ONLINE INTERACTIVE PROBLEM-SOLVING

by

Venkateshwar Rao Thota

A PROJECT

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Berthe Y. Choueiry

Lincoln, Nebraska

December, 2004

ONLINE INTERACTIVE PROBLEM-SOLVING

Venkateshwar Rao Thota, M.S.

University of Nebraska, 2004

Advisor: Berthe Y. Choueiry

With the advent of online computer systems, we now have a new dimension in problem-solving capabilities. Instead of simply using the power of a computer to achieve fast turnaround, we can develop interactive systems that are user-friendly and capable of integrating the human user into the decision loop and exploiting his/her insight for problem solving. Interactive problem-solving can thus be defined as a process in which a computer and a user work side-by-side to define, analyze and solve a problem.

In this report,we describe a system we built for interactively solving the Graduate Teaching Assistant Assignment Problem (GTAAP). We have developed a constraint-based system that is operated through a web-based visual interface. The system provides the manager the ability to assign GTAs to various departmental tasks by taking various constraints into consideration. The decision-making system uses consistency algorithms to maintain the overall consistency of the problem.

The system has drastically reduced the number of conflicting decisions, increased the quality of the process and solution, and decreased the amount of time and effort spent on generating the assignment.

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor, Dr. Berthe Y. Choueiry, whose valuable contributions and insight made this project a reality. I am grateful for her constant support and patience during my research. I would like to thank Dr. Hong Jiang, Dr. Fred Choobineh, and Dr. Steven Dunbar for being in my committee.

I also thank my colleagues, the members of Constraint Systems Laboratory, especially Ryan Lim and Praveen Guddeti for their co-operation and help in making me understand the GTA project.

I would also like to thank Ms. Deborah Derrick, who checked my report and provided feedback on English.

I am greatly indebted to my parents, Mr. Soma Narayana Thota and Mrs. Lalitha Thota for motivating towards Masters degree. I thank all my friends for their help during my masters study.

I dedicate this project to my parents, Soma Narayana and Lalitha, who supported and encouraged me during my masters study and also throughout my career.

Contents

1 Introduction **1**
1.1 GTA assignment problem . . . 2
1.2 Problem-solving strategies . . . 2
1.2.1 General setting . . . 2
1.2.2 The context of the GTAAP . . . 4
1.3 Definitions . . . 5
1.4 GTAAP modeled as a CSP . . . 7
1.5 Interactive system . . . 9
1.6 Organization of the report . . . 11

2 Methodology **12**
2.1 Overall methodology . . . 13
2.2 System initialization . . . 14
2.2.1 Node-Consistency algorithm . . . 15
2.2.2 Arc-consistency algorithm . . . 16
2.3 Assignment of GTAs to classes . . . 18
2.3.1 Assigning a GTA . . . 18
2.3.2 Unassigning a GTA . . . 19
2.4 Assignment of classes to GTAs . . . 21

3 Design and implementation **23**
3.1 GTA assignment project . . . 25
3.2 Interactive system . . . 28
3.3 Interface for interactive selections . . . 29
3.3.1 Driver for user's actions . . . 30
3.3.2 Assignment of GTAs to classes . . . 32
3.3.3 Assignment of classes to GTAs . . . 39
3.3.4 Other features . . . 41
3.4 Interactive solver . . . 45
3.4.1 Socket listener . . . 46
3.4.2 Consistency algorithms . . . 48

4 Conclusion and future work **49**

Bibliography **51**

A File and Data structures **54**
A.1 File structure 54
A.2 Data structures 56
A.2.1 Assigned Courses 56
A.2.2 Log of Assignments/Unassignments 57
A.2.3 Communication with PHP scripts 57
A.3 Variables and functions 58
A.3.1 ac.lisp 58
A.3.2 loadgta.lisp 58
A.3.3 misc-funcs.lisp 59
A.3.4 savecsp.lisp 59
A.3.5 showgtas.lisp 60
A.3.6 showcourses.lisp 61
A.3.7 sock-listener.lisp 62
A.3.8 Function calls 62

B Communication between the web interface and the interactive solver **66**

List of Figures

1.1 Problem-solving techniques: batch versus interactive. 3
1.2 A CSP represented as a graph. 7
1.3 Web interface of interactive problem-solving. 10

2.1 Overall Methodology. 13

3.1 GTA assignment project system architecture [Lim *et al.*, 2004a]. 25
3.2 Global view of the architecture of the interactive system in GTAAP. . . . 26
3.3 Detailed view of the architecture of the interactive system. 28
3.4 Possible actions for the manager under 'Interactive Selections.' 30
3.5 Snapshot of perspective 'Assign GTAs to Classes.' 32
3.6 Domains of courses when assigning. 34
3.7 An example of a course and a list of consistent GTAs present in its domain. 35
3.8 Assignment of GTAs to Classes. 36
3.9 Options menu. 38
3.10 Snapshot of perspective 'Assign Classes to GTAs.' 39
3.11 An example of a GTA and a list of courses in which he/she is present. . . 40
3.12 Dual perspective for decision making. 41
3.13 Drop-down menu to save/retrieve assignments. 43

A.1 Function: ac-1. 62
A.2 Function: assign-gta. 63
A.3 Function: check-assignment. 63
A.4 Function: check-year-sem. 63
A.5 Function: clear-saved-assignments 63
A.6 Function: display-gtas 64
A.7 Function: display-courses 64
A.8 Function: load-assigned-courses 65
A.9 Function: load-initialize-csp 65
A.10 Function: make-current 65
A.11 Function: save-current-csp 65
A.12 Function: start-server 65
A.13 Function: do-command 65

B.1 Conversion of manager's actions into Lisp function calls. 68

List of Tables

3.1 Database table where names of alternative scenarios are stored. 42
3.2 Database table for storing the details of an alternative scenario. 42
3.3 Lisp commands and their description. 47

B.1 Global variables in Lisp environment. 68

Chapter 1

Introduction

The early 1990s saw a drastic increase in the development and deployment of interactive decision-support systems. These programs are designed to take advantage of the experience and intuition of the human user to solve problems. Some of the systems attempt to maintain generality and remain applicable across domain areas. Others focus on restricted problem area.

The Graduate Teaching Assistant Assignment Project is one such dedicated system for solving the Graduate Teaching Assistant Assignment Problem (GTAAP). The project is being developed at the Constraint Systems Laboratory of the Department of Computer Science and Engineering (CSE) of the University of Nebraska-Lincoln (UNL). One of the main features of this project is interactive problem-solving, which allows a human manager to interactively assign GTAs to various departmental tasks by taking various constraints into consideration.

In this chapter, we give a short description of the GTAAP and the motivating reasons for developing an interactive system. We recall the definitions and related terms, and summarize our contributions.

1.1 GTA assignment problem

As stated by Glaubius [2001]: "The GTA assignment problem can be described as the task of assigning GTAs to courses, based on their qualifications, availability, and preferences, to academic courses in a semester for jobs such as grading, supervising labs and recitations, and teaching introductory classes." Typically every semester, a group of 25 to 40 GTAs must be assigned to 55 to 70 courses. The problem is often tight and sometimes over-constrained. Given its size and the variety of constraints, the problem is difficult to solve manually and usually involves various staff and faculty. The process is tedious and error-prone, and the results tend to be less than satisfactory [Glaubius and Choueiry, 2002c].

1.2 Problem-solving strategies

In this section, we explain the motivation for using interactive techniques for solving the GTAAP. We explain, in general, the deficiencies involved in using automated solvers and, more specifically, the difficulties in using them for solving the GTAAP.

1.2.1 General setting

Problem-solving techniques can be broadly divided into batch processing and interactive processing [Kopfer and Schnberger, 2002]. The algorithmic flow involved in the two techniques is shown in Figure 1.1.

In batch processing, a series of non-interactive jobs are executed all at one time. An automatic problem solver is used to find a solution to the given problem. The underlying algorithm used in the solver searches for a solution until it reaches a pre-defined termination criterion. Although most real-world applications use automatic problem-solving techniques, they have some deficiencies [Pu and Faltings, 2002]:

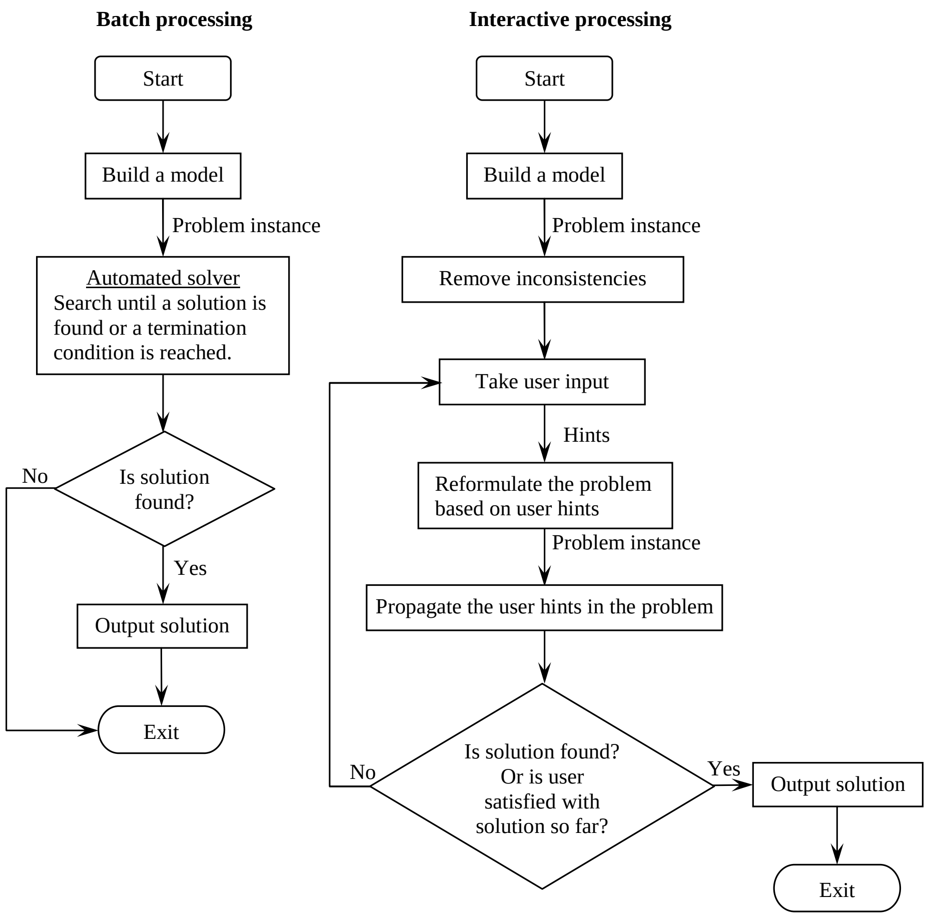


Figure 1.1: Problem-solving techniques: batch versus interactive.

- The process of finding one or more solution cannot be manipulated after the algorithm has been started because new information cannot be propagated to the solving process. The search process and its direction cannot be influenced once the automatic search is started.

- In some complex problems, the search algorithms may not find a solution or cannot find a solution in a reasonable amount of time. As the search proceeds, a user might be able to provide hints for guiding the search. However, the underlying design of these search mechanisms does not allow the user to add his/her input during the search. The user can incorporate or implement his/her hints and ideas, as additional constraints on the initial problem formulation, only after the algorithm terminates.

- A complete problem formulation, which is the basic assumption of an automatic search processes, may be tedious and error-prone for some problems, where constraints are subjective and ill-formulated.

In any case, the user has to take the solution that has been generated by the algorithm as a basis for further modifications and adaptations, and is not supported by the search algorithms during this difficult process.

The second approach, used to overcome the problems caused by batch processing, is interactive processing. In this case, modifications done by the human user are recognized. The interactive algorithm continues its computational task while taking into consideration the additional information and the hints given by the user. This process continues until a solution is found or the user is satisfied with the solution obtained so far. Interactive problem solvers are unique because they combine the human's problem knowledge and intuition for creative solving with the computational power of a computer that can handle a large number of constraints and large amount of data.

1.2.2 The context of the GTAAP

Various automated search algorithms have been developed and tested for solving the GTAAP [Lim *et al.*, 2004a]. The behavior and performance of these search strategies with different characterizations are documented in [Zou and Choueiry, 2003a; 2003b; Zou, 2003]. The search strategies that are developed are (deterministic) backtrack (BT) search with various ordering heuristics [Glaubius and Choueiry, 2002a], a local search [Zou and Choueiry, 2003a], a multi-agent based search [Zou and Choueiry, 2003a], and a randomized backtrack search with a new restart strategy [Guddeti and Choueiry, 2004]. On average, the GTAAP usually consists of sixty variables and approximately thirty values per variable. Thus, the search space (30^{60} possible combinations with only a few valid solutions [Glaubius and Choueiry, 2002b]) of the problem is large and general search algorithms do not terminate within a reasonable amount of time. This is evident from the performance of BT, even though the algorithm is theoretically sound and complete, as discussed in [Guddeti and Choueiry, 2004]. The performance of local search is affected by local optima [Zou and Choueiry, 2003a] and multi-agent-based search by deadlocks when the problem is over-constrained [Zou and Choueiry, 2003a]. Multi-agent search is consistently the only technique capable of solving tight but solvable problems.

The interactive processing of GTAAP strives to avoid these difficulties by involving human intuition into the system. The manager, with his/her experience in solving the problem, assigns GTAs to courses while the system continues to optimize the problem and provides feedback to the manager. The system provides the manager the ability to test and verify various scenarios until he/she finds a suitable assignment to all the courses. A manager can identify various sources of conflicts that may not be apparent to an automatic solver, and take appropriate steps. As a result, the process decreases the overall amount of time and effort spent on making the assignment by avoiding the generation of solutions that are not acceptable in practice.

1.3 Definitions

The GTAAP is a multi-criteria optimization problem for which we have developed a constraint-based model. We give a brief definition of the Constraint Satisfaction Problem (CSP) and related terms in this section.

Definition 1. *Constraint:* A constraint is a logical relation among several unknowns (or variables), each taking a value in a given domain (e.g., A+B=C and 'the circle is inside the square') [Barták, 1998].

Definition 2. *Constraint satisfaction problem (CSP):* Mathematically, a CSP is defined as follows:

Given: $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$, where

- $\mathcal{V}$: a set of variables

  $\mathcal{V} = \{V_1, V_2, \dots, V_n\}$

- $\mathcal{D}$: a set of variable domains (domain values)

  $\mathcal{D} = \{D_{V_1}, D_{V_2}, \dots, D_{V_n}\}$

  such that D_{V_i} is the domain of variable V_i

- $\mathcal{C}$: a set of constraints

  $\mathcal{C} = \{C_{V_1}, C_{V_2}, \dots, C_{V_i,V_j,\dots,V_k}, \dots, C_{V_n}\}$

  such that $C_{V_i,V_j,\dots,V_k} = \{(x, y, \dots, z)/(x \in D_{V_i}) \wedge (y \in D_{V_j}) \wedge \dots (x \in D_{V_k})\} \subseteq D_{V_i} \times D_{V_j} \times \dots \times D_{V_k}$

  and $C_{V_i,V_j,\dots,V_k}$ is a constraint between variables $V_i, V_j, \dots, V_k$.

Query: Find a value for each variable from its domain such that all the constraints are satisfied.

Informally, a CSP is a problem where one must choose values to a given set of decision variables that satisfy a given set of constraints or criteria.

For any given constraint $C_{V_i,V_j,\dots,V_k}$ the set of variables $V_i, V_j, \dots, V_k$ is called the constraint's *scope* and the size of this set is the constraint's *arity*. If the arity of a constraint is one, then it is called a *unary constraint*. If the arity is two then it is called a *binary constraint*.

A CSP can be represented as an undirected graph with nodes representing the variables and edges representing constraints (see Figure 1.2). The edges connect

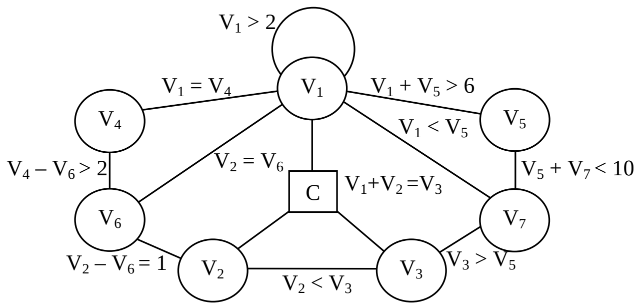


Figure 1.2: A CSP represented as a graph.

variables that share a constraint. A unary constraint is represented by an arc originating and terminating at the same node. If a constraint involves more than two variables (non-binary constraint), then a new type of node is generated to represent the non-binary constraint and is linked to the nodes of the variables in the scope of the constraint. Figure 1.2 shows a unary constraint $V_1 > 2$ on node V_1, a binary constraint $V_2 < V_3$ between nodes V_2 and V_3, and a non-binary constraint $V_1 + V_2 = V_3$, represented by a separate node C, between nodes V_1, V_2, and V_3.

1.4 GTAAP modeled as a CSP

The GTAAP can naturally be formulated as a CSP [Glaubius and Choueiry, 2002c]. Courses are variables whose domains consist of the available GTAs. Constraints in the system are based on the practical constraints drawn out from descriptions provided by the CSE department. In this section, we list the variables, values, and constraints in the problem. More details can be found in [Glaubius and Choueiry, 2002a].

The GTAAP as a CSP: In a given semester, given a set of GTAs, a set of courses, and a set of constraints on allowable assignments, find an assignment of GTAs to courses that is:

- *Consistent*: the assignment breaks no constraints.
- *Satisfactory*: maximize the number of courses covered (first) and the satisfaction of the assigned GTAs (second).

Courses: Courses are modeled as variables in the CSP. There are three types of courses offered: lectures, labs, and recitations. These courses may be offered during the entire semester, or only during the first or last half. Lectures usually require a GTA grader, while labs and recitations require an instructor.

Domains: GTAs make up the domains of the variables. A GTA may serve as an instructor if he/she has International Teaching Assistant (ITA) certification. Each GTA may specify a preference value on a scale of 0 to 5 for each course offered.

Constraints: Three different types of constraints have been formulated: unary, binary, and non-binary constraints:

- *Unary*:
    1. ITA Certification: A GTA must be ITA certified to teach the constrained course.
    2. Enrollment: A GTA cannot be assigned to a course in which he/she is enrolled.
    3. Overlap: A GTA cannot be assigned to a course that requires an instructor if he/she is enrolled in a course at the same time.
    4. Zero preference: A GTA cannot be assigned to a course for which he/she has zero preference. The GTA gives a zero preference to a course if he/she has justification (e.g., currently enrolled or time conflicts).

- *Binary*:

  1. Mutex: Courses cannot be assigned the same GTA.

- *Non-Binary*:

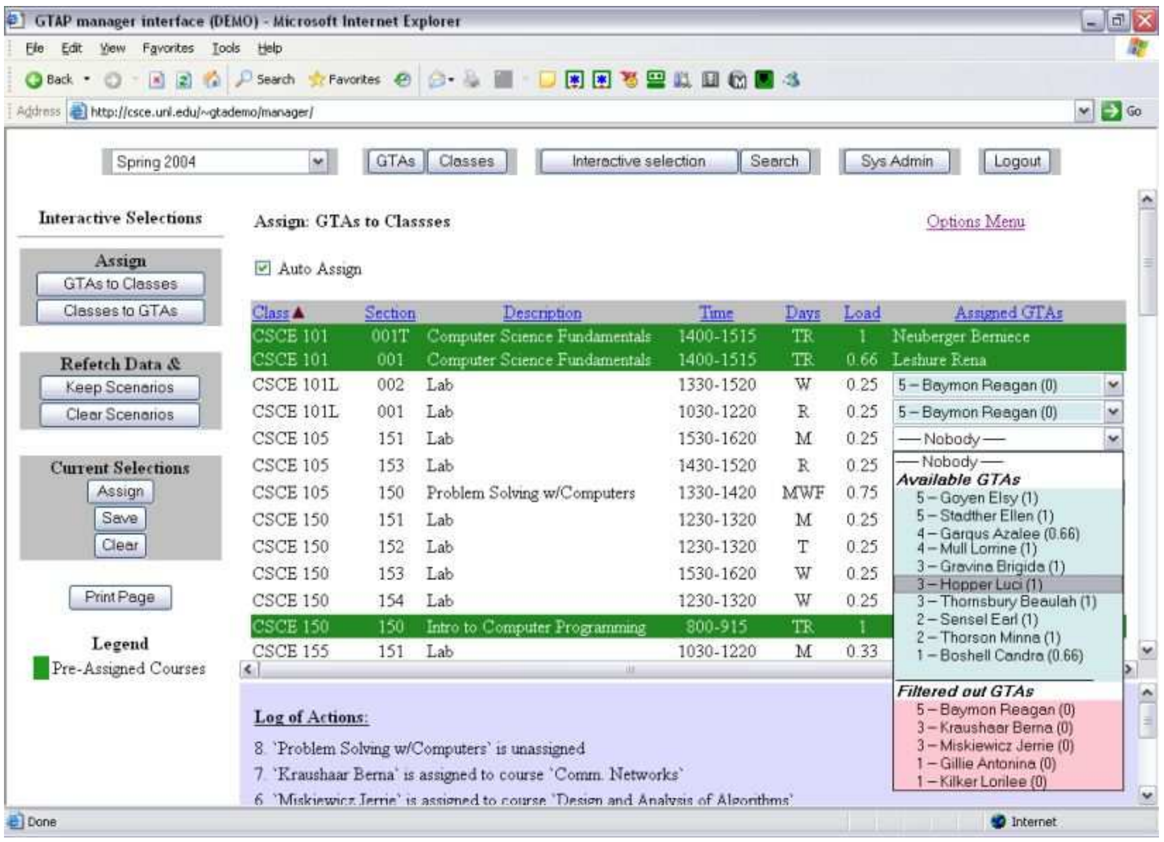
  1. Equality: all courses should be assigned the same GTA.
  2. Capacity: no GTA should be assigned a workload that exceeds his/her capacity.
  3. Confinement: assignments to two specific sets of courses should be mutually exclusive.

As a part of the GTAAP, a web interface was developed to simplify the collection of data and specification of constraints [Lim *et al.*, 2004a]. As discussed in Section 1.2, a number of algorithms have been implemented to assist the human manager in generating solutions automatically [Lim *et al.*, 2004b].

1.5 Interactive system

We built an online interactive system that assists a manager in solving a GTAAP instance. This system allows the manager to interactively assign GTAs to courses and visualize how the assignment affects the GTAs available for other courses. The interactive functionality is one of the main features of the system deployed at CSE and uses the GTA and course data collected via the web. A snapshot of the web interface is shown in Figure 1.3.

A manager can view the list of GTAs for any course at any time and take appropriate action. He/she can visually see the lists getting filtered, which allows him/her to take the appropriate actions. When the manager makes an assignment on the web


GTAP manager interface (DEMO) - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://csce.unl.edu/~gtademo/manager/
Go
Spring 2004
GTAs Classes Interactive selection Search Sys Admin Logout
Interactive Selections
Assign
GTAs to Classes
Classes to GTAs
Refetch Data &
Keep Scenarios
Clear Scenarios
Current Selections
Assign
Save
Clear
Print Page
Legend
Done
Internet

🟩 Pre-Assigned Courses

Assign: GTAs to Classses

Options Menu

☑ Auto Assign

Class ▲	Section	Description	Time	Days	Load	Assigned GTAs
CSCE 101	001T	Computer Science Fundamentals	1400-1515	TR	1	Neuberger Bernece
CSCE 101	001	Computer Science Fundamentals	1400-1515	TR	0.66	Leshure Rena
CSCE 101L	002	Lab	1330-1520	W	0.25	5 – Baymon Reagan (0)
CSCE 101L	001	Lab	1030-1220	R	0.25	5 – Baymon Reagan (0)
CSCE 105	151	Lab	1530-1620	M	0.25	— Nobody —
CSCE 105	153	Lab	1430-1520	R	0.25	
CSCE 105	150	Problem Solving w/Computers	1330-1420	MWF	0.75	
CSCE 150	151	Lab	1230-1320	M	0.25	
CSCE 150	152	Lab	1230-1320	T	0.25	
CSCE 150	153	Lab	1530-1620	W	0.25	
CSCE 150	154	Lab	1230-1320	W	0.25	
CSCE 150	150	Intro to Computer Programming	800-915	TR	1	
CSCE 155	151	Lab	1030-1220	M	0.33	

— Nobody —
Available GTAs
5 – Goyen Elsy (1)
5 – Stadther Ellen (1)
4 – Garqus Azalee (0.66)
4 – Mull Lorrine (1)
3 – Gravina Brigida (1)
3 – Hopper Luci (1)
3 – Thornsbury Beaulah (1)
2 – Sensel Earl (1)
2 – Thorson Minna (1)
1 – Boshell Candra (0.66)
Filtered out GTAs
5 – Baymon Reagan (0)
3 – Kraushaar Berna (0)
3 – Miskiewicz Jerrie (0)
1 – Gillie Antonina (0)
1 – Kilker Lorilee (0)

Log of Actions:

8. 'Problem Solving w/Computers' is unassigned
7. 'Kraushaar Berna' is assigned to course 'Comm. Networks'
6. 'Miskiewicz Jerrie' is assigned to course 'Design and Analysis of Algorithms'

Figure 1.3: Web interface of interactive problem-solving.

interface, the system checks all the constraints, and then removes from the domains of the 'unassigned' courses the GTAs that can no longer be assigned to them given the decision made. The manager can relax or impose new constraints. In case the problem cannot be solved, the manager can see the requirement to hire new GTAs and distribute the course load. Alternatively, a manager can also see if there are more GTAs than required and eliminate some of them.

There are several advantages for having the system online. The system is instantly available at anytime, anywhere. Because our system is evolving, the manager always has access to the latest developments. The system runs on a web server. Any updates and upgrades are implemented directly through the server.

The system runs on the department's Unix server (i.e., `cse.unl.edu`). However, it is accessible to anyone running a web browser on any operating system. The browser code, in HTML and JavaScript, is governed by standards organizations, which ensures code compatibility with current and future browsers.

1.6 Organization of the report

The rest of the report is organized as follows. In Chapter 2, we present the methodology of interactive problem-solving. Chapter 3 provides the detailed design and implementation of the system. Chapter 4 summarizes the project with our conclusions and provides directions for future improvements. Appendix A explains the files and data structures involved in the project and Appendix B explains the communication between the visual interface and the interactive solver.

Chapter 2

Methodology

In this chapter, we give a detailed description of the methodology that facilitates interactive problem-solving. We also discuss the consistency algorithms that are required to maintain a consistent problem.

The system gives the manager the ability to do the interactive assignments from two perspectives

1. "Assignment of GTAs to Classes" and

2. "Assignment of Classes to GTAs"

Since both perspectives are different views of the same CSP formulation, an assignment in one perspective is immediately reflected in another perspective. This gives the manager more flexibility to comprehend and solve the problem. The consistency algorithms implemented in the system ensure that the problem is consistent. In the following sections, we first describe the overall methodology, the consistency algorithms that are involved, and finally the assignment and unassignment procedures in the two perspectives.

2.1 Overall methodology

As explained in Section 1.2, the CSP is solved based on user interactions. The interactions in the context of the GTAAP are the assignment and unassignment of GTAs to courses. We use consistency algorithms to provide the user with consistent choices only and prevent him/her from making incorrect decisions, thus facilitating user's task.

Figure 2.1 shows the overall flow of operations. The methodology can be further

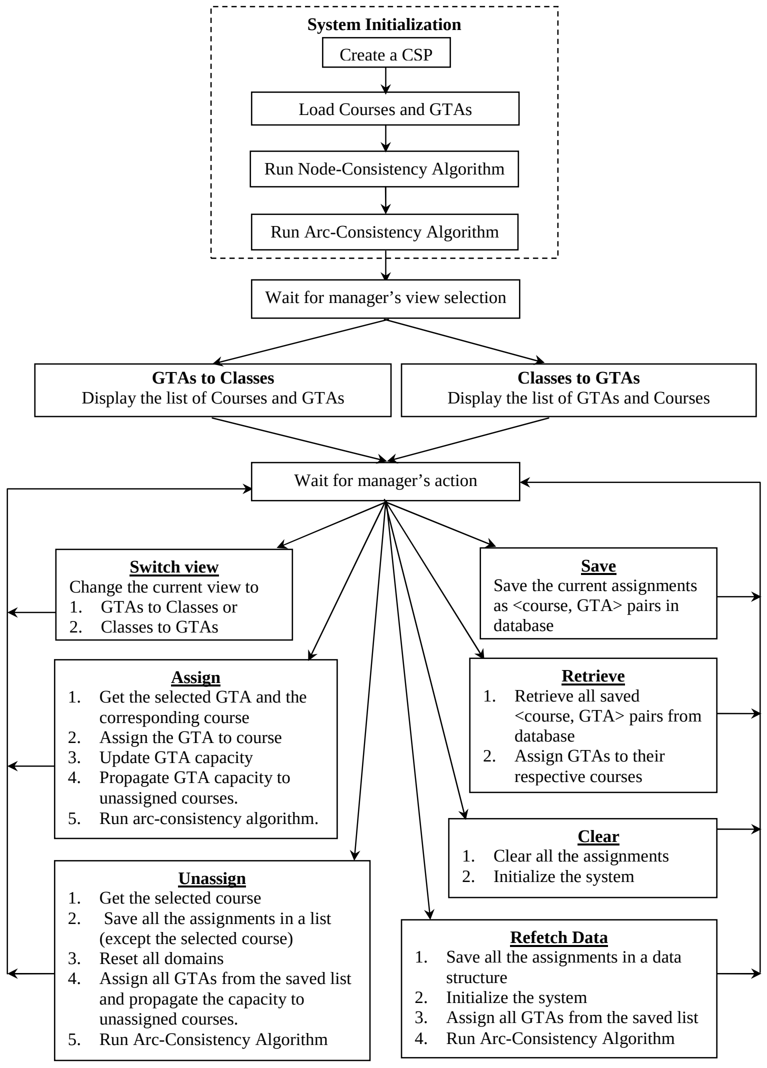

System Initialization
Create a CSP
Load Courses and GTAs
Run Node-Consistency Algorithm
Run Arc-Consistency Algorithm
Wait for manager's view selection
GTAs to Classes
Display the list of Courses and GTAs
Classes to GTAs
Display the list of GTAs and Courses
Wait for manager's action
Switch view
Change the current view to
1. GTAs to Classes or
2. Classes to GTAs
Save
Save the current assignments as <course, GTA> pairs in database
Assign
1. Get the selected GTA and the corresponding course
2. Assign the GTA to course
3. Update GTA capacity
4. Propagate GTA capacity to unassigned courses.
5. Run arc-consistency algorithm.
Retrieve
1. Retrieve all saved <course, GTA> pairs from database
2. Assign GTAs to their respective courses
Clear
1. Clear all the assignments
2. Initialize the system
Unassign
1. Get the selected course
2. Save all the assignments in a list (except the selected course)
3. Reset all domains
4. Assign all GTAs from the saved list and propagate the capacity to unassigned courses.
5. Run Arc-Consistency Algorithm
Refetch Data
1. Save all the assignments in a data structure
2. Initialize the system
3. Assign all GTAs from the saved list
4. Run Arc-Consistency Algorithm

Figure 2.1: Overall Methodology.

divided into two phases, the initialization phase and problem-solving phase. During the initialization phase, the system encodes the problem definition. During the problem-solving phase, the manager can do interactive assignments from either of the two perspectives. Each of the two perspectives has options to assign, unassign, save, or clear the assignments. Since the system is event-driven (i.e., responds to users actions such as mouse clicks and key strokes), it executes the user's command and waits for the next command. The manager can switch between the two perspectives anytime. We give a detailed description of each of the individual component in the following sections.

2.2 System initialization

During the initialization process, a CSP is created using the course and the GTA information and the consistency algorithms are then run on it. The course and GTA information is loaded from the GTAAP database. The CSP is encoded as explained in Section 1.4. The node-consistency and arc-consistency algorithms are run on the CSP to eliminate inconsistent GTAs present in course domains. These algorithms ensure that the manager is always presented with a consistent set of courses and GTAs so that the manager can only do a valid and consistent assignment. We summarize the initialization steps in **Procedure 1**. To further maintain consistency, these algorithms are run after an assignment (see Section 2.3.1) or unassignment operation (see Section 2.3.2).

Procedure 1 Initialize system

```
1: Create a CSP.
2: Load course and GTA information into the CSP.
3: Run the node-consistency algorithm.
4: Run the arc-consistency algorithm.
```

The consistency algorithms are in general applicable to any CSP. In the following sections, we give a detailed description of the consistency algorithms in the context of the GTAAP.

2.2.1 Node-Consistency algorithm

In a CSP, if the domain D of a variable V contains a value 'a' that does not satisfy all the unary constraints on V, then V cannot be assigned by the value 'a'. Thus, the problem is made node-consistent by removing the values from the domain D of each unassigned variable V that do not satisfy unary constraint on V.

In the GTAAP, if every GTA present in the course domain satisfies the unary constraints (i.e., ITA certification, enrollment, overlap, and zero preference constraints), then the problem is said to be node-consistent. For example, if a course requires ITA certification and a GTA present in the course domain does not have the certification, then the GTA cannot be assigned to the course. Thus, the GTA can be removed from the course domain. Such inconsistencies are eliminated using the node-consistency algorithm shown in **Algorithm 2** [Mackworth and Freuder, 1984]. The worst-case

Algorithm 2 Node-Consistency

Input: a CSP
Output: the node-consistent CSP

```
1: $C \leftarrow \{c_i\}$, the set of all the courses
2: for each $c \in C$ do
3:    for each $g \in D_c$ do
4:       if any unary constraint on $c$ is inconsistent with $g$ then
5:          $D_c \leftarrow D_c \setminus \{g\}$ /* remove $g$ from $D_c$ */
6:       end if
7:    end for
8: end for
```

time complexity of the algorithm is $\mathcal{O}(n.d)$, where, n is the number of courses and

d is the maximum number of GTAs present in a course-domain. **Algorithm 2** is executed every time a new CSP is loaded (i.e., at the initializationphase). It is not called by the assignment **Procedure 5** and unassignment **Procedure 7** procedures.

2.2.2 Arc-consistency algorithm

In a CSP, a constraint (also called an arc when the CSP is represented as a graph) from V_i to V_j is said to be arc-consistent if for every value x in the current domain of V_i there is some value y in the domain of V_j such that $V_i = x$ and $V_j = y$ are permitted by the binary constraint between V_i and V_j. The concept of arc-consistency is directional, i.e. if an arc (V_i, V_j) is consistent, then it does not automatically mean that (V_j, V_i) is also consistent. The V_i can thus be arc-consistent with respect to the constraint between V_i and V_j by deleting those values from the domain of V_i for which there does not exist a value in the domain of D_j that satisfies the binary constraint. Deleting such values does not eliminate any solution of the original CSP.

For a GTAAP to be arc-consistent, all the binary constraints (i.e., Mutex and equality constraints), must themselves be arc-consistent. For example, if two courses c_i and c_j have between them an equality constraint (i.e., the two courses must have the same GTA), then both domains must have the same GTAs. Each GTA g_{c_i} present in the domain of c_i must have a consistent (same) GTA g_{c_j} in c_j such that $g_{c_i} = g_{c_j}$; otherwise the g_{c_i} can be removed from domain of c_i. The filtering algorithm is shown in **Algorithm 3**.

To make every course arc-consistent, it is not sufficient to execute REVISE for each constraint just once. Once REVISE reduces the domain of some course c_i, then the previously revised constraints associated with c_i (i.e., some c_j, c_i) have to be revised again, because some of the members of the domain of c_j may no longer be compatible with the remaining members of the revised domain of c_i. **Algorithm 4**

Algorithm 3 Revise(c_i,c_j)

Input: two courses that share a constraint
Output: true the domain of c_i are modified

```
1: $DELETE \leftarrow false$
2: for each $g_{c_i} \in D_{c_i}$ do
3:   if there is no such $g_{c_j} \in D_{c_j}$ such that $c_i \leftarrow g_{c_i}$ and $c_j \leftarrow g_{c_j}$ are consistent then
4:     $D_{c_i} \leftarrow D_{c_i} \setminus \{g_{c_i}\}$ /* delete $g_{c_i}$ from $D_{c_i}$ */
5:     $DELETE \leftarrow true$
6:   end if
7: end for
8: return $DELETE$
```

does these revisions to make the CSP consistent.

Algorithm 4 AC-1

Input: a CSP
Output: an arc-consistent CSP

```
1: $Q \leftarrow \{(c_i, c_j)$ in constraints of GTA, $i \neq j$ $\}$
2: repeat
3:   $CHANGE \leftarrow false$
4:   for each $(c_i, c_j) \in Q$ do
5:     $CHANGE \leftarrow Revise(c_i, c_j) \vee CHANGE$
6:   end for
7: until not $CHANGE$
```

The worst-case time complexity of the algorithm is $\mathcal{O}(d^3.n.e)$, where d is the maximum domain size, n is the number of variables, and e is the number of constraints. In general, even though the GTAAP is over-constrained, the arc-consistency algorithm runs faster during the interactive assignments. This is because most of the GTAs are removed during the initialization phase and there are fewer course domains that need filtering and updating.

In general, during the execution of AC-1, when the domain of a variable is emptied, the problem is declared unsolvable and execution is stopped. However, here the execution is not stopped because the GTAAP is often an over-constrained problem. In such cases it may not always be possible to find a complete set of assignments to all the course variables. A partial set of assignments is also acceptable.

In the following sections, we provide a detailed description of the assignment and unassignment procedures.

2.3 Assignment of GTAs to classes

For the assignment of GTAs to classes, the web interface displays a list of courses and their corresponding list of consistent GTAs. The manager can choose to assign GTAs to courses or to unassign courses. The following sections describe the two procedures.

2.3.1 Assigning a GTA

Whenever a GTA g is chosen for assignment from the domain of a course c, all the other course domains are updated and the inconsistent GTAs are filtered. **Procedure 5** shows the steps for doing the assignment. Because the user is always presented

Procedure 5 Assign(c, g)

Input: a course and a GTA from its domain

```
1: $Assigned - value(c) \leftarrow g$ /* Assign GTA $g$ to course $c$ */
2: $Capacity(g) \leftarrow Capacity(g) - Load(c)$ /* Update the capacity of the GTA */
3: Filter-GTAs($c, g$) /* Update the domains of other unassigned courses */
4: Call AC-1 /* Run arc-consistency algorithm */
5: **return** *true*
```

with a consistent CSP, he/she can only choose a consistent GTA for a course. The assignment is done in Step 1 and the following steps ensure the consistency of the CSP. The course load is discounted from the total capacity of the GTA in Step 2. In Step 3, the capacity of g is propagated to unassigned courses using the Filter-

GTAs procedure. The arc-consistency algorithm, called in Step 4, ensures that the assignment propagates to unassigned courses. The Filter-GTAs procedure is shown in **Procedure 6**. In Step 1 of the procedure, the set of all the current unassigned

Procedure 6 Filter-GTAs(c, g)

Input: a course and a GTA from its domain

```
1: Let $C \leftarrow \{c_i\}$, the set of all unassigned courses
2: $D_c \leftarrow \{g\}$ /* Keep only the assigned GTA in the course domain */
3: for each $c' \in C$ do
4:    if $(g \in D_{c'}) \wedge (Capacity(g) < Load(c'))$ then
5:       $D_{c'} \leftarrow D_{c'} \setminus \{g\}$ /* delete $g$ from domain of $D_{c'}$ */
6:    end if
7: end for
8: return $true$
```

courses is stored in C. In Step 2, the domain of the course D_c is updated so that it contains only the assigned GTA and all the other consistent GTAs are removed. Because the capacity of the GTA is decreased following the assignment, the GTA should be removed from the domains of the unassigned courses whose load exceeds the remaining capacity of the GTA. Thus, all the other course domains are updated and inconsistent GTAs are filtered in Steps 3 to 7. For each course c' not equal to c, if the GTA g is present in the course domain and the capacity of the g is less than course load of c', then g is removed from the domain of c'.

2.3.2 Unassigning a GTA

Whenever a GTA is unassigned from a course c, all the course domains are updated to reflect the change. **Procedure 7** shows the steps involved. In Step 1, all the other course assignments are saved in a list L as $\langle$course, GTA$\rangle$ pairs except for the course assignment with c. The assigned GTA of the course is obtained in Step 2. In Step 3, the capacity of the GTA is updated by adding the course load.

In Step 4, the course is removed. In Step 5, the course domains are reset to their initial arc-consistent domains, since the domains are previously updated because of the assignments/unassignments made earlier (arc-consistency algorithm is executed every time an assignment is made). In Steps 6 to 9, the courses are re-assigned and their domains are filtered. The course is assigned with its GTA in Step 7 and the capacity of g' is propagated to unassigned courses using the Filter-GTAs procedure. Here, the capacity of the GTA is not discounted from the course load since the capacity is already updated during the assignment previously made. Finally, the arc-consistency algorithm is run in Step 10 to propagate the effect of the assignments on the unassigned variables. Here, we are resetting course domains and redoing all

Procedure 7 Un-assign(c)

Input: course to be unassigned

```
1: $L \leftarrow \{\langle c_i, g_i \rangle\}$, the set of all courses and their assigned GTAs, except course $c$
2: $g \leftarrow Assigned-val(c)$ /* Get the GTA assigned to course*/
3: $Capacity(g) \leftarrow Capacity(g) + Load(c)$ /* Update the capacity of the GTA */
4: $Assigned-value(c) \leftarrow nil$ /* Remove the course assignment */
5: Reset all the course domains to their initial arc-consistent domains
6: for each $\{c', g'\} \in L$ do
7:     $Assigned-value(c') \leftarrow g'$ /* Re-assign the GTA */
8:     Filter-GTAs($c', g'$) /* Update the domains of other unassigned courses */
9: end for
10: Call AC-1 /* Run arc-consistency algorithm */
11: return
```

the assignments instead of doing incremental unassignments [Bessière, 1991]. This is because the data structures that would be necessary to follow such an approach would be heavy even though the CSP data structures for the GTA problem are light.

2.4 Assignment of classes to GTAs

The manager can also interactively assign courses to GTAs. Here the same CSP encoding is viewed from a different perspective. Unlike the "Assignment of GTAs to Classes" perspective, here the manager is presented with a list of GTAs and corresponding set of courses (i.e., the course domain in which the GTA is present). The manager thinks that he/she is assigning a course to a GTA, but in the background, the GTA is selected from the course domain and assigned to the course. Here, since the same CSP is used, it is not necessary to perform initialization steps (see Section 2.2). Whenever the manager chooses this view, the GTAs and their corresponding course lists are obtained by **Procedure 8**. In this procedure, a domain for a GTA g is constructed by going through all the courses $\{c_1, c_2, \ldots, c_n\}$ and checking if the GTA g is present in their domain. For each GTA g, its domain D_g is constructed by including all the courses in which GTA g is present. This done in Steps 3 to 10. Finally, all the GTAs and their domains are displayed in Steps 11 to 13. In this perspective, for

Procedure 8 Display-GTAs

```
 1: $C \leftarrow \{c_i\}$, the set of all courses in the CSP
 2: $G \leftarrow \{g_i\}$, the set of all GTAs in the CSP
 3: for each $g \in G$ do
 4:     for each $c \in C$ do
 5:         /* Create domains for GTAs with courses as the domain elements */
 6:         if $g \in D_c$ then
 7:             $D_g \leftarrow D_g \cup \{c\}$
 8:         end if
 9:     end for
10: end for
11: for each $g \in G$ do
12:     Display $g, D_g$ /* Display the gta, $g$ and its corresponding domain */
13: end for
14: return
```

any assignment or unassignment of courses, the same steps as outlined in Section 2.3 are followed. The only difference is that every time after assignment/unassignment of

GTAs to courses, the GTAs and the corresponding courses are obtained and displayed as explained in **Procedure 8**.

Summary

This chapter reviewed the basic consistency algorithms that are required to maintain a consistent CSP. We also discussed the assignment and unassignment procedures that are required to filter GTAs from course domains.

Chapter 3

Design and implementation

In general, an interactive system for problem solving is built as a three-tier architecture, with:

1. a visual interface for user interaction,
2. a back-end algorithm for problem solving, and
3. a database or file system for storing the data and saving partial or complete solutions.

For better user interaction, the visual interface should be intuitive, easy to learn, and capable of providing feedback on the system status so that users are in control of the problem-solving process [Pu and Faltings, 2002]. The back-end solver algorithm must account for the user's input and integrate it into the problem encoding. The algorithm should also be able to propagate the user decision to prepare the new encoding for another input from user. Some of the other desirable features [Kopfer and Schnberger, 2002] of the system are:

- The ability to present alternative ways of visualizing the problem.

- The ability to show to the human user the consequences of his/her actions.
- The ability to undo previous actions (in the GTAAP, it is variable assignments).

In this chapter we give the design and implementation details of the interactive system, taking the above design principles into consideration. The chapter also explains where the system fits our project. As stated in Chapter 1, interactive assignments or selections is one of the main features of GTA project and uses the GTA and course data collected from the GTA web interface.

We start with a short description of the GTA assignment project system architecture, describe the overall interactive system, and finally the design and implementation details of the visual interface and the interactive solver.

3.1 GTA assignment project

The overall system architecture of GTA assignment project is shown in Figure 3.1 [Lim *et al.*, 2004a]. The current implementation consists of a web interface for data

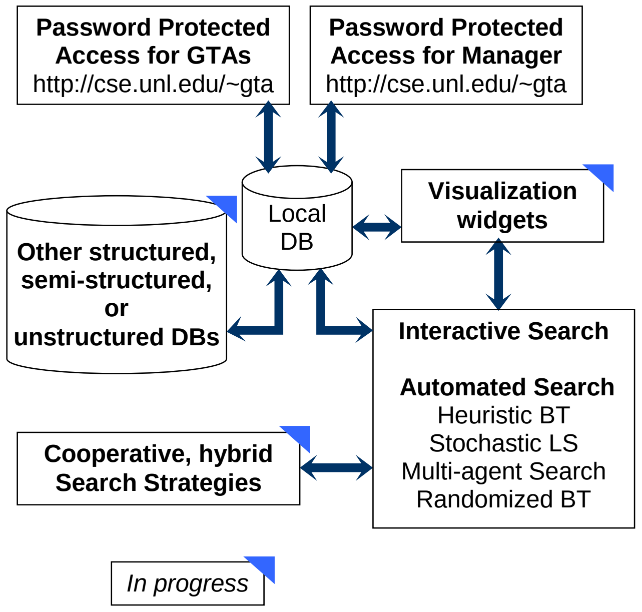


Figure 3.1: GTA assignment project system architecture [Lim *et al.*, 2004a].

acquisition, a relational database for storing the collected data, a number of search algorithms for problem solving, and an additional set of features to generate reports. The interactive system is represented by the 'Interactive Search' component appearing in the lower-right corner of Figure 3.1.

Figure 3.2 highlights the architecture of the interactive system. The web interface

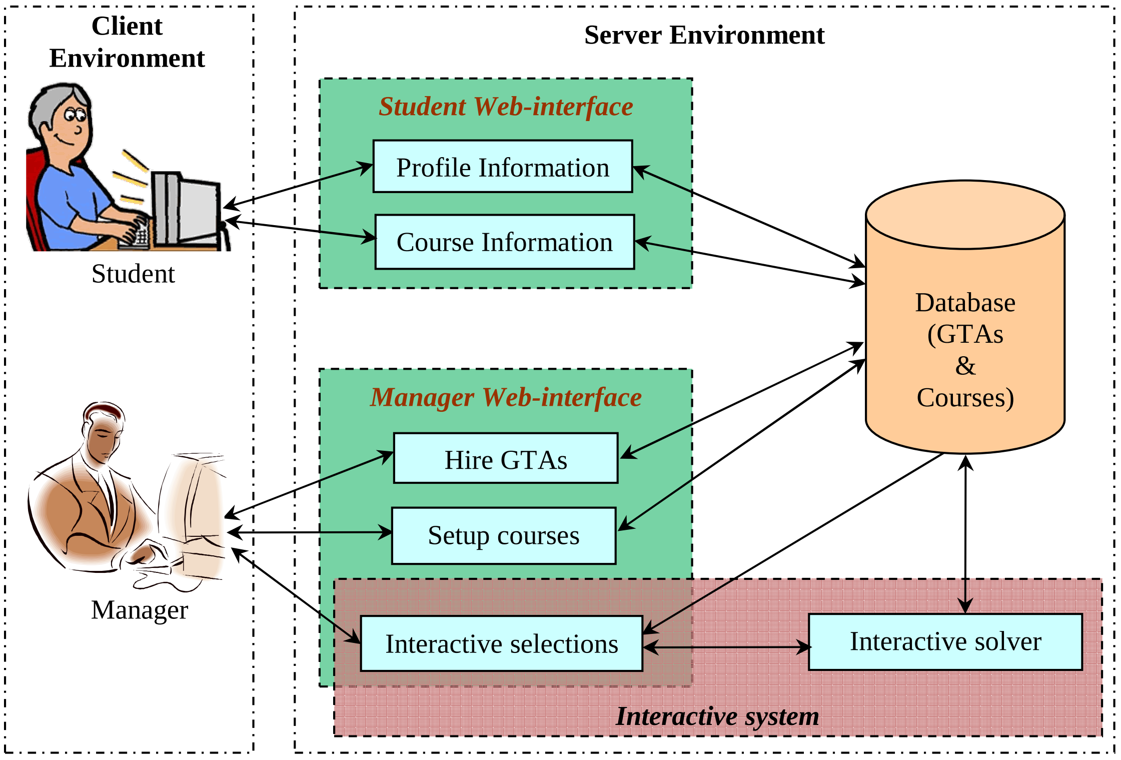


Figure 3.2: Global view of the architecture of the interactive system in GTAAP.

is accessible to both the manager and the students. When an applicant logs in, the student interface is displayed and when a manager logs in, the manager interface is displayed. While a student can access only his/her profile, the manager has complete access to all the student profiles and the entire system. The manager can hire students, set up courses, and define constraints between them. The information about the students and courses is stored in the database. The web interface for interactive selections also connects to the database to retrieve information about courses and students, but it does so indirectly, through the interactive solver. The interface directly connects to the database only to load any previously made assignments that are stored in the database (see Section 3.3.4). A detailed description of the web interface and the interactive solver is given in Section 3.2.

Before a semester begins, the manager loads the list of courses offered by the department from a university-wide database. He/she may add new courses into the system and define constraints between them. All the students who want to be a GTA sign up online for an account. If a student already has an account, he/she simply logs into the account, updates a profile and provides information about the courses that he/she is enrolled in for a semester. A student's profile includes information about his/her personal details, undergraduate/graduate GPAs, deficiencies, GRE scores, ITA qualifications and the courses he/she is registered in for the semester. Once all the students fill their preferences, the manager hires potential GTAs (based on their qualifications and other commitments) for the semester and proceeds to do interactive selections.

The manager's web-interface provides two operational modes for problem solving: interactive and automatic [Lim *et al.*, 2004a]. In both modes, the problem is modeled as a constrained optimization problem. The web interface allows the applicants and the manager to specify the information used for determining the consistency of a solution. This includes the type and load of courses, the qualifications of the GTAs, their hiring capacity, and their preferences for each course on a scale from 5 to 0 ('5 Best choice', '4 Favorite', '3 Qualified', '2 Able to handle', '1 Avoid if possible', '0 Cannot handle'). This information, in addition to the class schedule (retrieved from a university-wide database), is stored in the database.

The GTAAP is implemented using MySQL as the back-end database, PHP as the front-end for web interfaces, and Common Lisp and C++ for the search algorithms.

3.2 Interactive system

The interactive system is designed as a three-tier online client-server system (see Figure 3.3):

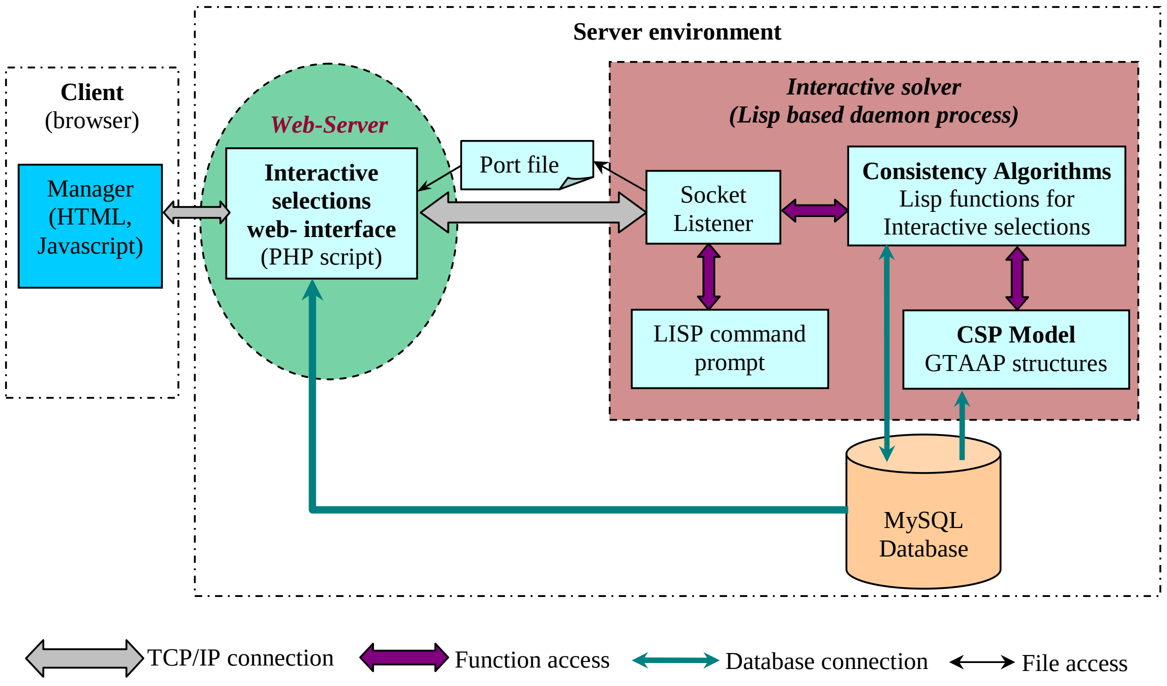


Figure 3.3: Detailed view of the architecture of the interactive system.

1. An interface for user interaction (labeled 'Interactive selections web-interface' in Figure 3.3),

2. Algorithms for problem solving (labeled 'Interactive solver' in Figure 3.3), and

3. A database from which the GTA and course information is loaded.

The web interface provides the access to the interactive solver. At any time, the interface displays an updated and consistent list of courses and GTAs from the CSP. The interactive solver runs as a background daemon process and connects to the database to retrieve the data. Whenever the interactive solver is started on the server, it loads all the GTA and course information from the database. It also has the ability to reload the updated information from the database anytime, upon a user's request.

The web interface connects to the interactive solver using the TCP/IP protocol. Whenever a manager assigns (or unassigns) a GTA to a class through the interface, the solver checks all the constraints, filters the domains, and sends the updated course and GTA information of all the variables and domains, back to the web interface. When the manager undoes or changes assignments, the domains of all the variables are then recomputed from the initial data while maintaining any selections that were previously made. An interactive selection on the web interface comes as a request to the server and the updated domain information of the courses is sent back to the client as a response. In the next section, we describe the individual components of the system.

The database (MySQL) in the system is used to store information about the courses, GTAs, and the course assignments from the web interface.

3.3 Interface for interactive selections

The web interface in the system acts as a visual interface through which the manager interacts with the interactive solver. The manager's actions regarding the courses and GTAs displayed in the interface serve as input to the interactive solver, which outputs new decisions directly to the interface as courses and their domains change. The interface interprets the manager's actions (e.g., assign, unassign, save, or clear) and converts them into commands (see Appendix B) that are understandable by the solver. The interactive solver executes the commands and sends the output back to the interface. The web interface parses this output and generates a new web page ready to receive the manager's new actions.

In the following subsections, we describe the driver that is used to guide the user actions, followed by the interfaces to assign 'GTAs to classes' and 'classes to GTAs' with other features.

3.3.1 Driver for user's actions

Figure 3.4 shows the possible actions available to the manager from the 'Interactive selections' option of GTA project's interface. Each of the boxes is linked to a web

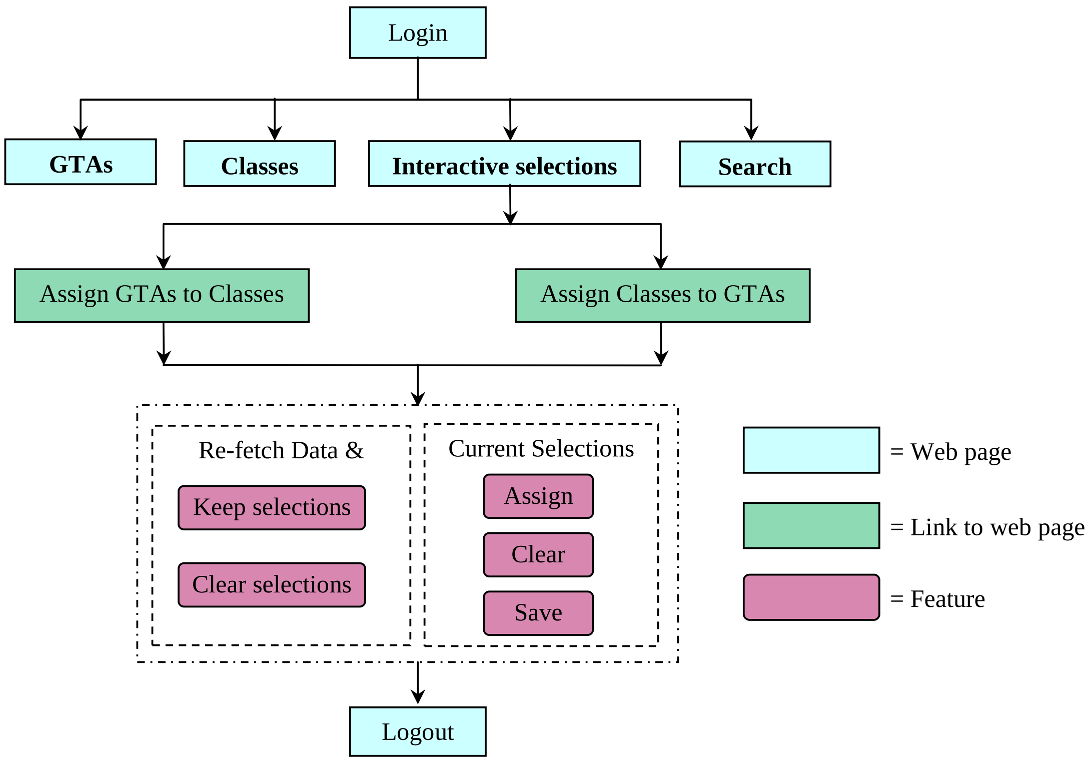


Figure 3.4: Possible actions for the manager under 'Interactive Selections.'

page or a feature on the web page. The manager logs in from the login page using a username and password. After proper authentication, the manager is presented with different pages to do different tasks. These pages include:

- the 'GTAs' page to access and modify the GTA profiles,

- the ‘Classes’ page to add or modify course information in the system,
- the ‘Interactive Selections’ page to do interactive problem-solving, and
- the ‘Search’ page to perform automated search.

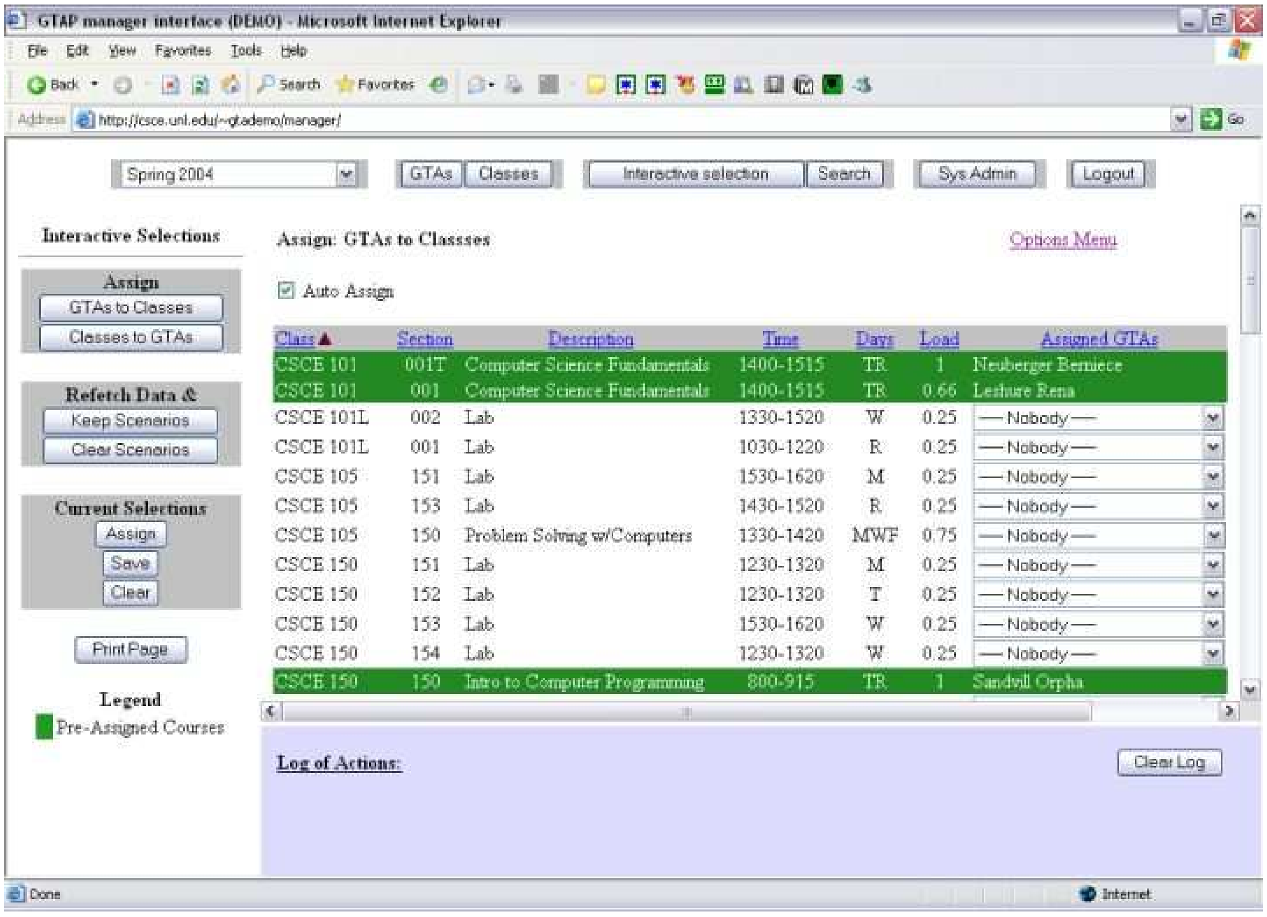
The web interface is built in HTML, PHP and Javascript. As shown in Figure 3.3, the PHP scripts on the server interact with the database and interactive solver to generate HTML pages that are displayed on the manager’s browser. We use Javascripts for creating drop-down menus and validating data on the client side. Some of the attractive features of PHP are its ability to:

- Dynamically generate web pages,
- Connect and retrieve information from a different server, given its IP address and socket number,
- Store user preferences during a session, and
- Possess other capabilities such as sorting and parsing.

The course and GTA data structures in the web interface are built as PHP objects. All courses and GTAs are defined as classes. A course object encapsulates course name, number, section, timings, load, and a set of GTA objects that are consistent with the course. A GTA object encapsulates the GTA name, advisor name, speak test information, ITA certification information, and a set of course objects (course domains) in which the GTA is present. The object-oriented features enable easier implementation of sorting features on the web interface. The objects can be sorted easily according to the data stored inside the object. For example, all the objects can be sorted, in ascending or descending order, according to the course name. Also, the object-oriented features make the interface scalable. In the future, any new details about the courses or GTAs can be added by modifying the required class structures without re-implementing the code for sorting procedures.

3.3.2 Assignment of GTAs to classes

This perspective gives a course-centered view of the CSP. The web page displays courses and the corresponding list of consistent GTAs in their domain. Figure 3.5 shows a snapshot of the perspective. Each row contains the course number, section,



Assign: GTAs to Classses

✓ Auto Assign

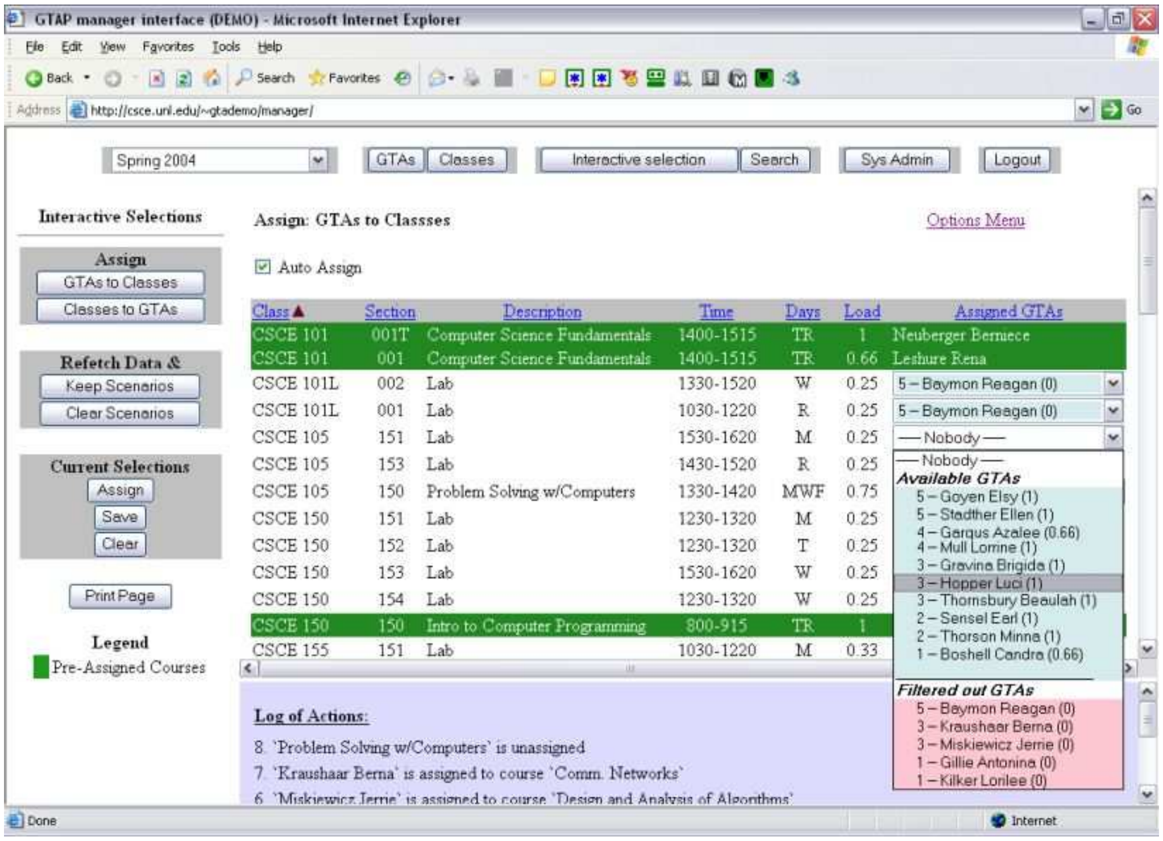
Class ▲	Section	Description	Time	Days	Load	Assigned GTAs
CSCE 101	001T	Computer Science Fundamentals	1400-1515	TR	1	Neuberger Berniece
CSCE 101	001	Computer Science Fundamentals	1400-1515	TR	0.66	Leshure Rena
CSCE 101L	002	Lab	1330-1520	W	0.25	— Nobody —
CSCE 101L	001	Lab	1030-1220	R	0.25	— Nobody —
CSCE 105	151	Lab	1530-1620	M	0.25	— Nobody —
CSCE 105	153	Lab	1430-1520	R	0.25	— Nobody —
CSCE 105	150	Problem Solving w/Computers	1330-1420	MWF	0.75	— Nobody —
CSCE 150	151	Lab	1230-1320	M	0.25	— Nobody —
CSCE 150	152	Lab	1230-1320	T	0.25	— Nobody —
CSCE 150	153	Lab	1530-1620	W	0.25	— Nobody —
CSCE 150	154	Lab	1230-1320	W	0.25	— Nobody —
CSCE 150	150	Intro to Computer Programming	800-915	TR	1	Sandvill Orpha

Log of Actions:

Figure 3.5: Snapshot of perspective 'Assign GTAs to Classes.'

title, timings, course load, and a list of selectable GTAs in its domain (in a drop-down menu). The drop-down menu contains the GTA preference for the course, GTA name, and the available capacity. The background of the row is shown in green if the course has a pre-assigned GTA. Pre-assigned courses will have a fixed GTA assigned to the course and thus the course variable will not be included in any consistency algorithms. The page also contains an 'Auto Assign' option. When this option is selected, the page is updated immediately when the manager chooses a GTA from the drop-down menu, and propagation algorithms are launched over the unassigned courses. If this option is not selected, the manager can do multiple assignments and then he/she can manually press the 'Assign' button in the left frame of the page to confirm the changes to the interactive solver. A log of the manager's actions is displayed at the bottom portion of the browser.

A course along with the GTAs present in its domain is shown in Figure 3.6.



Assign: GTAs to Classses

✓ Auto Assign

Class ▲	Section	Description	Time	Days	Load	Assigned GTAs
CSCE 101	001T	Computer Science Fundamentals	1400-1515	TR	1	Neuberger Berniece
CSCE 101	001	Computer Science Fundamentals	1400-1515	TR	0.66	Leshure Rena
CSCE 101L	002	Lab	1330-1520	W	0.25	5 – Baymon Reagan (0)
CSCE 101L	001	Lab	1030-1220	R	0.25	5 – Baymon Reagan (0)
CSCE 105	151	Lab	1530-1620	M	0.25	— Nobody —
CSCE 105	153	Lab	1430-1520	R	0.25	
CSCE 105	150	Problem Solving w/Computers	1330-1420	MWF	0.75	
CSCE 150	151	Lab	1230-1320	M	0.25	
CSCE 150	152	Lab	1230-1320	T	0.25	
CSCE 150	153	Lab	1530-1620	W	0.25	
CSCE 150	154	Lab	1230-1320	W	0.25	
CSCE 150	150	Intro to Computer Programming	800-915	TR	1	
CSCE 155	151	Lab	1030-1220	M	0.33	

— Nobody —
Available GTAs
5 – Goyen Elsy (1)
5 – Stadther Ellen (1)
4 – Garqus Azalee (0.66)
4 – Mull Lorrine (1)
3 – Gravina Brigida (1)
3 – Hopper Luci (1)
3 – Thornsbury Beaulah (1)
2 – Sensel Earl (1)
2 – Thorson Minna (1)
1 – Boshell Candra (0.66)
Filtered out GTAs
5 – Baymon Reagan (0)
3 – Kraushaar Berna (0)
3 – Miskiewicz Jerrie (0)
1 – Gillie Antonina (0)
1 – Kilker Lorilee (0)

Log of Actions:

8. 'Problem Solving w/Computers' is unassigned
7. 'Kraushaar Berna' is assigned to course 'Comm. Networks'
6. 'Miskiewicz Jerrie' is assigned to course 'Design and Analysis of Algorithms'

Figure 3.6: Domains of courses when assigning.

Depending on the GTAs' input, each course has a certain number of possible GTAs in its domain. A GTA can be present in more than one course domain. These GTAs are displayed in sorted order according to the GTA's preference for the course. A course, its information, and the corresponding GTAs present in its drop-down menu are shown in Figure 3.7. The GTAs displayed in the drop-down menu are divided into

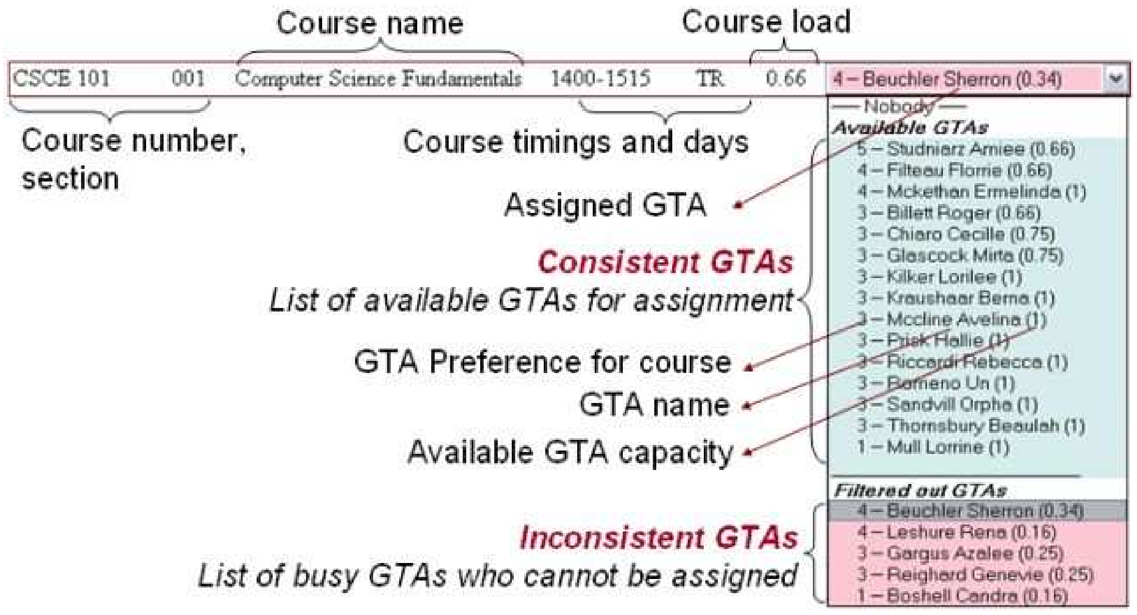
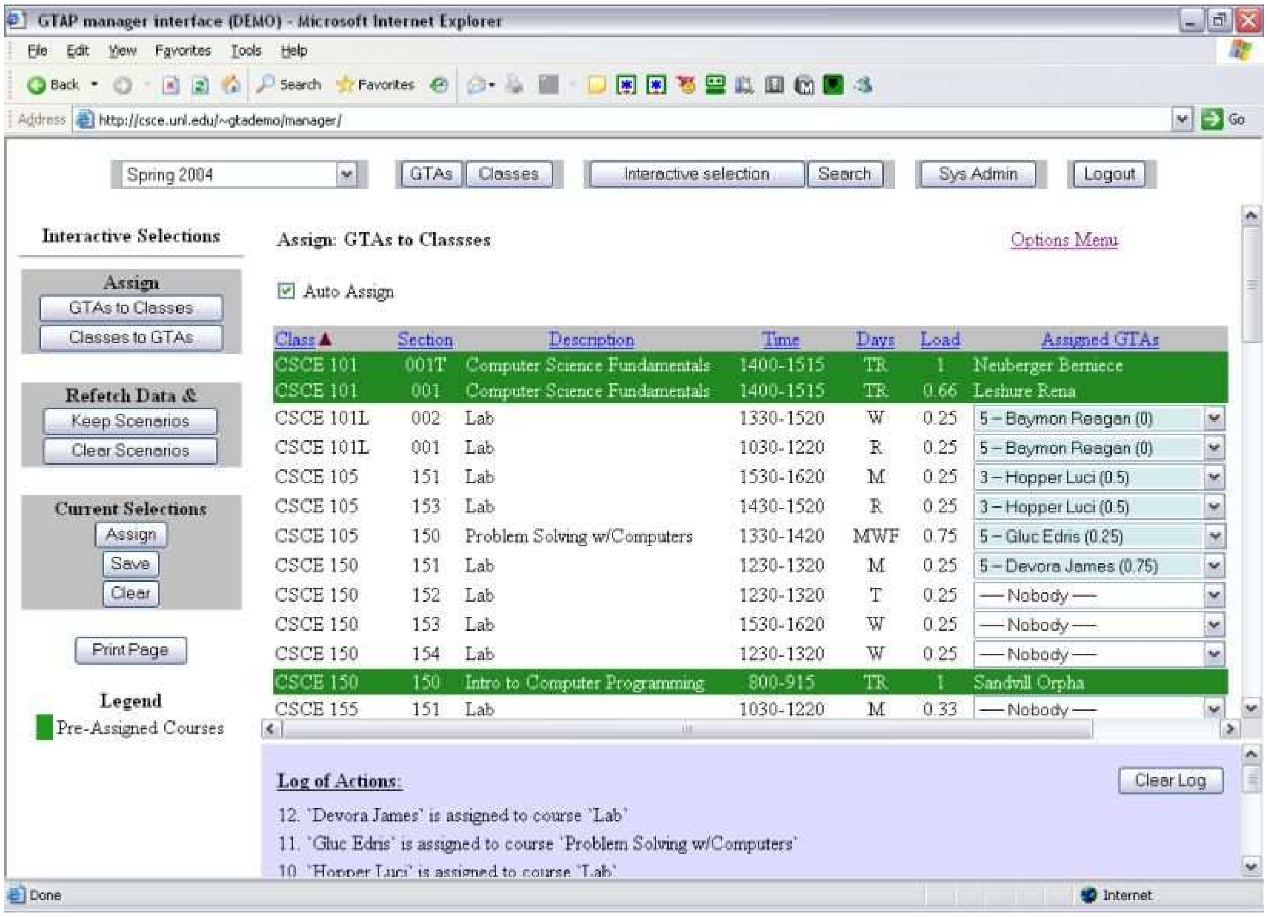


Figure 3.7: An example of a course and a list of consistent GTAs present in its domain.

'Available GTAs' and 'Filtered-out GTAs.' 'Available GTAs' are the ones who can be assigned to the course, and 'Filtered out GTAs' are those whose remaining capacity is less than the course load. These GTAs are eliminated from the course domain by constraint propagation. In each portion, the GTAs are listed in decreasing preference order, as a primary sorting criterion, and then in increasing lexicographical of the GTA's 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).

A snapshot of the manager's browser after a few assignments is shown in Figure 3.8. The manager can assign a GTA to a particular course by selecting from the



Legend

🟩 Pre-Assigned Courses

Assign: GTAs to Classses

☑ Auto Assign

Class ▲	Section	Description	Time	Days	Load	Assigned GTAs
CSCE 101	001T	Computer Science Fundamentals	1400-1515	TR	1	Neuberger Bernece
CSCE 101	001	Computer Science Fundamentals	1400-1515	TR	0.66	Leshure Rena
CSCE 101L	002	Lab	1330-1520	W	0.25	5 – Baymon Reagan (0)
CSCE 101L	001	Lab	1030-1220	R	0.25	5 – Baymon Reagan (0)
CSCE 105	151	Lab	1530-1620	M	0.25	3 – Hopper Luci (0.5)
CSCE 105	153	Lab	1430-1520	R	0.25	3 – Hopper Luci (0.5)
CSCE 105	150	Problem Solving w/Computers	1330-1420	MWF	0.75	5 – Gluc Edris (0.25)
CSCE 150	151	Lab	1230-1320	M	0.25	5 – Devora James (0.75)
CSCE 150	152	Lab	1230-1320	T	0.25	— Nobody —
CSCE 150	153	Lab	1530-1620	W	0.25	— Nobody —
CSCE 150	154	Lab	1230-1320	W	0.25	— Nobody —
CSCE 150	150	Intro to Computer Programming	800-915	TR	1	Sandvill Orpha
CSCE 155	151	Lab	1030-1220	M	0.33	— Nobody —

Log of Actions:

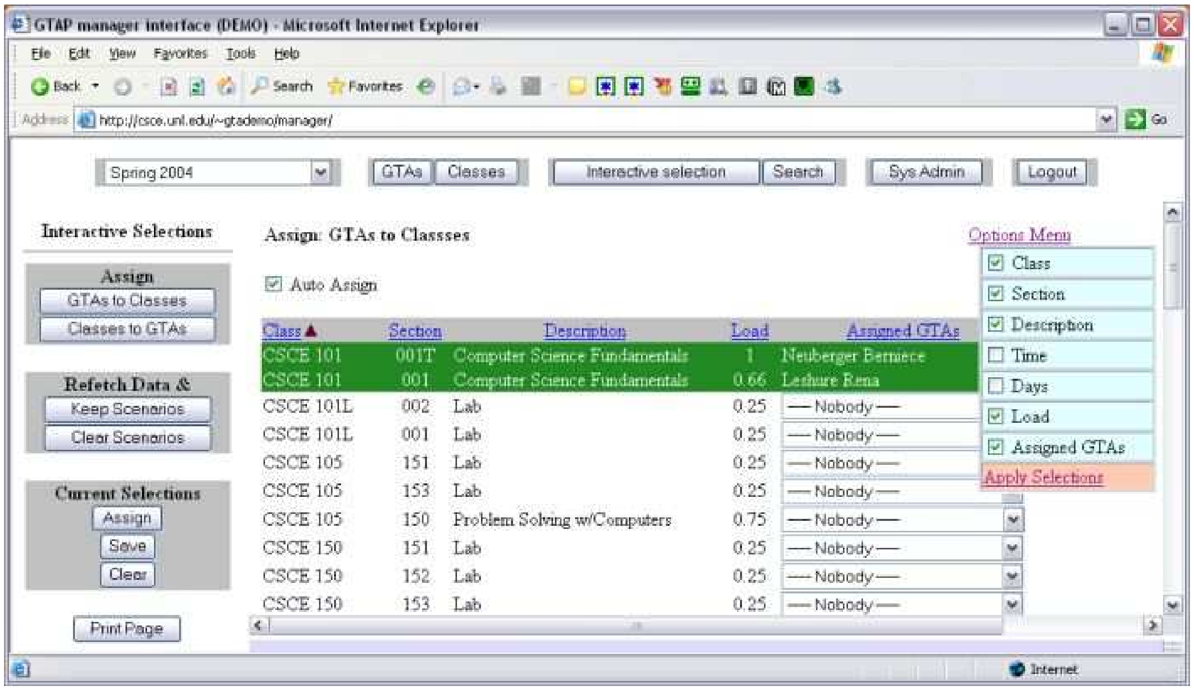
12. 'Devora James' is assigned to course 'Lab'
11. 'Gluc Edris' is assigned to course 'Problem Solving w/Computers'
10. 'Hopper Luci' is assigned to course 'Lab'

Figure 3.8: Assignment of GTAs to Classes.

drop-down menu. Whenever a GTA is assigned to a course, constraint propagation (which involves checking all the associated unary, binary and non-binary constraints) is performed and all the course domains are automatically updated according to the assignment. This is done so that the arc-consistency of the CSP is maintained. All inconsistent GTAs are filtered from course domains and the manager is always presented with an updated and a sorted list of possible consistent choices. This constrains the manager to always do a consistent assignment.

The page is also equipped with sorting features to provide better assistance to the manager. These features allow a manager to sort columns in ascending or descending order. The sorting functionality is implemented in PHP. Figure 3.5 shows the column names displayed as hyperlinks to PHP scripts. These scripts retrieve the data from the interactive solver and sort them according to the column name before generating the HTML page. A symbol (‘▲’ for ascending and ‘▼’ for descending) after the sorted column name indicates the sorting order.

The page also has flexibility to allow the manager to control the display on the page. The interface provides an 'Options menu' from which the manager can select or unselect the column names. This allows the manager to view only selected column names in the interface. The menu is built using Javascript and the manager's options are stored in PHP sessions[1]. These options are sent to the web interface when the manager selects or unselects a column name in the menu. The PHP script checks the session information to read the manager's options and generates an HTML page with or without the selected column. A snapshot of the menu is shown in Figure 3.9. Here, the course timings and days are not displayed in the interface since the manager



☑ Auto Assign

Class ▲	Section	Description	Load	Assigned GTAs
CSCE 101	001T	Computer Science Fundamentals	1	Neuberger Bernece
CSCE 101	001	Computer Science Fundamentals	0.66	Leshure Rena
CSCE 101L	002	Lab	0.25	— Nobody —
CSCE 101L	001	Lab	0.25	— Nobody —
CSCE 105	151	Lab	0.25	— Nobody —
CSCE 105	153	Lab	0.25	— Nobody —
CSCE 105	150	Problem Solving w/Computers	0.75	— Nobody —
CSCE 150	151	Lab	0.25	— Nobody —
CSCE 150	152	Lab	0.25	— Nobody —
CSCE 150	153	Lab	0.25	— Nobody —

☑ Class
☑ Section
☑ Description
☐ Time
☐ Days
☑ Load
☑ Assigned GTAs

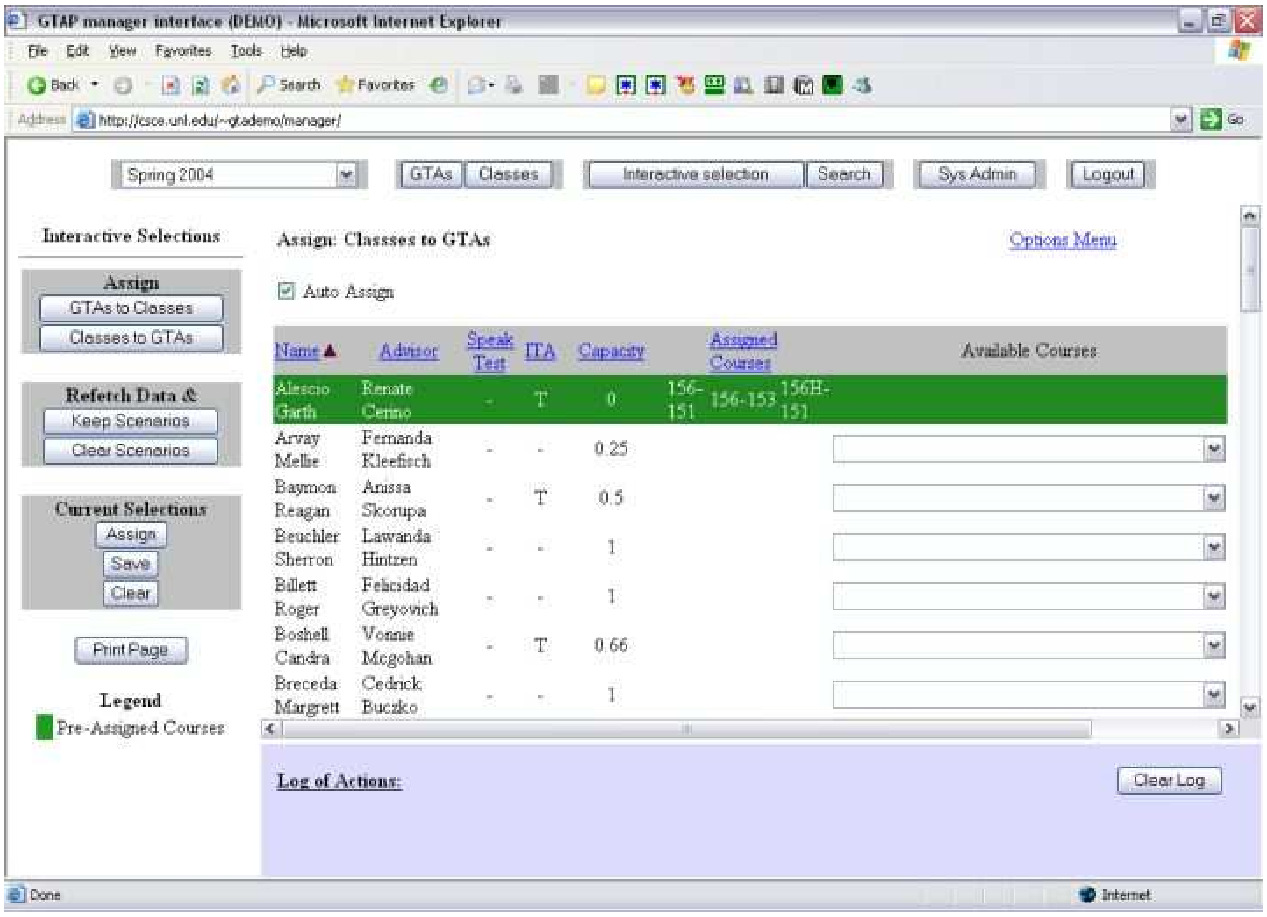
Figure 3.9: Options menu.

unselected the two options in the 'Options menu.'

[1]PHP sessions are used to store information about user preferences on the server side.

3.3.3 Assignment of classes to GTAs

The web interface for 'Assignment of Classes to GTAs' is shown in Figure 3.10. This



Assign: Classses to GTAs

☑ Auto Assign

Name ▲	Advisor	Speak Test	ITA	Capacity	Assigned Courses	Available Courses
Alescio Garth	Renate Cerino	-	T	0	156-151 156-153 156H-151	
Arvay Mellie	Fernanda Kleefisch	-	-	0.25		
Baymon Reagan	Anissa Skorupa	-	T	0.5		
Beuchler Sherron	Lawanda Hintzen	-	-	1		
Billett Roger	Felicidad Greyovich	-	-	1		
Boshell Candra	Vonnie Mcgohan	-	T	0.66		
Breceda Margrett	Cedrick Buczko	-	-	1		

🟩 Pre-Assigned Courses

Log of Actions:

Figure 3.10: Snapshot of perspective 'Assign Classes to GTAs.'

page design is similar to the 'GTAs to Classes' page, but here the GTA information and corresponding courses are displayed. The GTA information includes the GTA name, advisor name, speak test, ITA qualification, GTA capacity, and the courses to which the GTA is assigned. The drop-down menu contains the courses to which the GTA can be assigned. The drop-down menu also contains the GTA preference towards the course, the course number, section number and the course load.

Figure 3.11 shows an example of a GTA and the list of courses present in which he/she is present. Like the GTA menu described earlier, the course menu is divided

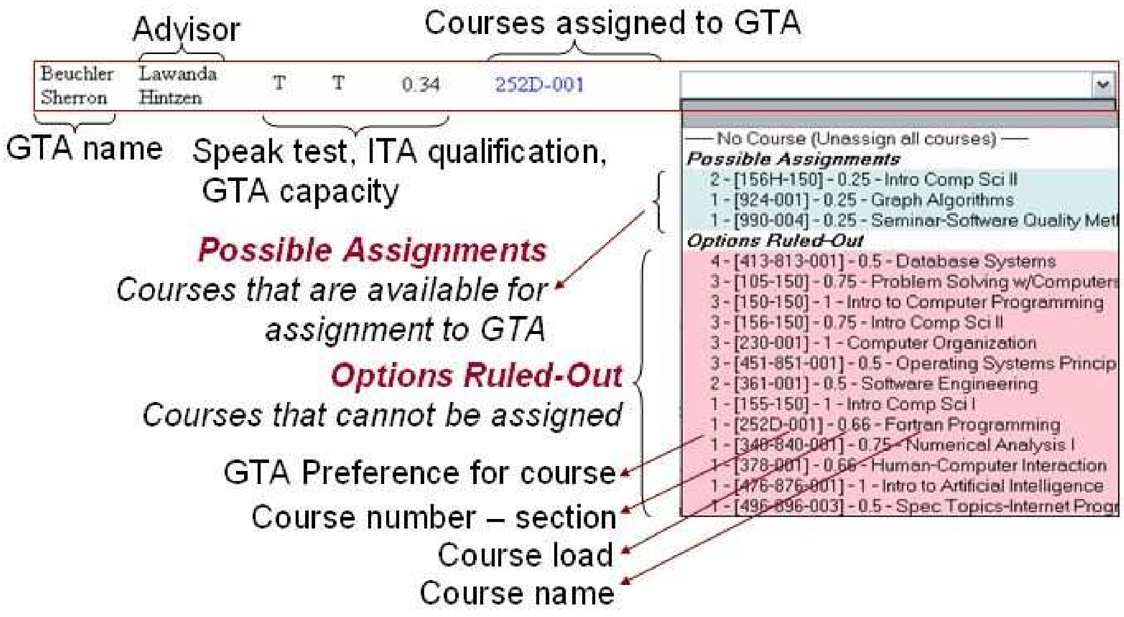

Advisor
Courses assigned to GTA
Beuchler Sherron
Lawanda Hintzen
T T 0.34 252D-001
GTA name
Speak test, ITA qualification, GTA capacity
Possible Assignments
Courses that are available for assignment to GTA
Options Ruled-Out
Courses that cannot be assigned
GTA Preference for course
Course number – section
Course load
Course name
— No Course (Unassign all courses) —
Possible Assignments
2 - [156H-150] - 0.25 - Intro Comp Sci II
1 - [924-001] - 0.25 - Graph Algorithms
1 - [990-004] - 0.25 - Seminar-Software Quality Met
Options Ruled-Out
4 - [413-813-001] - 0.5 - Database Systems
3 - [105-150] - 0.75 - Problem Solving w/Computer
3 - [150-150] - 1 - Intro to Computer Programming
3 - [156-150] - 0.75 - Intro Comp Sci II
3 - [230-001] - 1 - Computer Organization
3 - [451-851-001] - 0.5 - Operating Systems Princip
2 - [361-001] - 0.5 - Software Engineering
1 - [155-150] - 1 - Intro Comp Sci I
1 - [252D-001] - 0.66 - Fortran Programming
1 - [340-840-001] - 0.75 - Numerical Analysis I
1 - [378-001] - 0.66 - Human-Computer Interaction
1 - [476-876-001] - 1 - Intro to Artificial Intelligence
1 - [496-896-003] - 0.5 - Spec Topics-Internet Progr

Figure 3.11: An example of a GTA and a list of courses in which he/she is present.

into 'Possible Assignments' and 'Options Ruled-Out.' All the courses that have a course load less than the GTA capacity can be assigned to a GTA. These courses are listed as 'Possible Assignments.' The remaining courses that cannot be assigned to a GTA are listed as 'Options Ruled-Out.'

The ‘Options menu’ and sorting features in the interface are similar to the features in the ‘GTAs to classes’ interface. Because the two perspectives are derived from the same CSP, an assignment made in one of them is immediately reflected in the other. Figure 3.12 shows the two perspectives derived from the same CSP. Thus, the manager is always presented with a consistent CSP in both perspectives.

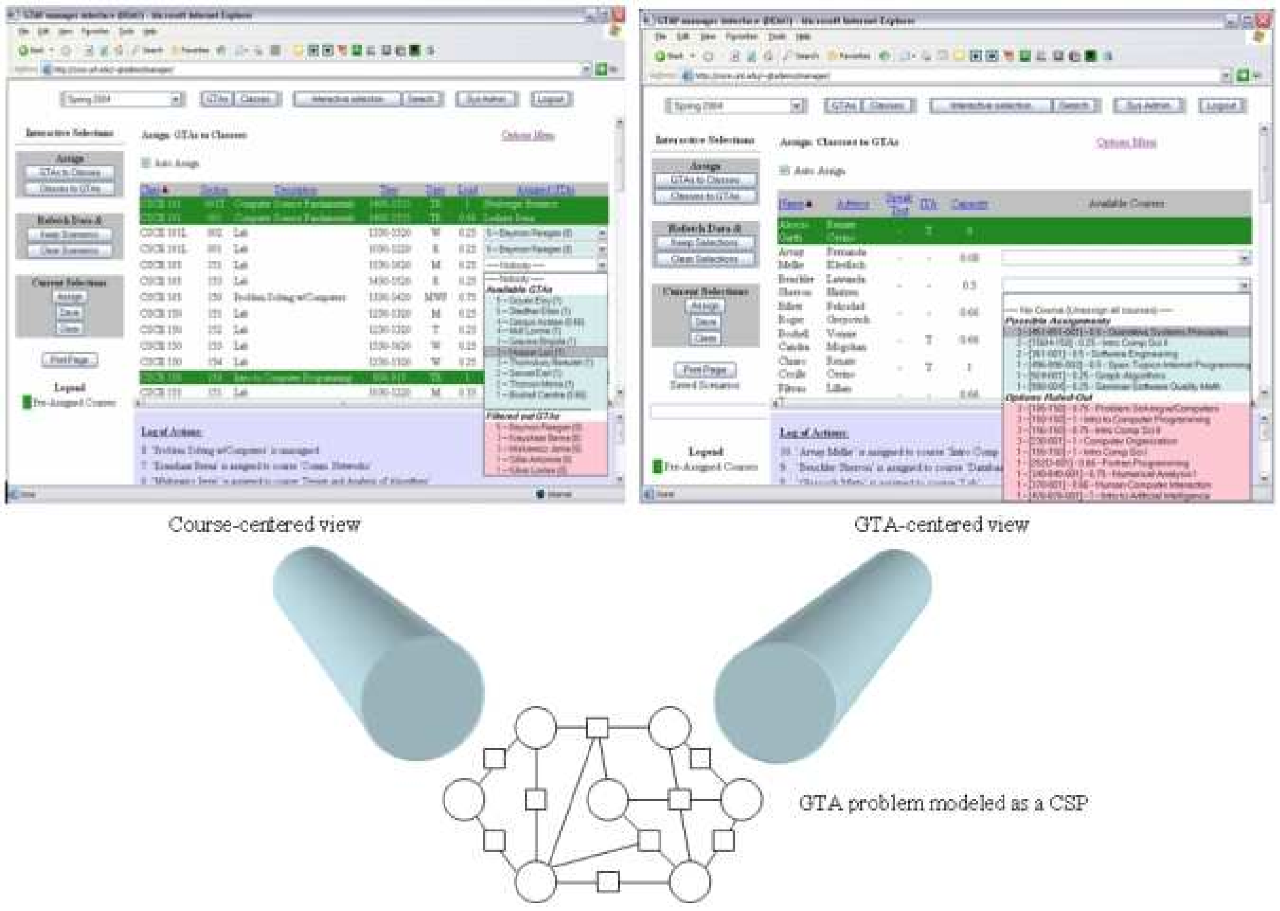


Figure 3.12: Dual perspective for decision making.

3.3.4 Other features

Each of the two perspectives has the following additional set of features. These features allow the manager to save the current session and retrieve them at a later time.

3.3.4.1 Confirming assignments

By default a manager can assign one GTA to a course at a time before consistency checking is launched. The 'Assign current selections' feature allows him/her to do multiple assignments at the same time before starting constraint propagation.

3.3.4.2 Saving scenarios

Using this feature the manager can give a friendly name and save all the current assignments in the database so that he/she can explore alternative assignments. Using the name given to the CSP, the web interface creates a Lisp command

```
(save-current-csp *php-stream* csp-name)
```

and sends it to the interactive solver for execution. This function saves the CSP in the database. A detailed description of all Lisp commands is given in Appendix B.

The structures of tables defined in MySQL database are shown in Tables 3.1 and 3.2. Table 3.1 stores all the names of saved scenarios along with the unique

Table 3.1: Database table where names of alternative scenarios are stored.

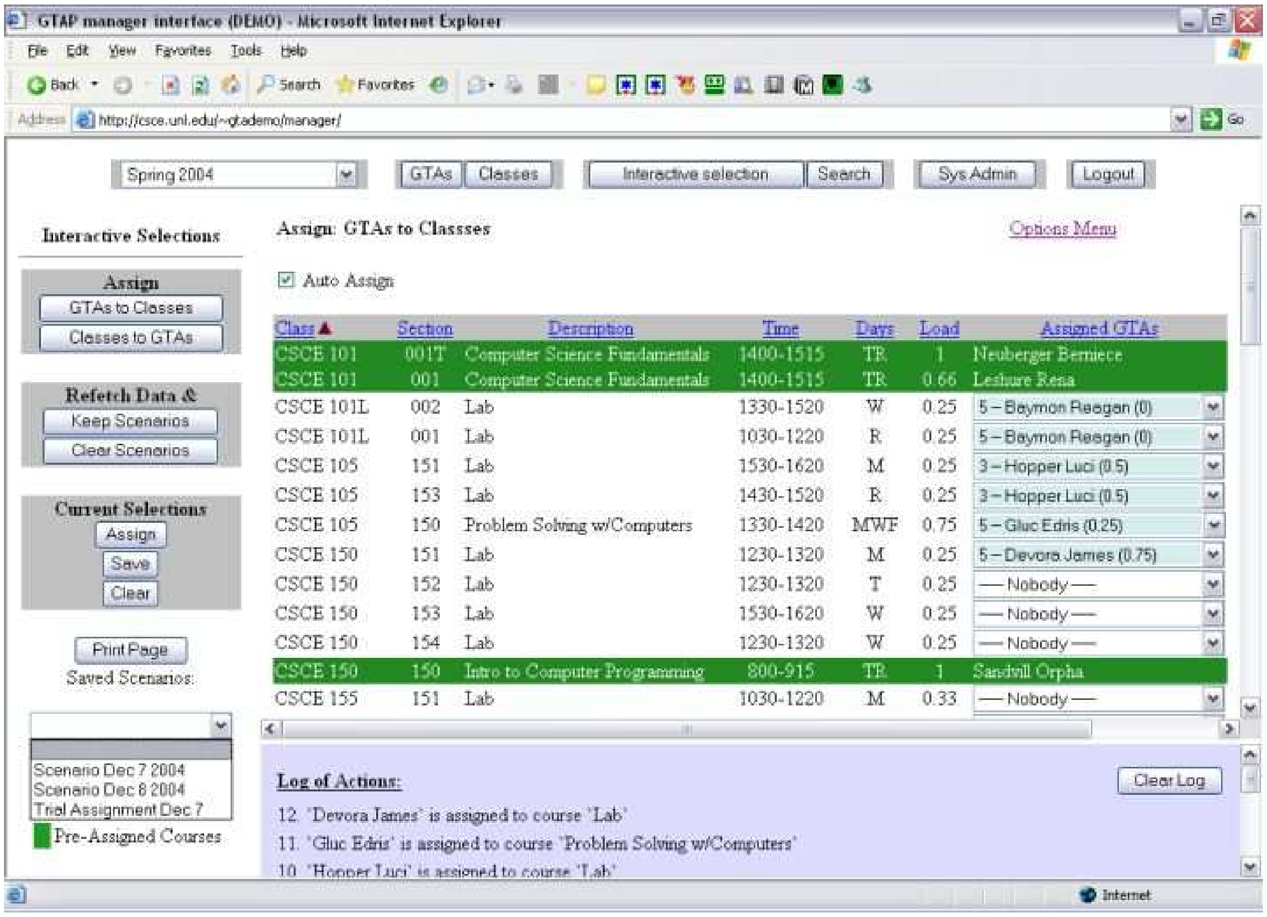
Assignment-id	**Assignment (scenario) name**	**Year**	**Semester**
1	Trial on Fri Aug 13	2004	23
2	Trial on Fri Aug 14	2004	23

Table 3.2: Database table for storing the details of an alternative scenario.

Assignment-id	**Course-id**	**GTA-id**
1	32	21
1	34	32
2	34	42
2	24	41

id generated by the database, the year, and semester. Table 3.2 stores the course and the GTA ids. The course-id and GTA-id are used to link to the GTAAP course database tables that contain complete course and GTA details.

The manager can save any number of assignments and can resume the saved selections at any time later *as long as the list of courses has not been modified in the database*. Once all current selections are saved, the system is initialized again from scratch and a new CSP is displayed. All saved assignments are displayed in a drop-down menu as shown in Figure 3.13. The menu is located in the left-hand corner of the page.



Assign: GTAs to Classses

☑ Auto Assign

Class ▲	Section	Description	Time	Days	Load	Assigned GTAs
CSCE 101	001T	Computer Science Fundamentals	1400-1515	TR	1	Neuberger Berniece
CSCE 101	001	Computer Science Fundamentals	1400-1515	TR	0.66	Leshure Rena
CSCE 101L	002	Lab	1330-1520	W	0.25	5 – Baymon Reagan (0)
CSCE 101L	001	Lab	1030-1220	R	0.25	5 – Baymon Reagan (0)
CSCE 105	151	Lab	1530-1620	M	0.25	3 – Hopper Luci (0.5)
CSCE 105	153	Lab	1430-1520	R	0.25	3 – Hopper Luci (0.5)
CSCE 105	150	Problem Solving w/Computers	1330-1420	MWF	0.75	5 – Gluc Edris (0.25)
CSCE 150	151	Lab	1230-1320	M	0.25	5 – Devora James (0.75)
CSCE 150	152	Lab	1230-1320	T	0.25	— Nobody —
CSCE 150	153	Lab	1530-1620	W	0.25	— Nobody —
CSCE 150	154	Lab	1230-1320	W	0.25	— Nobody —
CSCE 150	150	Intro to Computer Programming	800-915	TR	1	Sandvill Orpha
CSCE 155	151	Lab	1030-1220	M	0.33	— Nobody —

Log of Actions:

12. 'Devora James' is assigned to course 'Lab'
11. 'Gluc Edris' is assigned to course 'Problem Solving w/Computers'
10. 'Hopper Luci' is assigned to course 'Lab'

Figure 3.13: Drop-down menu to save/retrieve assignments.

The saved assignments are retrieved by selecting the name of the assignments from the drop-down list. As shown in Figure 3.3, the web interface directly connects to the database to retrieve the names of all saved scenarios. After selecting the name of saved assignments, a new CSP is created and the GTA assignments are made by procedure **Procedure 5**. The interface issues a warning message to save any current assignments before retrieving the CSP from the database. All current assignments are lost unless they are saved in a scenario.

3.3.4.3 Clearing stored scenarios

This feature clears all the current assignments by initializing the CSP again and running the consistency algorithms.

3.3.4.4 Re-fetch data & keep scenarios

During the interactive assignments, new GTAs or courses can be added or removed at any point in time. This feature gives the manager the ability to re-fetch the updated or new data from the database while keeping the current assignments. Whenever the manager uses this feature, the following sequence of steps are performed:

1. Save the current assignments.

2. Reload the GTA and course data.

3. Initialize the CSP.

4. Re-established maintained assignments.

5. Propagate constraints by running the consistency algorithms.

3.3.4.5 Refetch data & clear selections

This feature is similar to the 'Re-fetch data & keep selections' feature, but the only difference is that all the saved and the current assignments will be cleared. The CSP of the GTAAP instance must be rebuilt when some of the hired GTAs are removed from the database or new ones are hired. Thus, it becomes imperative to remove clear the saved assignments that might contain the deleted GTA.

3.4 Interactive solver

The interactive solver is basically a set of Lisp functions that execute the interactive selections. The solver is built over the existing GTAAP's underlying data structures. As shown in Figure 3.3, the solver contains an implementation of a socket-listening interface that provides access to the solver and a set of Lisp functions that do interactive selections using the basic data structures designed for the GTAAP. The GTA data structures, built using object-oriented features of Lisp, contain implementation of a CSP model for the GTAAP. While the web interface for interactive selections provides an abstract view of the GTAAP, the interactive solver holds the GTAAP as a CSP. It does the actual constraint checks, domain filtering and course assignments in the CSP. Whenever a manager makes an assignment in the web interface, a full arc-consistency is executed over the unassigned variables to filter their domains in the CSP using **Procedure 5**.

The solver runs as an independent process on a given socket. During the initialization, the socket listener module checks for an available free port and starts a listener on that port. This port number, which is written to a text file, is used for all communications with the web interface. This text file is accessed by the web interface (PHP script) to determine the port number on which the process is running.

The solver must run as a background daemon process for faster and more efficient interaction with the web interface. A Lisp program runs from the Lisp environment. To make the solver run continuously as a background process, we have used a utility called 'detachtty' [det, 2001]. This utility can make any interactive program on Unix machines run in the background. 'detachtty' allows a user to run interactive programs as background processes, and to connect to these processes over the network. It is mainly designed for long-running processes. The main components of the interactive solver are the socket listener and the consistency algorithms

3.4.1 Socket listener

As mentioned in Section 3.2, this module acts as a gateway to the interactive solver. It accepts and sends any information from a client, usually the web interface. It listens to requests on a given socket and calls appropriate functions. The socket listener retrieves the output (if any) of executed function and sends the data back. All the requests are accepted in the form of calls to Lisp functions. Although the listener can handle any call to a Lisp function, the common requests that are sent by the web interface are listed in Table 3.3.

Table 3.3: Lisp commands and their description.

Lisp commands	**Description**
`(in-package gta)`	Makes sure the control is in `gta` package and not in `user` package.
`(load-initialize-csp)`	Creates a CSP and initializes it with data from current year and semester.
`(assign-gta *php-stream* gtaname course *current-csp*)`	Assigns a GTA to a course in the `*current-csp*`.
`(check-assignment *php-stream* course *current-csp*)`	Checks if a given `course` is assigned with any GTA.
`(check-year-sem year semester)`	Checks if the `*current-csp*` is of a particular `year` and `semester`.
`(display-gtas *php-stream* *current-csp*)`	Retrieves all the GTAs and the course domains in which they are present.
`(display-courses *php-stream* *current-csp*)`	Retrieves all the courses and the GTAs present in their domain.
`(load-assigned-courses *current-csp*)`	Loads all the previously assigned courses, from the variable `*assigned-courses*`, into the `*current-csp*`.
`(process-nc *current-csp*)`	Makes the current CSP node-consistent.
`(ac-1 *current-csp*)`	Makes the current CSP arc-consistent.
`(save-current-csp *php-stream* csp-name)`	Saves the current CSP into the database with a given name.
`(make-current *php-stream* csp-name)`	This method loads a saved CSP from MySQL database and makes it the `*current-csp*`.
`(clear-saved-assignments nil)`	Clears all the saved assignments in the MySQL database.

3.4.2 Consistency algorithms

As explained in Section 3.3, ‘Interactive solver’ is a set of routines that together do the interactive assignments. It has the core logic to assign, unassign, and check constraints. The methodology and the algorithms are explained in Chapter 2. The function names are listed in Table 3.3. The module includes functions to:

1. Retrieve courses and the GTAs present in their domains.
2. Retrieve GTAs and courses in which they are present.
3. Assign a GTA to a course (Procedure 5).
4. Unassign a GTA from a course (Procedure 7).
5. Perform node-consistency (Algorithm 2).
6. Perform arc-consistency (Algorithm 4).
7. Refetch data from database.
8. Save/clear/retrieve assignments from the database.

Summary

In this chapter we described the design and implementation of the interactive system, which is designed as a three-tier architecture with a web interface, a constraint-based interactive solver, and a relational database.

Chapter 4

Conclusion and future work

In this project, we have designed and implemented an interactive system for solving the GTAAP. We have used propagation algorithms for maintaining the consistency of the problem at any time. These basic algorithms of Constraint Processing proved to be exactly what is needed to effectively support a human user in the difficult task of assigning GTAs to courses in our department. The success of the system has been replicated in other departments at the university. Our approach can be extended in many directions. We list a few of these directions below:

1. The interactive solver can be linked to other search algorithms and with the help of a user, it can guide a search process through the search space effectively and efficiently.

2. The interface can interact with the user at various stages of the search process and take hints from the user and/or the automated search algorithms to avoid livelocks/deadlocks, check termination conditions, etc.

3. The socket listener module in the interactive solver can also be used in the development of visualization tools to view the solution space of the problem.

4. The partial solutions that are obtained during the interactive assignments can be compared and combined into complete solutions.

5. Finally, the interface can be further modified so that the manager can make an initial set of assignments, and can let the search algorithms take over the problem and to find the remaining assignments.

Bibliography

[Barták, 1998] Roman Barták. On-Line Guide to Constraint Programming. http://kti.ms.mff.cuni.cz/~bartak/constraints, 1998.

[Bessière, 1991] Christian Bessière. Arc-Consistency in Dynamic Constraint Satisfaction Problems. In *Proc. of AAAI-91*, pages 221–226, 1991.

[det, 2001] Detachtty: Software to attach or detach an interactive processes from the network http://packages.debian.org/stable/admin/detachtty, 2001.

[Glaubius and Choueiry, 2002a] Robert Glaubius and Berthe Y. Choueiry. Constraint Modeling and Reformulation in the Context of Academic Task Assignment. In *Working Notes of the Workshop Modelling and Solving Problems with Constraints, ECAI 2002*, Lyon, France, 2002.

[Glaubius and Choueiry, 2002b] Robert Glaubius and Berthe Y. Choueiry. Constraint Modeling and Reformulation in the Context of Academic Task Assignment. In *Poster presentation at the Fifth International Symposium on Abstraction, Reformulation and Approximation, SARA 2002*, 2002.

[Glaubius and Choueiry, 2002c] Robert Glaubius and Berthe Y. Choueiry. Constraint Modeling in the Context of Academic Task Assignment. In Pascal Van Hentenryck, editor, *Proceedings of* 8^{th} *International Conference on Principle and Practice of Constraint Programming (CP'02)*, volume 2470 of *Lecture Notes in Computer Science*, page 789, Ithaca, NY, 2002. Springer Verlag.

[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. 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.

[Kopfer and Schnberger, 2002] H. Kopfer and J. Schnberger. Interactive solving of vehicle routing and scheduling problems: Basic concepts and qualification of tabu search approaches. In *Proceedings of the* 35th *Annual Hawaii International Conference on System Sciences (HICSS 02)*, volume 3, page 84. IEEE Computer Society, 2002.

[Lim *et al.*, 2004a] Ryan Lim, Venkata Praveen Guddeti, and Berthe Y. Choueiry. An Interactive, Constraint-Based System for Task Allocation in an Academic Environment. In Mark Wallace, editor, *An Interactive, Constraint-Based System for Task Allocation in an Academic Environment 04*, volume 3258 of *Lecture Notes in Computer Science*, page 817, Toronto, Canada, 2004. Springer Verlag.

[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.

[Mackworth and Freuder, 1984] Alan K. Mackworth and Eugene C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, (25) 1:65–74, 1984.

[Pu and Faltings, 2002] Pearl Pu and Boi Faltings. Effective Interaction Principles for User-Involved Constraint Problem Solving. In *Second International Workshop on User-Interaction in Constraint Satisfaction, the Eighth International Conference on Principles and Practice of Constraint Programming, September 2002*, 2002.

[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, Master's thesis, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, December 2003.

Appendix A

File and Data structures

This appendix describes the file and data structures used in the interactive solver.

A.1 File structure

The following tree structure displays the directory and file structure:

```
-GTA-+
     |-- bin --+
     |         |-- detachtty-src --+
     |         |                   |-- Source files for detachtty.
     |         |-- startdaemon.pl
     |         |-- configure-load.lisp
     |         |-- detachtty
     |-- web-interface --+
     |                   |-- ac.lisp
     |                   |-- configure-load.lisp
     |                   |-- global-variables.lisp
     |                   |-- loadgta.lisp
     |                   |-- makecsp.lisp
     |                   |-- misc-funcs.lisp
     |                   |-- savecsp.lisp
     |                   |-- showcourses.lisp
     |                    |-- showgtas.lisp
     |                    |-- sock-listener.lisp
     |                    |-- testcsp.lisp
     |                    |-- testserv.lisp
     |-- Files.list
```

In the above tree, the directory ‘GTA’ also contains the files and directories (not shown) related to GTA package. The content of each directory is described below:

- *bin*: This directory is used to hold external utilities that are used to start and run the interactive solver. For now it contains compiled ‘detachtty’ utility.
- *detachtty-src*: contains the source code for compiling the detachtty utility for various operating systems.
- *web-interface*: contains the Lisp files that load the GTA data structures.

The content of each file is described as follows:

- *ac.lisp*: contains an implementation of arc-consistency algorithm and related functions.
- *configure-load.lisp*: a configuration file that contains default values of global variables and system dependent path information to the compiler and log files.
- *Files.list*: contains paths to all the files present in the web-interface folder. This file is included in the make file that is used to compile the overall GTA package.
- *global-variables.lisp*: contains a list of global variables that are used in the solver.
- *load-gta.lisp*: contains functions to load the GTA package, initialize the CSP, and run consistency algorithms.
- *makecsp.lisp*: the start up file that compiles and loads the CSP.
- *misc-funcs.lisp*: contains miscellaneous functions that are used in other functions.
- *savecsp.lisp*: contains functions to save the current assignments in the database.

- *showcourses.lisp*: contains functions to send the course and corresponding GTA information when PHP scripts (GTAs to Class assignment in the web-interface) send requests.
- *showgtas.lisp*: contains functions to send the GTA and corresponding course information when PHP scripts (Class to GTA assignment in the web-interface) send requests.
- *sock-listener.lisp*: contains implementation of the Lisp-based server.
- *startdaemon.pl*: a perl script to start the interactive solver as a daemon process using the 'detachtty' utility present in the directory.
- *testcsp.lisp*: contains the default values values of global variables and system-dependent path information (as described in configure-load.lisp file) and also calls various functions to initialize the CSP and load GTA data. This file can be used to test if the data is loading as expected.
- *testserv.lisp*: contains calls to various functions to initialize the CSP, load GTA data and start the server. As explained in Chapter 3, the detachtty utility is used to run the GTA daemon process as a background process. As a result, it may not be possible to view run-time errors when the manager makes an interactive selection. This file is used to start the server as a foreground process. This allows the user to view various Lisp function calls whenever the user interacts with the web interface.

A.2 Data structures

In this section we describe the data structures used to store the assignments and history of assignments/unassignments.

A.2.1 Assigned Courses

All the course assignments are temporarily stored in a structure called `*assigned-courses*` before they are permanently stored in database. The structure is described as follows:

```
*assigned-courses* = ((g-1 c-1)
                       (g-2 c-2)
                       ...
                       (g-n c-n)
                       )
```

The list, `(g-i c-i)` is maintained as a GTA course pair where `g-i` is the GTA who is assigned to course `c-i`. An example of the list is,

```
(("Mary" 105-150) ("John" 101L-001))
```

This structure is mainly used when a course is unassigned.

A.2.2 Log of Assignments/Unassignments

Every time a course is assigned or unassigned a text message indicating the action is stored in a list called `*messages*`. This list is used to store the history of actions for future reference. The structure of the list is as follows:

```
*messages* = (m-1 m-2 ... m-i ... m-n)
```

Here, `m-i` is the text string and the overall structure holds all the text strings. The order of the messages is also maintained. An example of the list is,

```
("John is assigned to course Computer Science Fundamentals"
    "Computer Problem Solving is unassigned")
```

This structure is accessed by PHP scripts to display the log of user actions on the visual interface.

A.2.3 Communication with PHP scripts

The variable, `*php-stream*` is used for communication between PHP scripts (web-interface) and Lisp code (interactive solver). This variable is defined as a socket stream in the file sock-listener.lisp file. Any information written over to this variable from Lisp environment will be sent to PHP scripts through TCP/IP protocol. For example, the following print message in Lisp code sends the string, ‘Test message: 10’ sends to PHP script.

```
(format *php-stream* "Test message: ~a " 10)
```

A.3 Variables and functions

In this section, we review the functions and the internal function calls in the system. The variables used in the interactive solver are described in Table B.1. The following sections list the functions present in each file.

A.3.1 ac.lisp

ac-1 *csp* [*FUNCTION*]

Implementation of arc-consistency algorithm.

revise *vari varj constr* [*FUNCTION*]

Used by the arc-consistency algorithm to filter inconsistent GTAs.

get-constraints *csp* [*FUNCTION*]

Used by the arc-consistency algorithm to retrieve binary constraints in a given CSP such that the variables involved in the constraint are not assigned and the domain of any variable is not empty.

copy-current-domains *csp* [*FUNCTION*]

Used during the initialization phase(after executing the arc-consistency algorithm) to copy the arc-consistent GTAs in the current-domain to the initial-domain.

get-all-vars-bin-constraints *csp* [*FUNCTION*]

Used by `copy-current-domains` function to retrieve all the binary constraints in a given CSP.

A.3.2 loadgta.lisp

execute-func *function-call* [*FUNCTION*]

Executes a given function and returns error message if the function call fails. Used as a wrapper function to check errors so that the system does not crash.

load-initialize-csp *stream* [*FUNCTION*]

Creates a CSP and initializes it with data from current year and semester.

check-year-sem *year semester* [*FUNCTION*]

Checks if the `*current-csp*` is of a particular `year` and `semester`.

A.3.3 misc-funcs.lisp

get-gta *gta-name* [*FUNCTION*]

Retrieves the GTA object, given GTA name.

get-course *key* [*FUNCTION*]

Retrieves the course object, given a course symbol.

A.3.4 savecsp.lisp

get-is-id *csp-name* [*FUNCTION*]

Retrieves the actual database generated id (unique) corresponding to the given user-friendly csp name.

save-current-csp *stream name* [*FUNCTION*]

Saves the current CSP into the database with a given name.

load-assigned-courses *stream name* [*FUNCTION*]

Loads all the previously assigned courses, from the variable `*assigned-courses*`, into the `*current-csp*`.

make-current *stream csp-name* [*FUNCTION*]

This method creates a CSP `*current-csp*` and makes the assignments saved in the database.

clear-saved-assignments *stream* [*FUNCTION*]

Clears the assignments (in the current year and semester) saved in the database.

clear-all-saved-assignments *stream* [*FUNCTION*]

Clears all the assignments stored in the database.

get-saved-csps *stream* [*FUNCTION*]

Retrieves all the CSPs that are stored in the database (This function is called from the PHP script).

A.3.5 showgtas.lisp

save-course-assignment *gta-name course-num-section* [*FUNCTION*]

Saves the GTA assignment to a given course in the `*assigned-courses*` structure.

get-assigned-courses *gta-name csp* [*FUNCTION*]

Retrieves all the courses to which a GTA is assigned.

get-possible-courses *gta-name csp* [*FUNCTION*]

Retrieves all the courses in the domain where the GTA is present.

get-filtered-courses *gta-name csp* [*FUNCTION*]

Retrieves all the courses from the domains where the GTA is filtered.

unassign-gta-courses *gta-name csp* [*FUNCTION*]

Unassigns the given GTA from all the courses to which he/she is assigned.

display-gtas *stream csp* [*FUNCTION*]

Displays all the GTAs and the courses in which the GTAs are present. The function is called from PHP scripts.

A.3.6 showcourses.lisp

print-gta-weights *stream domain course* [*FUNCTION*]

Retrieves and sorts all the GTAs present in a course domain.

display-courses *stream csp* [*FUNCTION*]

Retrieves all the courses and the GTAs present in the course domain.

get-courseobj *number section my-csp* [*FUNCTION*]

Retrieves a course object given a course number and section.

get-rem-capacity *gtaobj* [*FUNCTION*]

Obtains the current remaining available capacity of a GTA.

do-assignment *gta course csp* [*FUNCTION*]

Assigns a GTA to a course (does not do any constraint checks). This function is called by `assign-gta` function.

un-do-assignment *course-var csp* [*FUNCTION*]

Unassigns a course variable.

assign-gta *stream gtaname course csp* [*FUNCTION*]

Assigns a GTA to a course in the given CSP.

get-messages *stream* [*FUNCTION*]

Returns the log of assignments/unassignments from `*messages*` structure. This method is called from PHP scripts.

check-assignment *stream course csp* [*FUNCTION*]

Checks if the given course is assigned with a GTA.

A.3.7 sock-listener.lisp

start-server *portfile logfile port* [*FUNCTION*]

Starts the interactive solver as a server on a given port and outputs all function calls into a logfile.

writeport *port filename* [*FUNCTION*]

Outputs the port number on which the server is started to a given filename.

writetime *filename* [*FUNCTION*]

Outputs the current time stamp to a given file that is accessed by PHP scripts to know the time at which the server is started.

write-to-log *filename input output* [*FUNCTION*]

Writes all the function calls made by PHP scripts into a log file.

do-command *sock-stream logfile* [*FUNCTION*]

Reads and executes a function called (from PHP script), as a string, from a socket stream and saves it to a logfile using `write-to-log` function.

A.3.8 Function calls

In this section we pictorially show the internal function calls (to and from other functions) in the system.

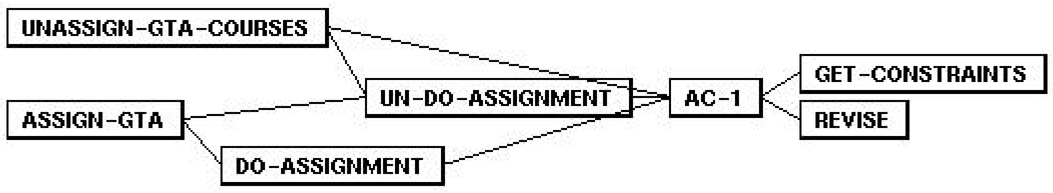


Figure A.1: Function: ac-1.

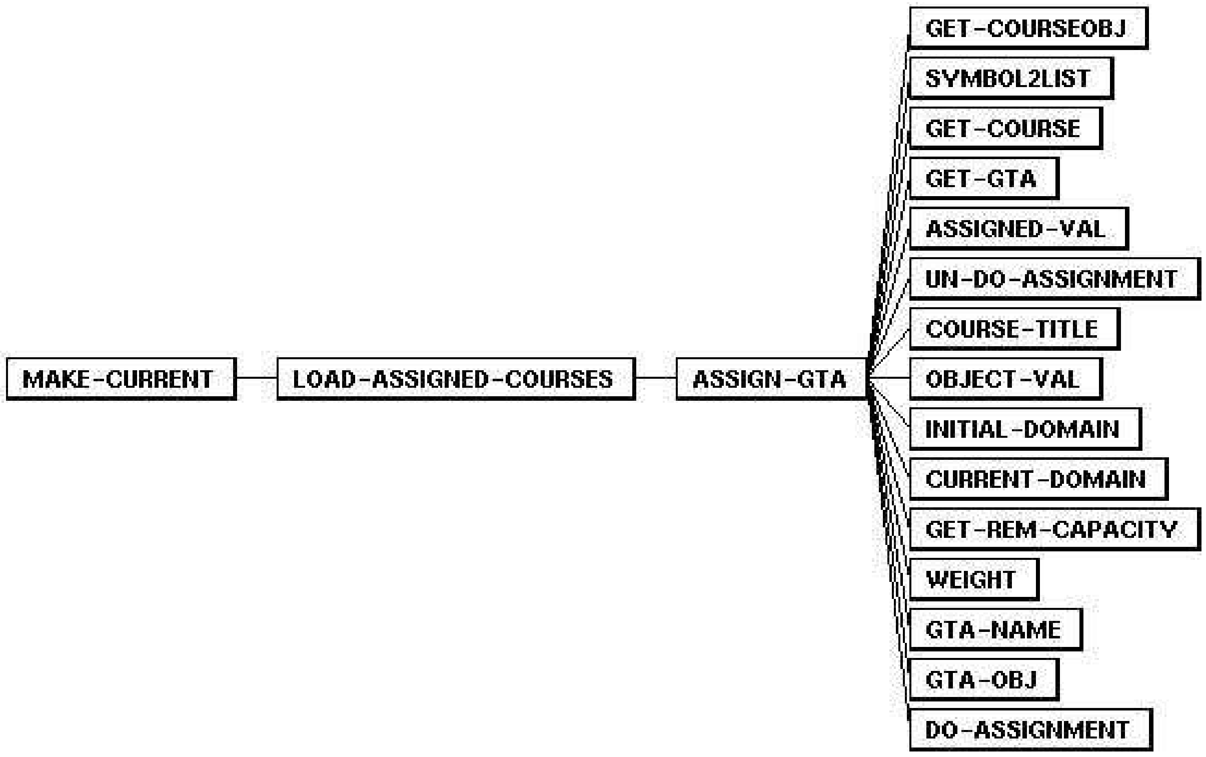


Figure A.2: Function: assign-gta.

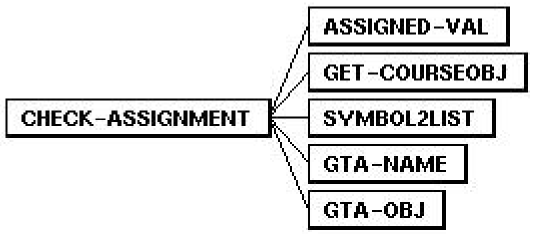


Figure A.3: Function: check-assignment.

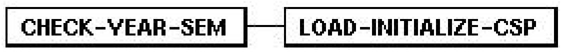


Figure A.4: Function: check-year-sem.

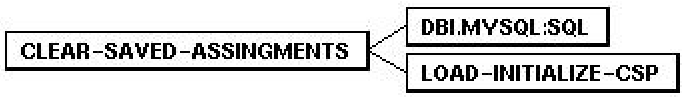


Figure A.5: Function: clear-saved-assignments

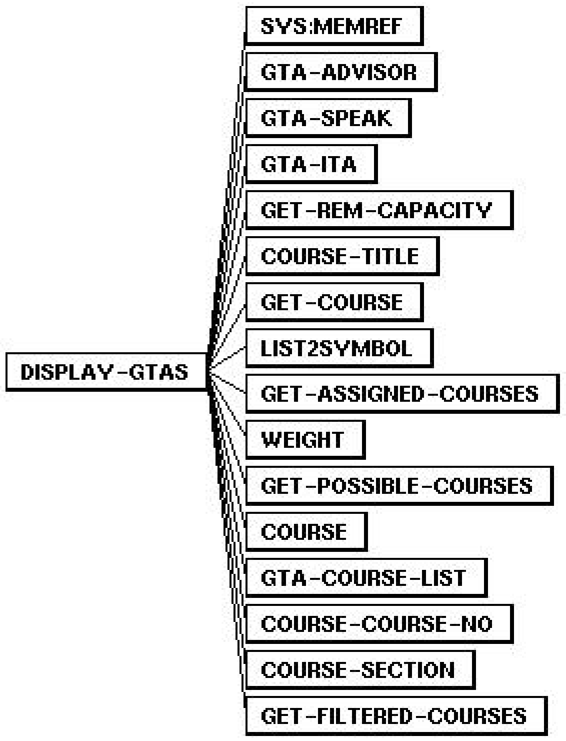


Figure A.6: Function: display-gtas

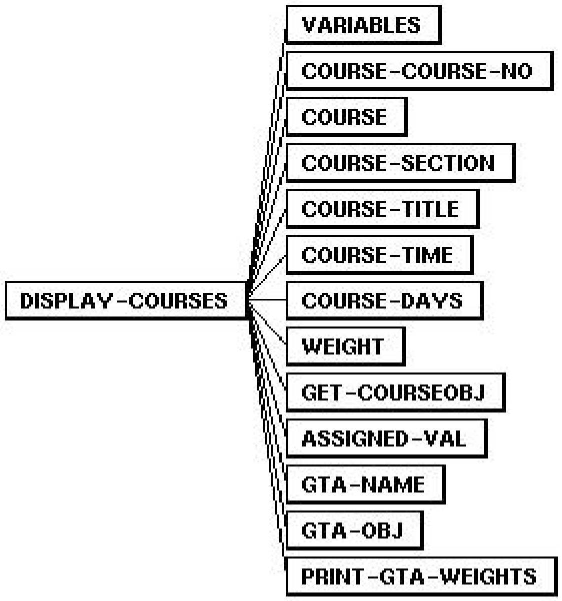


Figure A.7: Function: display-courses

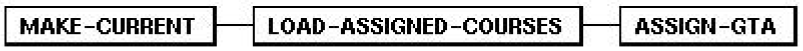


Figure A.8: Function: load-assigned-courses

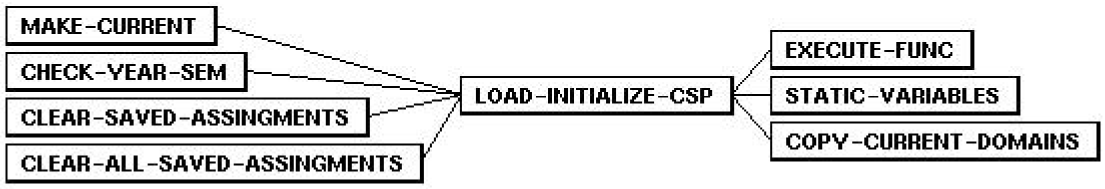


Figure A.9: Function: load-initialize-csp

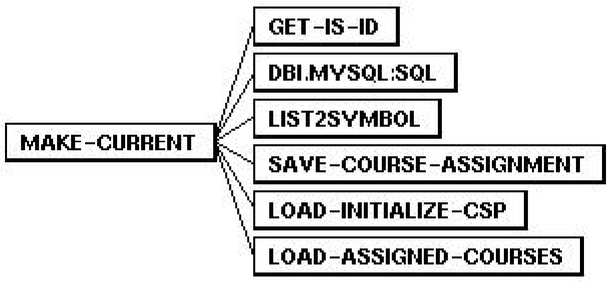


Figure A.10: Function: make-current

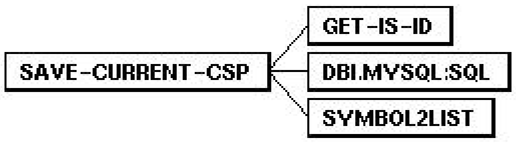


Figure A.11: Function: save-current-csp

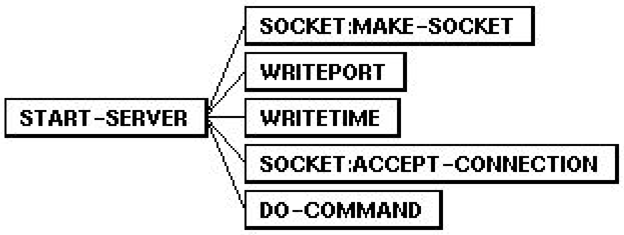


Figure A.12: Function: start-server

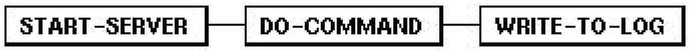


Figure A.13: Function: do-command

Appendix B

Communication between the web interface and the interactive solver

The manager can connect to the interactive solver through web interface using a browser (or any program that can connect to interactive solver and parse the data). At any time, the manager on the client (browser) side initiates the request. When the web interface (PHP script) is accessed, it sends a request to the interactive solver (daemon process) for the current updated list of courses and the corresponding GTAs. The solver sends the information back to the web interface, including the assignments. The web interface parses the information and renders the output as an HTML page. The series of events can be summarized as:

On server side (`cse.unl.edu`):

1. Initialize the GTA system (see Section 2.2).
2. Start the socket listener on a port number (The starting default port number is specified in configuration file `configure-load.lisp`. The socket listener module searches for next 100 ports for an available port and starts the server).
3. Make this process run in the background using the utility 'detachtty.'
4. The socket listener stores the port number, on which the server is started, in a temporary file (e.g., lispport.txt).

On client side (browser):

1. The browser sends a request to web server for the PHP script.
2. The PHP script checks the port file (lispport.txt) to know the port number on which the socket listener is running.
3. Using this port number, the PHP script connects to the socket listener and sends the Lisp commands (see Table 3.3).
4. The socket listener executes the commands and sends the output back to the PHP script.
5. The PHP script reads the output and generates an HTML page.

As explained in Sections 3.2 and 3.3, the web interface is implemented in PHP and the interactive solver is implemented in Lisp. The two environments communicate through the socket listener using TCP/IP protocol. The socket-listener interface, built inside the Lisp daemon process, listens to connections on a predefined port (the default port number is 9000 stored in file `configure-load.lisp`). The port number is obtained from the text file `lispport.txt` shared on the server by Lisp and PHP scripts.

The PHP scripts connect to this port and send the Lisp commands, defined in Table 3.3. The global variables (by convention, asterisks are used around the names of global variables) used in these commands are listed in Table B.1. The functionality of these Lisp commands is explained in Chapter 2.

Upon initialization, the interactive solver loads the Lisp command environment into system memory and the socket listener interface present in the solver accepts and executes any Lisp function call sent by the PHP scripts. Since the commands are already in Lisp function format, the socket listener in the daemon process simply executes the functions on Lisp command prompt and sends any resulting output back to the PHP script.

A manager's action in the web interface is converted into a Lisp function call (the format is '`(function-name arguments)`') and is sent to the interactive solver. The solver executes the function on its command prompt and sends any output back to the web interface. The PHP scripts do the required conversion of the manager's action. The PHP scripts maintain a list of Lisp functions and variable names for each action. The scripts also know what kind of output to expect after the execution of each command. While some of the commands return a simple true or false value,

Table B.1: Global variables in Lisp environment.

Variable	Description
`*current-csp*`	CSP that holds all the courses, domains, and constraints.
`*php-stream*`	Acts as a communication medium between PHP and the Lisp environments using sockets. Any data written to this stream variable, from Lisp code, will be sent to the PHP script.
`*gta-year*`	Holds the current GTA year (e.g., 2002 and 2003).
`*gta-semester*`	Holds the current GTA semester (e.g., 1-Spring, 2-Fall, 21, and 22-Summer courses)
`*assigned-courses*`	This variable is used internally in Lisp data structures. It holds the courses that are assigned to courses. The internal data structure is explained in Appendix A.2.
`*messages*`	Holds the manager's actions as a list of strings. The internal data structure is explained in Appendix A.2.

others return a large amount of data. After the command is executed, the PHP script again sends request for the updated GTA and course information. The PHP script parses this data, generates an HTML page and sends it back to manager's browser. Figure B.1 shows the overall sequence of actions when the manager selects a GTA from the drop-down menu for a course. The numbers before text indicate the

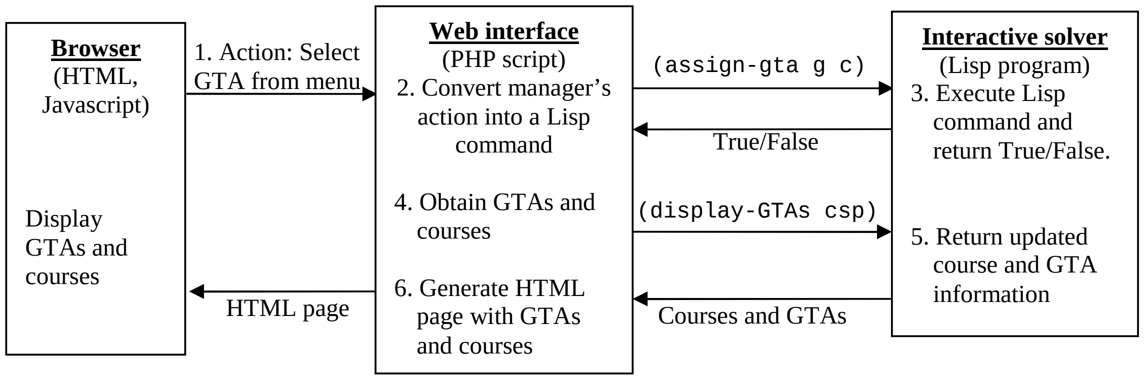

Browser
(HTML, Javascript)
Display GTAs and courses
1. Action: Select GTA from menu
HTML page
Web interface
(PHP script)
2. Convert manager's action into a Lisp command
4. Obtain GTAs and courses
6. Generate HTML page with GTAs and courses
(assign-gta g c)
True/False
(display-GTAs csp)
Courses and GTAs
Interactive solver
(Lisp program)
3. Execute Lisp command and return True/False.
5. Return updated course and GTA information

Figure B.1: Conversion of manager's actions into Lisp function calls.

step number. The manager initiates the overall process by selecting a GTA from the drop-down menu. In Step 1, the manager's action is sent to PHP script from HTML as a post method. In Step 2, the script creates the Lisp command '`(assign-gta *php-stream* c g *current-csp*)`' (abbreviated in the figure) and sends it to the interactive solver. In Step 3, the interactive solver executes the command and returns the output as a true value '`t`' or false value '`nil`'to indicate that the assignment was a success/failure. The PHP script again sends a request to the solver to obtain GTA and course information to by sending the Lisp command '`(display-GTAs *php-stream* *current-csp*)`' (abbreviated in the figure). The PHP scripts parse the data from the interactive solver in Step 6 and send the HTML page back to the manager's browser for further interaction.

Lisp commands that retrieve data other than true or false are commands for 'Assignment of courses to GTA' or for 'Assignment of GTAs to courses.' When a manager requests (usually, a button is pressed on the web interface) a display of courses and corresponding GTAs in their domain, the PHP script interprets this as command:

```
(display-courses *php-stream* *current-csp*)
```

and sends it to the solver for execution. The solver executes the command and retrieves all the courses and GTAs present in their domain from the CSP. The retrieved course information contains the course number, course section, course title, timings, course load and GTAs present in the course domain. For each GTA, the name, course preference and available capacity are retrieved. The course and GTA information is sent back to the PHP script in the following format:

The information for each course is enclosed in '`%`' symbols. If `course-i` indicates course '`i`' then the Lisp functions format the overall information of all the courses as:

```
%course-1%%course-2%%course-3%...%course-n%
```

The string `course-i` contains the overall course and GTA information, separated by '`@`', '`;`' and '`=`' symbols. The course string, `course-i` can be further divided by '`=`' symbol as:

```
%Course-information=Assigned-GTA=current-domain=initial-domain%
```

'`Assigned-GTA`' is the GTA information string that a course might already have been assigned during interactive assignments. The GTAs in current and initial domains are separated by the '`;`' symbol and the GTA information is separated by the '`@`' symbol. The course string is thus formed as:

```
course-i = %Course-information=CD-1;CD-2;...;CD-p=ID-1;ID-2;....;ID-q%
```

where, `CD-i` and `ID-j` represent the GTAs present in the current and initial domains of the course respectively. The initial domain is the domain of the course before any assignments are made, and the current domain is the filtered domain of the course after the assignments are made. The set difference between the initial and current domains gives the GTAs who are busy (i.e., GTAs who are assigned to other courses). The course and GTA information is further split by '`@`'. Thus, the `Course-information` string is:

```
            course-number@section@title@timings@days@load
```

and the `GTA-info` string is:

```
            course-preference@available-capacity@GTA-name
```

The overall course string can be summarized as:

```
course-i = %course-number@section@title@timings@days@load=gta-pref@
            available-capacity@gta-name;gta-pref@available-capacity@
            gta-name;.......;%
```

If a particular course is not assigned with a GTA, then the string is:

```
  course-i = %Course-information=current-domain=initial-domain%  or
  course-i = %number@section@title@timings@days@load=preference@
                 available-capacity@gta-name;....;%
```

Also if a course is assigned with a fixed GTA, then the string need not contain the domain information. The string is represented as:

```
  course-i = %Course-information=Assigned-GTA%   or
  course-i = %number@section@title@timings@days@load=Assigned-GTA-name%
```

An example of course string is:

```
%101@002@Lab@1330-1520@W@0.25=4@0.75@Nicholas = 4@1@Nate;
 4@0.66@Tim;4@0.75@Nicholas=4@1@Nate;4@0.66@Tim;4@0.75@Nicholas;
 4@0.20@Baucher;4@0.15@;Mary;%
```

If the above string is separated as

```
s1 = 101@002@Lab@1330-1520@W@0.25
s2 = 4@0.75@Nicholas
s3 = 4@1@Nate;4@0.66@Tim;4@0.75@Nicholas and
s4 = 4@1@Nate;4@0.6@Tim;4@0.75@Nicholas;4@0.2@Baucher;4@0.15@;Mary;
```

then , the string `s1` represents the course information and it is interpreted as course number `101`, the section number is `002`, the course is type '`Lab`', the timings are between '`1330-1520`' on '`W`' (Wednesday) with a course load of `0.25`. The next string, `s2` represents a GTA assigned to the course. It is interpreted as '`Nicholas`' has a capacity of `0.75` and his preference towards the course is `4`. The strings `s3` and `s4` are the GTAs present in current and initial domains. It can be observed that all GTAs who are not in current domain i.e. '`4@0.20@Baucher;4@0.15@;Mary`', have a capacities of 0.20 and 0.15 which is less than the course load of 0.25.

For displaying a GTA and corresponding compatible courses, the Lisp functions are written to output the data to PHP scripts in the following format:

Here also the information for GTAs is enclosed in '`%`' symbols as

```
%GTA-1%%GTA-2%%GTA-3%......%GTA-n%
```

Here, `GTA-i` is a string that contains overall GTA and their course information, separated by the '`@`', '`;`' and '`=`' symbols. The course string, `GTA-i` can be divided by the '`=`' symbol in two formats. For GTAs who are by pre-assigned (i.e., by default assigned to some courses prior to initialization), the format is

```
GTA-i = %GTA-info=!pre-assigned-courses%
```

and for other GTAs the format followed is

```
GTA-i = %GTA-info=Assigned-courses=Available-courses=Ruled-out-courses%
```

The string '`GTA-info`', contains the GTA name,GTA's Advisor, Speech Test ('`T`' or '`NIL`' to indicate whether the GTA has taken the Speech Test), ITA Certification ('`T`' or '`NIL`' to know whether the GTA is ITA certified) and GTA Capacity. The information is separated by the '`@`' symbol as,

```
GTA-name@Advisor-name@Speech-Test@ITA-Certification@GTA-Capacity
```

The string '`Assigned-courses`' contains the list of courses that are assigned to a GTA. The courses are separated by a '`;`' symbol and each course information is further separated by a '`@`' symbol as

```
    course-name@course-preference@number@section@course-load
```

Similarly, the strings '`Available-courses`' and '`Ruled-out-courses`' have the same format as explained above. The string '`Available-courses`' contains the list of courses in which the GTA is present and '`Ruled-out-courses`' contains courses from which the GTA is filtered. An example of a GTA string is,

```
%Nicholas@Mary@T@NIL@0.5=Lab@5@105@151@0.25;
 Graph Algorithms@5@924@001@0.25=Comp. Sci. Fundametals@5@101@002@0.5;
 Data Structures and Algorithms@4@310@150@0.5=CSP@5@421@821@1%
```

This string can be separated as

```
GTA-info          = Nicholas@Mary@T@NIL@0.5$
Assigned-courses  = Lab@5@105@151@0.25;Graph Alg@5@924@001@0.25
Available-courses = Computer Science Fundamentals@5@101@002@0.25;
                    Data Structures and Algorithms@4@310@150@0.5;
Ruled-out-courses = CSP@5@421@821@1
```

The `GTA-info` string indicates that the GTA, '`Nicholas`' whose advisor is '`Mary`' has taken speech test ('`T`' value) and does not have ITA certification ('`NIL`' value). He has an available capacity of 0.5. The '`Assigned-course`' string indicates that '`Nicholas`' is currently assigned to two courses, '`Lab`' and '`Graph Algorithms`'. The '`Available-courses`' string indicates that the GTA can still be assigned to either '`Computer Science Fundamentals`' or '`Data Structures and Algorithms`'. These two courses have loads of 0.25 and 0.5, respectively, which are less than or equal to the capacity of the GTA. The course '`CSP`' is ruled-out because its course load is 1.