```
public int getSemester( )
```

– **Returns** - Returns the semester.

- *getSemesterName*

```
public String getSemesterName( )
```

– **Usage**

* This method returns the string name for the given semester. 1 = Fall;
2 = Spring; 11 = Summer (pre); etc.

– **Returns** - The string name for the semester.

- *getYear*

```
public int getYear( )
```

– **Returns** - Returns the year.

- *semesterNameToCode*

```
public static int semesterNameToCode( java.lang.String name  
)
```

– **Usage**

* This method returns the semester number given the **String** name. 1
= Fall; 2 = Spring; 11 = Summer (pre); etc.

– **Returns** - The semester number for the semester name.

- *setSemester*

```
public void setSemester( int semester )
```

– Parameters

* `semester` - The semester to set.

• *setYear*

```
public void setYear( int year )
```

– Parameters

* `year` - The year to set.

• *toString*

```
public String toString( )
```

D.10 Package edu.unl.consyslab

Package Contents

Page

Classes

CSPConstraint	205
<i>The skeleton/base class for constraints.</i>	
CSPProblem	206
<i>This is the data structure that holds the required data for running our solvers.</i>	
CSPUtils	210
<i>Some basic CSP utilities that are used.</i>	
CSPValue	211
<i>This class defines the CSP Value object.</i>	
CSPVariable	214
<i>This class defines the CSP Variable object.</i>	
CSPVVP	218
<i>The data structure for variable-value pairs.</i>	
EVector	219
<i>This extends the Vector data structure allows us to obtain the hashcode of the items in the Vector.</i>	
FC_Solver	220
<i>A forward-check solver.</i>	
jAssignment	226
<i>jAssignment is the data structure for transferring the saved assignments from the database to the graphical interface (and vice-versa).</i>	
LabelParams	231
<i>LabelParams allows parameters to be passed by reference for fcLabel and fcUnlabel methods.</i>	
Log	233

This class is a placeholder to simplify the logging of events.

Setup	234
<i>This class sets up the CSPPProblem so we don't have to manually do it each time.</i>	
Utils	236
<i>This class provides commonly used functions and methods that are globally available for other classes to use.</i>	

D.11 Classes

D.11.1 CLASS CSPConstraint

The skeleton/base class for constraints.

D.11.1.1 DECLARATION

```
public class CSPConstraint
    extends java.lang.Object
```

D.11.1.2 FIELDS

- public String id
 - The **String** identifier for this constraint.
- public CSPProblem problem
 - The problem this constraint is associated with.
- public Vector variables
 - The variables that this constraint is covers.

D.11.1.3 CONSTRUCTORS

- *CSPConstraint*
public **CSPConstraint**()

– Usage

- * Default constructor.

D.11.2 CLASS CSPPProblem

This is the data structure that holds the required data for running our solvers.

D.11.2.1 DECLARATION

```
public class CSPPProblem
    extends java.lang.Object
    implements java.lang.Cloneable
```

D.11.2.2 FIELDS

- public Vector value
 - The list of value (the domain). This list (**Vector**) holds objects of type **CSPValue**.
- public Vector variable
 - The list of variable. This list (**Vector**) holds objects of type **CSPVariable**.
- public Vector staticVariable
 - The list of static/permanent/preassigned variables. This list (**Vector**) holds objects of type **CSPVariable**.

- public Hashtable constraint
 - This hashtable contains the `Course` as key and a `Vector` of constraints as values
- public int constraintCount
 - The constraint count.
- public Vector saved
 - The saved assignments are retrieved and stored here. `saved` takes values of type `jAssignment`

D.11.2.3 CONSTRUCTORS

- *CSPPProblem*

```
public CSPPProblem( )
```

- Usage

- * Constructor.

- *CSPPProblem*

```
public CSPPProblem( java.lang.String sid, java.util.Vector
svalue, java.util.Vector svariable, java.util.Vector sclosed-
Variable, java.util.Vector sstaticVariable, java.util.Hashtable
sconstraint )
```

- Usage

- * Constructor.

- Parameters

- * `sid` - The String ID for this problem
- * `svalue` - The Vector of values
- * `svariable` - The Vector of variables
- * `sclosedVariable` - The Vector of closed variables (closed courses)
- * `sstaticVariable` - The Vector of static variables
- * `sconstraint` - The Hashtable of constraints.

D.11.2.4 METHODS

- *clone*

```
public Object clone( )
```

- *get_constraint*

```
public Hashtable get_constraint( )
```

– **Usage**

- * Gets the constraint.

– **Returns** - The constraint.

- *get_id*

```
public String get_id( )
```

– **Usage**

- * Returns the problem identifier.

– **Returns** - The problem identifier.

- *getSemester*

```
public int getSemester( )
```

- **Usage**
 - * Returns the semester.
 - **Returns** - The semester.
-

- *getYear*

```
public int getYear( )
```

- **Usage**
 - * Returns the year.
 - **Returns** - The year.
-

- *set_id*

```
public void set_id( java.lang.String sid )
```

- **Usage**
 - * Set the problem identifier.
 - **Parameters**
 - * `sid` - The identifier.
-

- *setSemester*

```
public void setSemester( int semester )
```

- **Usage**
 - * Sets the semester.
 - **Parameters**
 - * `semester` - The semester to set.
-

- *setYear*

```
public void setYear( int year )
```

- **Usage**

- * Sets the year.

- **Parameters**

- * **year** - The year to set.

D.11.3 CLASS CSPUtils

Some basic CSP utilities that are used.

D.11.3.1 DECLARATION

```
public class CSPUtils  
  
extends java.lang.Object
```

D.11.3.2 CONSTRUCTORS

- *CSPUtils*

```
public CSPUtils( )
```

- **Usage**

- * Creates a new instance of CSPUtils

D.11.3.3 METHODS

- *buildTimePoints*

```
public Vector buildTimePoints( java.util.Vector varList )
```

- *courseCliques*

```
public void courseCliques( edu.unl.consyslab.CSPPProblem problem )
```

- *coveringCliques*

```
public void coveringCliques( edu.unl.consyslab.CSPPProblem problem )
```

- *getCliques*

```
public Vector getCliques( java.util.Vector varList )
```

D.11.4 CLASS CSPValue

This class defines the CSP Value object.

D.11.4.1 DECLARATION

```
public class CSPValue
  extends java.lang.Object
  implements java.lang.Comparable
```

D.11.4.2 FIELDS

- public String id
 - the string id of this value
- public Object value
 - The value object for this CSPValue. For the GTAAP problem, the value should be of type **GTA**

D.11.4.3 CONSTRUCTORS

- *CSPValue*

```
public CSPValue( )
```

- **Usage**

- * Default constructor that does nothing.
-

- *CSPValue*

```
public CSPValue( java.lang.String sid, java.lang.Object svalue
 )
```

- **Usage**

- * Constructor.

- **Parameters**

- * **sid** - The String id for this value
- * **svalue** - The Object that this value takes.

D.11.4.4 METHODS

- *compareTo*

```
public int compareTo( java.lang.Object nCompare )
```

- *getGTACapacity*

```
public double getGTACapacity( )
```

- **Usage**

- * A quick way to obtain the GTA capacity for this CSPValue

- **Returns** - Returns a double value for the capacity for the GTA in this CSPValue.
-

- *getGTACurrentCapacity*

```
public double getGTACurrentCapacity( )
```

- **Usage**

- * A quick way to obtain the GTA current capacity for this CSPValue

- **Returns** - Returns a double value for the current capacity for the GTA in this CSPValue.
-

- *hashCode*

```
public int hashCode( )
```

- *setGTACurrentCapacity*

```
public void setGTACurrentCapacity( double capacity )
```

- **Usage**

* Sets the GTA in this CSPValue's current capacity

– **Parameters**

* *capacity* - A double value to be set as the GTA's current capacity.

• *toString*

```
public String toString( )
```

D.11.5 CLASS CSPVariable

This class defines the CSP Variable object.

D.11.5.1 DECLARATION

```
public class CSPVariable
    extends java.lang.Object
    implements java.lang.Comparable
```

D.11.5.2 FIELDS

- public String *id*
 - the string id of this variable.
- public String *type*
 - the type of this variable (lecture, recitation, etc.)
- public Object *variable*

- The variable object for this CSPVariable. For the GTAAP problem, the variable should be of type **Course**.
- public Object problem
 - The encapsulation of the problem
- public Vector constraint
 - the constraint of type any arbitrary constraint type (not CSPConstraint)
- public Vector initialDomain
 - the initial domain (vector of CSPValue)
- public Vector currentDomain
 - the current domain
- public Vector savedDomain
 - Initial arc-consistent domain.
- public Vector filteredOutDomain
 - The savedDomain “currentDomain
- public CSPValue assignedValue
 - The assigned value to this variable
- public boolean preassigned
 - Is this variable preassigned? If yes, then true, otherwise false.

D.11.5.3 CONSTRUCTORS

- *CSPVariable*

```
public CSPVariable( )
```

- Usage

- * Default constructor.

- *CSPVariable*

```
public CSPVariable( java.lang.String sid, java.util.Hashtable  
svariable, java.util.Hashtable sproblem, java.util.Vector  
sconstraint, java.util.Vector sinitialDomain, java.util.Vector  
sneighborCSPVariables )
```

- Usage

- * Constructor

- Parameters

- * `sid` - The string id of this variable.
 - * `svariable` - The variable
 - * `sproblem` - The problem
 - * `sconstraint` - The Vector of constraints
 - * `sinitialDomain` - The Vector of initial domain.
 - * `sneighborCSPVariables` - The Vector of neighbor variables.

D.11.5.4 METHODS

- *calculateFilteredOutDomain*

```
public void calculateFilteredOutDomain( )
```

– Usage

- * Finds out which variables were filtered out. This method returns void but stores the filtered-out variables in the class variable vector filtered-OutDomain.

- *compareTo*

```
public int compareTo( java.lang.Object nCompare )
```

- *equals*

```
public boolean equals( java.lang.Object nCompare )
```

- *getId*

```
public String getId( )
```

- *getCourseLoad*

```
public double getCourseLoad( )
```

– Usage

- * Sort cut to get the course load for this variable.

– Returns - double value for the load

- *hashCode*

```
public int hashCode( )
```

- *setId*

```
public void setId( java.lang.String sid )
```

- *toString*

```
public String toString( )
```

D.11.6 CLASS CSPVVP

The data structure for variable-value pairs.

D.11.6.1 DECLARATION

```
public class CSPVVP
    extends java.lang.Object
```

D.11.6.2 FIELDS

- public String id
 - The string id of this VVP
- public CSPVariable variable
 - The variable
- public Object value
 - The encapsulation of the value
- public CSPProblem problem
 - The encapsulation of the problem

D.11.6.3 CONSTRUCTORS

- *CSPVVP*

```
public CSPVVP( )
```

- **Usage**

- * The default constructor.

- *CSPVVP*

```
public CSPVVP( edu.unl.consyslab.CSPVariable var,
edu.unl.consyslab.CSPValue val )
```

- **Usage**

- * Initializes the CSPVVP with variable var and value val.

- **Parameters**

- * var - The variable to be initialized with.

- * val - The value to be initialized with.

D.11.6.4 METHODS

- *toString*

```
public String toString( )
```

D.11.7 CLASS EVector

This extends the Vector data structure allows us to obtain the hashcode of the items in the Vector. The hashcode is calculated by summing up the hashcodes of the items in the vector. A vector with elements (in order) x1, x2, x3 is equivalent to a vector with elements (in order) x3, x2, x1.

D.11.7.1 DECLARATION

```
public class EVector
    extends java.util.Vector
```

D.11.7.2 CONSTRUCTORS

- *EVector*

```
public EVector( )
```

D.11.7.3 METHODS

- *equals*

```
public boolean equals( java.lang.Object o )
```

- *hashCode*

```
public int hashCode( )
```

D.11.8 CLASS FC_Solver

A forward-check solver.

D.11.8.1 DECLARATION

```
public class FC_Solver
    extends java.lang.Object
```

D.11.8.2 FIELDS

- `public CSPPProblem problem`
 - The problem.
- `public Date startTime`
 - The start time.
- `public Date endTime`
 - The end time.

D.11.8.3 CONSTRUCTORS

- *FC_Solver*

```
public FC_Solver( edu.unl.consyslab.CSPPProblem argProblem
)
```

 - **Usage**
 - * Constructor.
 - **Parameters**
 - * `argProblem` - The problem.

D.11.8.4 METHODS

- *bcssp*

```
public void bcssp( int n, java.lang.String argStatus )
```

– **Usage**

* The search function.

– **Parameters**

* **n** - The level.

* **argStatus** - The status.

• *check*

```
public boolean check( int i, int j )
```

– **Usage**

* Check for support of assigned values of $v[i]$ and $v[j]$.

– **Parameters**

* **i** - Variable i

* **j** - Variable j

– **Returns** - `true` if there is support, `false` otherwise.

• *checkForward*

```
public boolean checkForward( edu.unl.consyslab.LabelParams
params, int i, int j )
```

– **Usage**

* `checkForward` takes the current variable at position i and checks it againsts the future variable at position j ($i < j \leq n$). It removes from `currentDomain[j]` values that are inconsistent with variable at i .

– **Parameters**

* **i** - current variable position

* **j** - future variable position

– **Returns** - true if there is no domain annihilation, false otherwise.

- *doAssignment*

```
public boolean doAssignment( int i, edu.unl.consyslab.CSPValue
value )
```

– **Usage**

* Performs an assignment of a value to a variable at a particular level, i.

– **Parameters**

* i - The level i
 * value - The value to be assigned.

– **Returns** - Always returns true

- *fcLabel*

```
public int fcLabel( edu.unl.consyslab.LabelParams params )
```

– **Usage**

* fcUnlabel is used to uninstantiate a variable and perform backtracking.

– **Parameters**

* params - The LabelParams is used to pass the parameters by reference.

– **Returns** - If fcLabel(...) runs and remains consistent, return params.get_i() + 1, otherwise, returns params.get_i().

- *fcUnlabel*

```
public int fcUnlabel( edu.unl.consyslab.LabelParams params
)
```


– **Usage**

- * `fcUnlabel` is used to uninstantiate a variable and perform backtracking, that is, when we are not consistent.

– **Parameters**

- * `params` - The `LabelParams` is used to pass the parameters by reference.

– **Returns** - The current level (from `params.get_i()`).

- *printSolution*

```
public void printSolution( )
```

– **Usage**

- * Prints out the text representation of the solution.

- *searchCSP*

```
public void searchCSP( )
```

– **Usage**

- * This method starts the search.

- *swap*

```
public void swap( int i, int j )
```

– **Usage**

- * Swap the position of the *i*-th variable with the *j*-th in the current domain and path. This is needed to implement dynamic variable ordering.

– **Parameters**

* i - The i-th variable

* j - The j-th variable

- *undoAssignment*

```
public void undoAssignment( int i )
```

- **Usage**

- * Undo the assignment for the i-th variable.

- **Parameters**

- * i - The i-th variable.

- *undoReductions*

```
public void undoReductions( int i )
```

- **Usage**

- * Undo reductions at level i.

- **Parameters**

- * i - The level.

- *updateCapacity*

```
public void updateCapacity( edu.unl.consyslab.CSPVariable var,
edu.unl.consyslab.CSPValue val, java.lang.String op )
```

- **Usage**

- * Updates the capacity for the given GTA in val. Increases if op is "+" or decreases if op is "-" with load from CSPVariable var.

- **Parameters**

- * `var` - CSPVariable variable
- * `val` - CSPValue value
- * `op` - Operator is either "+" or "-"

- *updateCurrentDomain*

```
public void updateCurrentDomain( int i )
```

- **Usage**

- * Looks at all the reductions for $v[i]$ and removes them from the domain of $v[i]$.

- **Parameters**

- * `i` - The level `i`

- *var_load*

```
public double var_load( edu.unl.consyslab.CSPVariable var )
```

- **Usage**

- * Obtains the variable's course load.

- **Parameters**

- * `var` - The CSPVariable.

- **Returns** - The course load from the provided CSPVariable.

D.11.9 CLASS `jAssignment`

`jAssignment` is the data structure for transferring the saved assignments from the database to the graphical interface (and vice-versa).

D.11.9.1 DECLARATION

```
public class jAssignment
    extends java.lang.Object
```

D.11.9.2 CONSTRUCTORS

- *jAssignment*

```
public jAssignment( java.lang.String s, int year, int
semester, int g, int c )
```

- **Usage**

- * Stores the jAssignment value.

- **Parameters**

- * **s** - The saveId to use.

- * **year** - The year.

- * **semester** - The semester.

- * **g** - The GTA.

- * **c** - The class.

D.11.9.3 METHODS

- *getClassId*

```
public int getClassId( )
```

- **Usage**

* Gets the course identifier.

– **Returns** - The course identifier.

- *getGtaId*

```
public int getGtaId( )
```

– **Usage**

* Gets the GTA identifier.

– **Returns** - The course identifier.

- *getSavedId*

```
public int getSavedId( )
```

– **Usage**

* Gets the saved identifier (from the database).

– **Returns** - Returns the savedId.

- *getSavedName*

```
public String getSavedName( )
```

– **Usage**

* Gets the saved name (from the database).

– **Returns** - Returns the savedName.

- *getSemester*

```
public int getSemester( )
```

– **Usage**

* Gets the semester.

– **Returns** - Returns the semester.

- *getYear*

```
public int getYear( )
```

– **Usage**

* Gets the year.

– **Returns** - Returns the year.

- *setClassId*

```
public void setClassId( int classId )
```

– **Usage**

* Sets the course identifier.

– **Parameters**

* `classId` - The course identifier.

- *setGtaId*

```
public void setGtaId( int gtaId )
```

– **Usage**

* Sets the GTA identifier.

– **Parameters**

* `gtaId` - The GTA identifier.

- *setSavedId*

```
public void setSavedId( int savedId )
```

– **Usage**

* Sets the saved identifier.

– **Parameters**

* `savedId` - The `savedId` to set.

• *setSavedName*

```
public void setSavedName( java.lang.String savedName )
```

– **Usage**

* Sets the saved name.

– **Parameters**

* `savedName` - The `savedName` to set.

• *setSemester*

```
public void setSemester( int semester )
```

– **Usage**

* Sets the semester.

– **Parameters**

* `semester` - The semester to set.

• *setYear*

```
public void setYear( int year )
```

– **Usage**

* Sets the year.

– **Parameters**

* `year` - The year to set.

- *toString*

```
public String toString( )
```

D.11.10 CLASS `LabelParams`

`LabelParams` allows parameters to be passed by reference for `fcLabel` and `fcUnlabel` methods.

D.11.10.1 DECLARATION

```
public class LabelParams
    extends java.lang.Object
```

D.11.10.2 CONSTRUCTORS

- *LabelParams*

```
public LabelParams( int i, boolean consistent )
```

- **Usage**

- * The constructor.

- **Parameters**

- * `i` - This is the value to set for `i`.

- * `consistent` - This is the value to set for `consistent`.

D.11.10.3 METHODS

- *get_consistent*

```
public boolean get_consistent( )
```

- **Usage**

- * Returns the consistent value.

- **Returns** - The consistent value.

- *get_i*

```
public int get_i( )
```

- **Usage**

- * Returns the i value.

- **Returns** - The i value.

- *set_consistent*

```
public void set_consistent( boolean b )
```

- **Usage**

- * Sets consistent to a new value.

- **Parameters**

- * **b** - The value to set consistent to.

- *set_i*

```
public void set_i( int i )
```

- **Usage**

- * Sets `i` to a new value.

- **Parameters**

- * `i` - The value to set `i` to.

D.11.11 CLASS **Log**

This class is a placeholder to simplify the logging of events.

D.11.11.1 DECLARATION

```
public class Log
    extends java.lang.Object
```

D.11.11.2 CONSTRUCTORS

- *Log*

```
public Log( )
```

D.11.11.3 METHODS

- *write*

```
public static void write( int level, java.lang.Object message )
```

- **Usage**

- * Writes the event list to the console.

– **Parameters**

- * `level` - 0 = silent; 1 = warnings; 2 = above + more messages; 10 = everything;
- * `message` - The object/message that should be logged.

D.11.12 CLASS Setup

This class sets up the `CSPPProblem` so we don't have to manually do it each time.

D.11.12.1 DECLARATION

```
public class Setup
    extends java.lang.Object
```

D.11.12.2 CONSTRUCTORS

- *Setup*

```
public Setup( )
```

– **Usage**

- * This is the default constructor. The problem returned is random.
-

- *Setup*

```
public Setup( boolean fromDb )
```

– **Usage**

- * A constructor that sets up a problem. If the problem is not random, the defined year and semester in the configuration file `GTAAPConfig` is retrieved from the database.

– **Parameters**

- * `fromDb` -

– **See Also**

- * `edu.unl.consyslab.gtajava.GTAAPConfig` (in D.9.1, page 193)

- *Setup*

```
public Setup( boolean fromDb, int argYear, int argSemester
)

```

– **Usage**

- * A constructor that sets up a problem. If the problem is not random, the year and semester given is retrieved from the database.

– **Parameters**

- * `fromDb` - If it is `true`, then the problem is not a random problem.
- * `argYear` - The year to use.
- * `argSemester` - The semester to use.

D.11.12.3 METHODS

- *getProblem*

```
public CSPPProblem getProblem( )

```

– **Usage**

- * Returns the problem.

– **Returns** - The problem.

- *setProblem*

```
public void setProblem( edu.unl.consyslab.CSPPProblem  problem
)

```

– **Usage**

* Sets the problem.

– **Parameters**

* `problem` - The problem.

D.11.13 CLASS Utils

This class provides commonly used functions and methods that are globally available for other classes to use. Such functions include database access methods, etc..

D.11.13.1 DECLARATION

```
public class Utils
extends java.lang.Object

```

D.11.13.2 CONSTRUCTORS

- *Utils*

```
public Utils( )

```

D.11.13.3 METHODS

- *ArcConsistency_1*

```
public static CSPPProblem ArcConsistency_1(
edu.unl.consyslab.CSPPProblem argProblem )
```

- **Usage**

- * Performs AC1 on the CSPPProblem.

- **Parameters**

- * `argProblem` - The CSPPProblem to run AC on.

- **Returns** - The arc consistent problem.

- *Assign*

```
public static void Assign( edu.unl.consyslab.CSPPProblem
problem, edu.unl.consyslab.CSPVariable c,
edu.unl.consyslab.CSPValue g )
```

- **Usage**

- * Do an assignment of a value to a variable.

- **Parameters**

- * `problem` - The problem.

- * `c` - The CSPVariable to be assigned the value.

- * `g` - The CSPValue to be assigned to the variable.

- *connectToDB*

```
public static void connectToDB( )
```

– **Usage**

- * Connects to the specified MySQL database defined in `GTAAPConfig`. This method checks if the current host is a CSE machine – if it is, we connect directly to the MySQL server; otherwise, if we are running the GUI, it attempts to establish a SSH after asking for CSE login credentials.

• *deleteJassignments*

```
public static void deleteJassignments( java.lang.String saved-
Name )
```

– **Usage**

- * Deletes `jAssignments` from the database given its `String` name.

– **Parameters**

- * `savedName` - The name of the previously saved assignments to be deleted from the database.

• *doubleToFraction*

```
public static String doubleToFraction( double value )
```

– **Usage**

- * Converts a double load to a `String` fraction. We factor into a small error, `delta`.

– **Parameters**

- * `value` - The double value to convert.

– **Returns** - The `String` representation of this fraction.

- *fetchAvailableYearSemesters*

```
public static Vector fetchAvailableYearSemesters( )
```

- **Usage**

- * This method fetches all the available years and semesters from the database.

- **Returns** - A vector of elements of type `YearSemester` in chronological order. That is, earlier semesters before the current semester.

- *fetchDataCourses*

```
public static void fetchDataCourses( int year, int semester,
edu.unl.consyslab.CSPPProblem problem, java.util.Vector vec,
java.lang.String type )
```

- **Usage**

- * Fetches the course data from the MySQL database.

- **Parameters**

- * `year` - The year that we want to fetch the data for.

- * `semester` - The semester that we want to fetch the data for.

- * `problem` - The `CSPPProblem`.

- * `vec` - The `Vector` to store the `CSPVariables` (courses).

- * `type` - The type of course to load. It can either be *grading*, *lecture*, *lab*, or *recitation*.

- *fetchDataGTA*

```
public static void fetchDataGTA( int year, int semester,
java.util.Vector variable, java.util.Vector vec )
```


– **Usage**

- * Fetches the GTA data from the MySQL database. We must first call the `fetchDataCourses`.

– **Parameters**

- * `year` - The year that we want to fetch the data for.
- * `semester` - The semester that we want to fetch the data for.
- * `variable` - The **Vector** of variables.
- * `vec` - The **Vector** to store the CSPValues (GTAs).

– **See Also**

- * `edu.unl.consyslab.Utils.fetchDataCourses(int year, int semester, CSPProblem problem, Vector vec, String type)`

• *fetchDataPreassignments*

```
public static void fetchDataPreassignments( int year,
int semester, edu.unl.consyslab.CSPProblem problem,
java.util.Vector vec, java.util.Vector staticVec )
```

– **Usage**

- * Fetches the preassigned course data from the MySQL database.

– **Parameters**

- * `year` - The year that we want to fetch the data for.
- * `semester` - The semester that we want to fetch the data for.
- * `problem` - The **CSPProblem**.
- * `vec` - The **Vector** that contains the **CSPVariables** (courses). After running this method, the preassigned variables are moved into `staticVec`
- * `staticVec` - The **Vector** to store the preassigned **CSPVariables**.

- *fetchInitialDomain*

```
public static void fetchInitialDomain( java.util.Vector  variable,  
java.util.Vector  value )
```

- **Usage**

- * Fetches the initial domain for the variables. This method sets the initial domains of all variables to the set of all values.

- **Parameters**

- * **variable** - The Vector of CSPVariables.
 - * **value** - The Vector of CSPValues.

- *fetchJassignmentNames*

```
public static void fetchJassignmentNames( java.util.Vector  
names )
```

- **Usage**

- * Fetch the names of the saved assignments into the Vector **names**.

- **Parameters**

- * **names** - This is the Vector of String values of the names of saved assignments are returned in.

- *fetchJassignments*

```
public static Vector fetchJassignments( java.lang.String  saved-  
Name )
```

- **Usage**

- * Fetches saved assignments from the database. This method puts the saved assignments in the `saved` attribute in `CSPProblem`.

– **Parameters**

- * `savedName` - The saved name of the assignments we would like to retrieve.

- **Returns** - The Vector of `jAssignments` which contain the saved assignments.

- *FilterGTAs*

```
public static void FilterGTAs( edu.unl.consyslab.CSPProblem
problem, edu.unl.consyslab.CSPVariable c,
edu.unl.consyslab.CSPValue g )
```

– **Usage**

- * Propagates unassigned courses.

– **Parameters**

- * `c` - The `CSPVariable` `c`.
- * `g` - The `CSPValue` `g`.

- *getCenterLocation*

```
public static Dimension getCenterLocation( java.awt.Dimension
size )
```

– **Usage**

- * Given the size `Dimension` of an object, we return the `Dimension` of the position for the window so that it is centered.

– **Parameters**

- * `width` - The width of the window to be centered.
 - * `height` - The height of the window to be centered.
 - **Returns** - The Dimension of the position.
-

- *getCenterLocation*

```
public static Dimension getCenterLocation( int width, int
height )
```

- **Usage**

- * Given the width and height of an object, we return the Dimension of the position for the window so that it is centered.

- **Parameters**

- * `width` - The width of the window to be centered.
- * `height` - The height of the window to be centered.

- **Returns** - The Dimension of the position.

- *getCourseFromVariable*

```
public static Course getCourseFromVariable( java.lang.Object
var )
```

- **Usage**

- * Gets the associated course from a given variable.
-

- *getScreenSize*

```
public static Dimension getScreenSize( )
```

- **Usage**

- * Gets the screen dimension.

– **Returns** - The Dimension for the screen size.

- *loadDBDriver*

```
public static void loadDBDriver( )
```

– **Usage**

* Loads the JDBC-MySQL driver so that we can use it.

- *NodeConsistent*

```
public static CSPPProblem NodeConsistent(
edu.unl.consyslab.CSPPProblem  problem )
```

– **Usage**

* Do node consistency on the problem.

– **Parameters**

* *problem* - The problem to run node consistency on.

– **Returns** - The node consistent problem.

- *Revise*

```
public static boolean Revise( edu.unl.consyslab.CSPPProblem
problem, edu.unl.consyslab.CSPVariable  ci,
edu.unl.consyslab.CSPVariable  cj )
```

– **Usage**

* Performs revise on the problem given two CSPVariables.

– **Parameters**

* *problem* - The problem.

* *ci* - CSPVariable *ci*.

* `cjCSPVariable` - `cj`.

- *saveJassignments*

```
public static void saveJassignments( edu.unl.consyslab.CSPPProblem
problem, java.lang.String  savedName )
```

- **Usage**

- * Saves the currently assigned variables given the `CSPPProblem` parameter to the database and a name.

- **Parameters**

- * `problem` - The `CSPPProblem`.
- * `savedName` - The name to save as.

- *setupConstraints_capacity*

```
public static void setupConstraints_capacity(
edu.unl.consyslab.CSPPProblem  problem )
```

- **Usage**

- * Sets the `CSPPProblem` hashtable entry for 'constraint' to the appropriate value.

capacity: Constraint that specifies that a gta may not have a load more than the specified maximum load. type: non-binary

- **Parameters**

- * `CSPPProblem` - `problem`

- **Returns** - void

- *setupConstraints_certification*

```
public static void setupConstraints_certification(
edu.unl.consyslab.CSPPProblem problem )
```

- **Usage**

- * Sets the CSPPProblem hashtable entry for 'constraint' to the appropriate value.

certification: Constraint specifying that GTAs must be ITA certified or English-speaking to teach recitations or labs. type: unary

- **Parameters**

- * CSPPProblem - problem

- **Returns** - void

- *setupConstraints_diffa*

```
public void setupConstraints_diffa( edu.unl.consyslab.CSPPProblem
problem, int year, int semester )
```

- **Usage**

- * Sets the CSPPProblem hashtable entry for 'constraint' to the appropriate value.

diffa: Binary constraints specifying that no gta may be assigned to two lab or recitation courses with overlapping times. type: non-binary

- **Parameters**

- * CSPPProblem - problem
- * int - year

* `int` - semester

– **Returns** - void

• *setupConstraints_equality*

```
public static void setupConstraints_equality(
edu.unl.consyslab.CSPPProblem problem, int year, int semester
)
```

– **Usage**

* Sets the CSPPProblem hashtable entry for 'constraint' to the appropriate value.

equality: Constraint that specifies that a set of courses must have the same GTA assigned to it type: non-binary

– **Parameters**

* CSPPProblem - problem

* `int` - year

* `int` - semester

– **Returns** - void

• *setupConstraints_mutex*

```
public static void setupConstraints_mutex(
edu.unl.consyslab.CSPPProblem problem )
```

– **Usage**

* Sets the CSPPProblem hashtable entry for 'constraint' to the appropriate value.

mutex: Mutex constraints 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/she is instructing. type: binary

– **Parameters**

- * `CSPPProblem` - problem
- * `int` - year
- * `int` - semester

– **Returns** - void

• *setupConstraints_nilPref*

```
public static void setupConstraints_nilPref(
edu.unl.consyslab.CSPPProblem  problem )
```

– **Usage**

- * Sets the CSPPProblem hashtable entry for 'constraint' to the appropriate value.

nilPref: Unary constraint that eliminates GTAs who have 0 preference for the constrained course from the domain the variable. type: unary

– **Parameters**

- * `CSPPProblem` - problem

– **Returns** - void

• *setupConstraints_overlap*

```
public static void setupConstraints_overlap(
```

`edu.unl.consyslab.CSPPProblem problem)`

– **Usage**

- * Sets the CSPPProblem hashtable entry for 'constraint' to the appropriate value.

overlap: Unary constraint that specifies that a gta may not teach a lab or recitation that overlaps in time with a course currently being enrolled in. type: unary

– **Parameters**

- * CSPPProblem - problem

– **Returns** - void

• *setupConstraints_takingCourse*

`public static void setupConstraints_takingCourse(
edu.unl.consyslab.CSPPProblem problem)`

– **Usage**

- * Sets the CSPPProblem hashtable entry for 'constraint' to the appropriate value.

takingCourse: Constraint specifying that GTAs is not currently enrolled in a course type: unary

– **Parameters**

- * CSPPProblem - problem

– **Returns** - void

- *setupConstraints*

```
public static void setupConstraints( edu.unl.consyslab.CSPProblem
problem, int year, int semester )
```

- **Usage**

- * Sets the CSPProblem hashtable entry for 'constraint' to the appropriate values by calling sub functions for all the constraint types.

- **Parameters**

- * CSPProblem - problem The CSP problem.
- * int - year The year.
- * int - semester The semester.

- *Unassign*

```
public static void Unassign( edu.unl.consyslab.CSPProblem
problem, edu.unl.consyslab.CSPVariable c )
```

- **Usage**

- * Do an unassign for a given CSPVariable

- **Parameters**

- * problem - The problem.
- * c - Unassign the CSPVariable c.

Bibliography

- [Garcia-Molina *et al.*, 2002] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems, The Complete Book*. Prentice Hall, 2002.
- [Glaubius, 2001] Robert Glaubius. A Constraint Processing Approach to Assigning Graduate Teaching Assistants to Courses. Undergraduate Honors Thesis. Department of Computer Science and Engineering, University of Nebraska-Lincoln, 2001.
- [Guddeti and Choueiry, 2004] Venkata Praveen Guddeti and Berthe Y. Choueiry. An Empirical Study of a New Restart Strategy for Randomized Backtrack Search. In *Workshop on Immediate Applications of CSPs (CP 04)*, pages 66–82, Toronto, Canada, 2004.
- [Guddeti and Choueiry, 2005] Praveen Guddeti and Berthe Y. Choueiry. Characterization of a New Restart Strategy for Randomized Backtrack Search. In Boi Faltings *et al.*, editor, *Recent Advances in Constraints*, volume 3419 of *Lecture Notes in Artificial Intelligence*, pages 56–70. Springer, 2005.
- [Guddeti, 2004a] Venkata Praveen Guddeti. A Dynamic Restart Strategy for Randomized BT Search. In Mark Wallace, editor, *Proceedings of 10th International Conference on Principle and Practice of Constraint Programming (CP 04)*, volume 3258 of *Lecture Notes in Computer Science*, page 796, Toronto, Canada, 2004. Springer Verlag.

- [Guddeti, 2004b] Venkata Praveen Reddy Guddeti. An Improved Restart Strategy for Randomized Backtrack Search. Master's thesis, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, December 2004. December 2004.
- [Lim *et al.*, 2004a] Ryan Lim, Venkata Praveen Guddeti, and Berthe Y. Choueiry. A Constraint-Based System for Hiring and Managing Graduate Teaching Assistants. In Mark Wallace, editor, *Proceedings of the 10th International Conference on Principle and Practice of Constraint Programming (CP 04)*, volume 3258 of *Lecture Notes in Computer Science*, page 817, Toronto, Canada, 2004. Springer Verlag. <http://www.springerlink.com/index/DHCD3DFK8EMR8Y2X>.
- [Lim *et al.*, 2004b] Ryan Lim, Venkata Praveen Guddeti, and Berthe Y. Choueiry. An Interactive System for Hiring and Managing Graduate Teaching Assistants. In *Conference on Prestigious Applications of Intelligent Systems (ECAI 04)*, pages 730–734, Valencia, Spain, 2004.
- [Reichenbach, 2001] Steve Reichenbach. Department Chair, Personal Communication, 2001.
- [Thota, 2004] Venkateshwar Rao Thota. Online Interactive Problem-Solving. Masters Project. Department of Computer Science & Engineering, University of Nebraska-Lincoln, 2004.
- [Zou and Choueiry, 2003a] Hui Zou and Berthe Y. Choueiry. Characterizing the Behavior of a Multi-Agent Search by Using it to Solve a Tight, Real-World Resource Allocation Problem. In *Workshop on Applications of Constraint Programming*, pages 81–101, Kinsale, County Cork, Ireland, 2003.

[Zou and Choueiry, 2003b] Hui Zou and Berthe Y. Choueiry. Multi-agent Based Search versus Local Search and Backtrack Search for Solving Tight CSPs: A Practical Case Study. In *Working Notes of the Workshop on Stochastic Search Algorithms (IJCAI 03)*, pages 17–24, Acapulco, Mexico, 2003.

[Zou, 2003] Hui Zou. Iterative Improvement Techniques for Solving Tight Constraint Satisfaction Problems. Master’s thesis, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, December 2003.