OPRAM: AN ONLINE SYSTEM FOR ASSIGNING CAPSTONE COURSE STUDENTS TO SPONSORED PROJECTS

by

Juedong Zhang

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

May, 2014

OPRAM: AN ONLINE SYSTEM FOR ASSIGNING CAPSTONE COURSE STUDENTS TO SPONSORED PROJECTS

Juedong Zhang, M.S.

University of Nebraska, 2014

Adviser: Dr. Berthe Y. Choueiry

OPRAM (Online Project Assignment Manager) is a web-based decision support system built to facilitate project assignment in the Capstone Course in the Department of Computer & Science Engineering at the University of Nebraska-Lincoln. The system aims to liberate its users from tedious but often error-prone tasks by providing two solvers at their disposal: one automatic and the other interactive. Both solvers are based on Constraint Processing, but operate in different modes. The automatic solver runs as a server process, and generates, upon request, a solution based on an optimization criterion. The interactive solver runs on the client side, and works with the user to find the solution in a collaborative fashion. The system also incorporates a graphical web interface to support smooth user interactions. Preliminary results have demonstrated our systems capability to significantly improve the time to solution as well as increase the quality of the assignment found.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Berthe Y. Choueiry, without whom this project would not exist. I am grateful for her sage advice, insightful criticisms, and patient encouragement.

I would also like to thank Dr. Hongfeng Yu who helped enormously in the design of user interface in this project.

My thanks and gratitude to Dr. Myra B. Cohen, Dr. Matthew B. Dwyer, Dr. Ying Lu, Dr. Byrav Ramamurthy, Mr. Casey Tubbs and Mr. Charles Daniel for their cooperation throughout this project.

Lastly, but certainly the greatest, I would like to thank my family and friends for all their love and support.

This work was partially supported by the NSF Grant No. RI-111795.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This report describes OPRAM (Online Project Assignment Manager), a web-based decision support system that we have designed and implemented to help the capstone course instructors solve the Capstone Course Project Assignment Problem in the Department of Computer Science & Engineering (CSE) at the University of Nebraska-Lincoln (UNL). The task in OPRAM is to assign senior-standing undergraduate students to work on projects sponsored by faculty members or industry partners. The objective is to find an assignment that allocates students to projects among their top choices while ensuring that they are also the candidates preferred by the project sponsors. A valid assignment also has to satisfy the capacity limit constraints imposed on each project.

The idea of this project was inspired by the Constraint Processing course that I took in Fall 2012. In Spring 2013, I developed and tested the automatic solver. In the same semester, I also built a prototype of the interactive solver. This solver was further refined with the concepts and knowledge I gained from the Data Visualization course, which I took in Fall 2013. In early 2014, I integrated both solvers, giving shape to the unified system of OPRAM.

In this chapter, we describe the background of the Capstone Course Project Assignment Problem, state our motivations for this project, summarize our contributions and conclude the chapter with the organization of this report.

1.1 Background

As an integral part of the undergraduate curriculum, CSE offers the senior-standing students professional development opportunities (a.k.a., the capstone course) by allowing them to work on real-world engineering projects under the supervision of the faculty. The projects are often sponsored by business or institution agencies in partnership with CSE. Each semester, students are organized into teams and each team works with a project sponsor to develop a software or hardware based solution tailored to the sponsor's needs.

The task of assigning the right students to the right projects is demanding. On the one hand, the students may prefer some projects over others based on their major, technical abilities, or personal interests. On the other hand, the sponsors who devote time and resources into the projects may want to select desirable candidates by evaluating their qualifications such as grade point average (GPA) or prior development experience. They may also want to set a limit on the maximum number of students allowed to participate in their projects.

At the beginning of a semester, each sponsor gives a project pitch in front of the class. The students are then asked to apply for the projects by submitting their resumes and cover letters. For each project applied, the student is also required to indicate his/her preference[1] for the project. Following the students' applications, project sponsors review the students who have applied for their projects and indicate

[1]Details on the preference scale and modeling are discussed in Chapter 2.

their own preferences for each of the applicants. Finally, after the sponsor review, the capstone course instructors assigns the students into teams based on the preferences and the project capacity limits. Figure 1.1 illustrates the sequence of events during the capstone course team formation process.

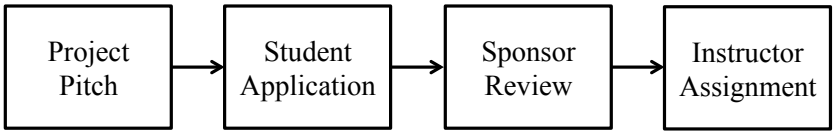

Figure 1.1: The event sequence of team formation in the Capstone course.

1.2 Motivations

The last event in the team formation process (Figure 1.1) is often challenging and can take a considerable amount of time and effort. In a typical semester, the course instructors need to assign between 10 to 45 students to approximately 3 to 11 projects[2]. Currently, this task is done manually with assignment information such as project capacities, student, and sponsor preferences printed on a piece of paper or presented in a Excel spreadsheet. When the instructors decide to assign a student to a project, a consistency check needs to be performed to make sure the project is still within its capacity. When the limit is reached, the project needs to be removed from the selection 'options' of all the other students who are not assigned to it to prevent future inconsistency. Moreover, whenever the instructors change their minds and decide to unassign a student from a project, a consistency check also needs to be carried out. In case the project becomes available again, it needs to be added back to the selection domains of every students who are previously disallowed to join that project due to capacity constraint. It is obvious that this process requires a considerable amount of

[2]The statistic is according to the Fall 2013 and Spring 2014 classes.

bookkeeping, a tedious and often error-prone job for the human beings. We claim that such a task can be easily and reliably facilitated with help of a computer software.

Secondly, finding a feasible solution that guarantees no project exceeds the capacity limit does not solve the problem. In order to better serve the interests of both the students and sponsors, the course instructors also need to take into consideration their declared preferences, which we define as the *quality of a solution*. This requirement implies that the instructors (or course managers) must be able to explore various alternative solutions and compare their qualities based on a defined metric. Previous work has demonstrated the effectiveness of using interactive systems to help solve the Graduate Teaching Assistant Assignment Problem [Glaubius and Choueiry, 2002; Zou, 2003; Guddeti, 2004; Thota, 2004; Lim *et al.*, 2004; Lim, 2006][3]. The major benefit of such systems is to allow users to focus on a single task (i.e., making assignments) and release them from having to check for consistency and propagate the effect of decisions. One of the main goals of this project is to build an interactive solve that can assist the user in decision making by providing feedback and intuitive visual hints.

Like other similar problems, the computational complexity of the Capstone Course Project Assignment Problem grows exponentially with the size of the problem. We remain aware that an increase in the number of students and projects quickly render the problem exceedingly difficult, and eventually impossible, to tackle for both humans and computers. With that in mind, we developed an automatic solver that is able to generate feasible solutions (if exist) on demand.

[3]An optimization problem similar to the Capstone Course Project Assignment Problem.

1.3 Contributions

This project was an initiative that we took in an effort to help CSE in an task particularly well suited for Constraint Processing. Below, we summarize five of our key contributions, which we describe in the rest of this report.

1. We have modeled the Capstone Course Project Assignment Problem as a Constraint Optimization Problem (COP). By doing so, existing techniques and methods developed in CP become directly applicable.

2. We have designed a hybrid search algorithm that combines the Forward Checking algorithm [Haralick and Elliott, 1980] and the Branch and Bound search algorithm. Our algorithm, based on systematic search, forms the basis of our automatic solver.

3. We have developed an interactive solver, with a dual project-student view, that allows our user to easily assign a student to a project and vise versa. The solver maintains consistency while the user explores the search space for solutions.

4. We have created a web GUI interface that encapsulates both solvers and provides a unified user experience. The interface has a color-map feature to provide hints for local assignments and replacements. It also has a built-in chart feature to allow the visual comparison of the solutions qualities at the global level.

5. We have co-built a MySQL database for data persistence and warehousing at the backend. This database provides a structured storage and retrieval of student and project information. It also supports other operations such as saving and loading solutions.

The resulting system has greatly simplified the project assignment workflow and is therefore able to save a substantial amount time and effort for the course instructors. The system is an integral component of the Innovation Lab (iLab)[4] infrastructure, a life cycle management facility for the capstone course. Its capabilities include a web portal for handling student and sponsor registration and data entry, project information exchange and team progress management in addition to project assignment. At the time of this writing, the iLab infrastructure is under active development. Nevertheless, normalized the programming interfaces necessary for integration, see Appendix B for details.

1.4 Organization of the Report

The rest of this report is organized as follows. In Chapter 2, we introduce the problem model and the methods used in solving the Capstone Course Project Assignment Problem. Chapter 3 describes the architecture of OPRAM and its features. It also serves the purpose of a user reference manual. We conclude this report and state the future works in Chapter 4. Appendix A contains the results of our search methods performed on two data sets. The rest of the appendixes focus on the technical aspects of our project implementation and they are intended for maintenance and further development purposes. Appendix B gives an overview of the integration into iLab infrastructure. Appendix C documents the database design and queries. Appendix D is a documentation of our software API.

[4]Innovation Lab (iLab) is used by CSE as a marketing name for the capstone course. More information about iLab can be found at: http://cse.unl.edu/InnovationLab.

Chapter 2

Problem Modeling & Solving Methods

In this chapter, we describe the model and methods we used to solve the Capstone Course Project Assignment Problem. First, we introduce the problem model. Then, we discuss the problem solving methods: ordering heuristics, local consistency techniques and the FC-BnB search algorithm. Finally, we evaluate the performance of our methods.

2.1 Problem Modeling

In this section, we define the basic structure of a Constraint Optimization Problem (COP) and model the Capstone Course Project Assignment Problem as a COP.

2.1.1 Constraint Optimization Problem (COP)

Many decision and optimization problems can be advantageously modeled as Constraint Satisfaction Problems (CSPs) or Constraint Optimization Problems (COPs),

respectively. Both CSPs and COPs have hard constraints that must be satisfied in order for a solution to be considered valid. However, a COP also includes soft constraints that express preferences and can often be summarized in terms of an objective function to maximize or minimize. Unlike hard constraints, soft constraints do not affect the validity of a solution but they determine the solution quality. Therefore, the goal in a COP is to find the solution with the best quality based on the the objective function [Dechter, 2003a]. Formally, a COP is defined as follows:

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

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

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

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

$$\mathcal{D} = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\} \text{ such that } D_{V_i} \text{ is the domain of variable } V_i$$

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

$$\mathcal{C} = \{C_1, C_2, \ldots, C_i, \ldots, C_m\}$$

with $C_i = C_{V_a, V_b, \ldots, V_c} = \{(x, y, \ldots, z)\} \subseteq D_{V_a} \times D_{V_b} \times \ldots \times D_{V_c}$

- f an objective function over a set of variable-value assignments $\{(V_i, v_i) \mid v_i \in D_{V_i}\}$ that is consistent with the constraints

$$f((V_1, v_1), (V_2, v_2), \ldots, (V_n, v_n)).$$

Query: Find a value for each variable from its domain such that all the hard con-

straints are satisfied and the output of the objective function is optimized.

It is useful to reiterate the terminologies associated with the hard constraint and it classification. For any given constraint $C_{V_a,V_b,\ldots,V_c}$ the set of variables $V_a, V_b, \ldots, V_c$ is called the constraint's *scope* and the size of this set is called the constraint's *arity*. If the arity of a constraint equals one, the constraint is called a *unary constraint*. If the arity of a constraint is two, the constraint is called a *binary constraint*. A constraint with an arity greater than two is generally referred to as a *non-binary constraint*.

The objective function takes the set of variables as input and generates an output indicating the goodness of a feasible solution. This output is comparable among solutions and is therefore the basis for quantitative measurement of the solution quality.

2.1.2 Capstone Course Project Assignment Problem as a COP

Given the above definition of a COP, we now model the Capstone Course Project Assignment Problem as a COP. Below, we list the variables, domains, hard constraints and the objective function of this problem.

Variable The students enrolled in the capstone course are modeled as variables.

Domains The domain of a variable is the set of projects to which the student has applied.

Hard constraints We identidy two types of hard constraints in the problem:

- *Zero preference* (unary constraint): A student cannot be assigned to a project if the student preference score and/or the sponsor preference score is zero (see Table 2.1). This constraint is unary because it restrains only one variable.

- *Capacity* (non-binary constraint): The number of student assigned to a project cannot exceed the capacity limit of that project. This constraint is a *global* constraint therefore its scope contains the entire set of variables.

Soft constraints We identidy two types of soft constraints in the problem:

- *Students' preference:* For each project he/she applies for, a student expresses his/her opinion towards the statement: "*I am interested in working on this project.*" The response is then converted into a student preference score (between 0 and 5) using the scale in Table 2.1. The scale is identical

Table 2.1: Preference scale.

Score	Description
0	Absolutely Disagree
1	Strongly Disagree
2	Disagree
3	Neither agree nor disagree
4	Agree
5	Strongly Agree

to the popular Five-point Likert Scale used in many questionnaires and surveys with the exception of a new item: "0 - Absolutely Disagree". This additional option allows a student to avoid being assigned to a project should he/she have absolute no interest in.

- *Sponsors' preference:* Similarly, a sponsor has the opportunity to express his/her preference for each applicant by indicating his/her level of agreement on the statement: "*I think this student is a good fit for my project.*"

Objective function The objective function is defined upon a combination of the preferences of the students and that of the sponsors. The combined preference[1] is

[1]From this point forward, the "preference" we mention refers to the combined preference unless stated otherwise.

obtained by taking the product of both preference scores. Figure 2.1 shows a visual illustration of this conversion. The above-defined preference gives the value of a

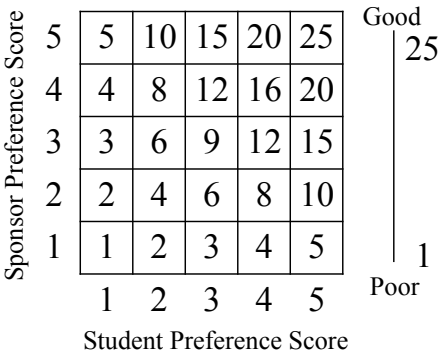

Figure 2.1: 2-D to 1-D preference conversion.

variable-value pair and the output of the objective function on a partial (or complete) solution is defined on the values of the variable-value pairs in that solution. We have considered the following optimization criteria:

1. *Maximize the average value* (Avg) of all preferences.

2. *Maximize the geometric average value* (Geo Avg) of all preferences.

3. *Maximize the minimum value* (Maxmin) of all preferences. In tems of this criterion, one solution S_1 is better than the other S_2 if the minimum preference in S_1 is larger than the minimum preference in S_2. When S_1 and S_2 have the same minimum preference, S_1 is better than S_2 if it has fewer occurrences of the minimum preference. Informally, this optimization criterion corresponds to find a solution that can make the least happy individual happier or to maximize the minimal happiness.

2.2 Problem Solving Methods

In this section, we discuss two types of ordering heuristics and two local consistency techniques we use in our backtrack-based search procedure. We then proceed to the discussion of our FC-BnB search algorithm.

2.2.1 Ordering heuristics

Heuristics ordering for the instantiation of variables and values during search are known to widely impact the performance of backtrack search. Ordering heuristics can be either static or dynamic. Dynamic ordering heuristics exploit the fact that the search space changes during search. Below, we discuss two types of dynamic ordering heuristics.

2.2.1.1 Dynamic Variable Ordering heuristic

Dynamic Variable Ordering heuristic [Bacchus and van Run, 1995] changes the order of variables in a COP during search. It attempts to reduce the branching factor by selecting the *most constrained variable first*. The heuristic operates on the *fail first* assumption, that is, the sooner it discovers the impossible variable-value combinations the quicker it will find a solution or declare the problem unsolvable.

In this project, we used the *least-domain first* implementation of the Dynamic Variable Ordering heuristic. As the name suggests, this implementation chooses the variable with the smallest domain size first. Putting that into the perspective of our problem, students who apply for less number of projects are considered first during search for assignment. Moreover, with each variable being instantiated (assigned a value), the domain of all the un-instantiated variables are filtered by forward checking to remove inconsistent values (discussed in Section 2.2.3). This change in the domain

size lead to the re-ordering[2] of the remaining variables before the instantiation of the next variable.

2.2.1.2 Dynamic Value Ordering heuristic

Dynamic Value Ordering heuristic is similar on an abstract level to Dynamic Variable Ordering heuristic except that it acts upon the values in the domain of an individual variable [Geelen, 1992]. The idea of this heuristic is that a solution is likely to be found earlier if we assign the most promising value for any given variable (*the most promising value first*).

In our problem, for a given variable (i.e., student) chosen for instantiation, we choose the value (i.e., project) with the largest preference value because we hope to improve the quality of the solution, while ignoring the capacity constraints. Like Dynamic Variable Ordering heuristic, the values are sorted in decreasing preference order after domain filtering.

2.2.2 Local consistency methods

Local consistency is a central topic of research in Constraint Processing [Dechter, 2003b]. Enforcing local consistency prior to or during search removes the values from the variables that cannot participate in a solution to the problem and reduces the search effort. The advantage of maintaining local consistency is to discover inconsistency at an early stage in the search process by detecting an empty domain (i.e., a domain *wipe out*) and therefore avoid *thrashing* search behaviors (i.e., exploring inconsistent subtrees). In the following sections, we discuss two local consistency methods in the context of the Capstone Course Project Assignment Problem. The

[2]The meaning of "dynamic."

algorithms as well as the complexity analyses of both methods can be found in the literature [Mackworth and Freuder, 1984; Dechter, 2003b].

2.2.2.1 Node consistency

Node consistency is the simplest type of local consistency. It ensures that every value in the domain of each individual variable (or node) is consistent with respect to all the unary constraints that apply to the variable. The complexity of this process is linear in the number of the variables and the domain size.

In our program, the node consistency algorithm is executed during the initialization of a problem instance. It removes values (i.e., projects) with a zero preference from the domain of each variable (i.e., each student's). The definition of a *zero preference* constraint is provided in Section 2.1.2.

2.2.2.2 Generalized arc consistency (GAC)

Generalized Arc Consistency is defined over non-binary constraints. It guarantees that every value in the domain of a variable can be extended to a consistent assignment over all the remaining variables in the scope of each constraint in the problem [Dechter, 2003b].

The *capacity constraints* in our problem are non-binary constraints and they are checked by our GAC algorithm throughout the search. Our GAC algorithm is not a generic one, but specialized to the capacity constraints. Whenever the load of a project is reached, the algorithm removes this project from the domain of all future variables, therefore guaranteeing that the constraints are satisfied. The GAC algorithm is the foundation of our interactive solver, and is executed after each assignment that a user makes.

2.2.3 FC-BnB search algorithm

In this section, we first introduce the Forward Checking method and our Branch and Bound search method with a heuristic function h. We then describe the hybrid algorithm FC-BnB we developed to solve the Capstone Course Project Assignment Problem.

2.2.3.1 Forward Checking (FC)

The Forward Checking (FC) algorithm [Haralick and Elliott, 1980] is a *lookahead* search technique. The idea is to quickly remove from the domains of the uninstantiated variables the values that are not consistent with the current path, thus saving the exploration of fruitless branches. When the algorithm makes a trial instantiation of a variable, it looks ahead at all future variables, and removes from their domains all the values that are incompatible with the trial instantiation.

When lookahead causes a domain wipe out of one or more future variables, search discovers inconsistency even before the corresponding variable is instantiated, which yields significant reduction of search effort.

FC performs more work in terms of consistency checks at the node level in the tree compared with *backtracking* (BT), but it cannot visit more nodes [Kondrak and van Beek, 1995]. In general, this additional effort leads to an improvement of search performance.

2.2.3.2 Branch and Bound with heuristic function h

The Branch and Bound (BnB) algorithm is a general technique for finding the optimal solution of various optimization problems. It systematically enumerates all candidate solutions and discards fruitless candidates by comparing them with the incumbent

(i.e. the current best solution). Our branch and bound method is guided by a heuristic function that estimates the cost of solving the problem from the current node in the tree, where the cost represents the 'quality' of a solution. The cost of a candidate solution is evaluated by the function f, which has two components: the cost function g and the heuristic function h. The function f estimates the quality of the current assignments up to the current variable i, and the h function estimates the quality of the assignments of the future variables.

$$f((V_1, v_1), (V_2, v_2), \ldots, (V_i, v_i), (V_{i+1}, v_{i+1}), \ldots, (V_n, v_n)) = \tag{2.1}$$
$$g((V_1, v_1), (V_2, v_2), \ldots, (V_i, v_i)) + h((V_{i+1}, v_{i+1}), (V_{i+2}, v_{i+2}), \ldots, (V_n, v_n))$$

The function h is an admissible heuristic if and only if it never underestimates the cost of the future assignments. In other words, h provides sufficient information about the remaining assignments that, when combined with the knowledge of the current path, the algorithm is able to determine whether it is worthwhile to continue exploring that future path or to safely prune it.

The benefit of our BnB method is straightforward: it prevents exploring sub-trees with poor solution quality, thus, reducing the size of the search space. The better the estimate of h, the larger the saving is.

2.2.3.3 FC-BnB in OPRAM

FC and BnB can naturally be combined to yield a hybrid search strategy, which we call FC-BnB. In essence, FC-BnB allows the consistency state information to be shared between the task that searches for valid solutions (FC) and the task that looks for the optimal solution (BnB). This sharing of information enables the algorithm to react immediately once it discovers inconsistencies on either hard or soft constraints.

Our FC-BnB algorithm adopts Prosser's implementation of the Forward Checking algorithm [Prosser, 1993]. We added in our BnB algorithm to handle the soft constraints. Our discussion below focuses on the BnB algorithm and is intentionally stated at a high level. Definitions of common data structures and functions can be found in Prosser's paper [Prosser, 1993].

Procedure 1 resembles Prosser's *fc-label* function. When a variable is instantiated, it first uses FC to check all the future variables against the hard (*capacity*) constraints (Lines 8–10 in Procedure 1). After consistency is ensured with the current partial assignment, it then evaluates the g and h functions. The evaluation of the g function is straightforward. The evaluation of the h function is carried out by assigning to *each* future variable the *best* value (i.e., the project with the highest preference) in its domain, disregarding all hard constraints. Because it disregards the hard constraints, the problem is relaxed, and we can easily prove that this h function is admissible.

If the estimated quality is better than the quality of the best solution obtained so far, the *heuristic-check-forward* subroutine (Line 5 in Procedure 1) sets the consistency state to true, and the search proceeds to the next variable (Line 18 in Procedure 1). Otherwise, the consistency state is set to false, and the search removes the current value from the domain of the current variable and tries the next value. This procedure also terminates (Line 20 in Procedure 1) after the domain *wipe out* of the current variable.

Procedure 2 gives a global description of the loop controlling the FC-BnB search algorithm. Unlike its counterpart (i.e., Prosser's *bcssp* function), this procedure finds "all"[4] the solutions in the order of incremental quality. When a new solution is

[3]This function is originally named *check-forward* by Prosser. We renamed it here to differentiate it from our *heuristic-check-forward* function.

[4]"All" is in quotation marks because our BnB algorithm is designed to intentionally omit other feasible solutions of inferior quality. Therefore, a new solution found is guaranteed to be better than the existing one.

Procedure 1 FC-BnB-Label.

```
1: procedure FC-BnB-label(i, consistent): INTEGER
2:     consistent ← false
3:     for v[i] ← each element of current-domain[i] while NOT consistent do
4:         if consistent then
5:             consistent ← heuristic-check-forward(i)        ▷ check soft constraints
6:         else
7:             consistent ← true
8:             for j = i + 1 to n while consistent do
9:                 consistent ← FC-check-forward(i, j)[3]     ▷ check hard constraints
10:            end for
11:        end if
12:        if NOT consistent then
13:            current-domain[i] ← remove(v[i], current-domain[i])
14:            undo-reductions(i)
15:        end if
16:    end for
17:    if consistent then
18:        return i + 1
19:    else
20:        return i
21:    end if
22: end procedure
```

found, the procedure updates the best solution *v_best*, and proceeds to find the next one by forcing the algorithm to backtrack (Lines 14–16 in Procedure 2). The procedure terminates after exploring the entire space, either returning the best solution or declaring the problem impossible (i.e., no solution can satisfy the hard constraints). In addition, we can also modified the procedure to terminate on an external condition (e.g., CPU time limit) if our user does not want to wait for the optimal solution.

[5]This function is virtually identical to Prosser's *fc-unlabe* function but, since we traverse the entire search space, it adds a conditional check on the index i to make sure it does not go below the bound (i.e., $i \not\leq 0$).

Procedure 2 FC-BnB Main.

```
1: procedure FC-BnB(n, status): ARRAY
2:     v_best ← nil
3:     consistent ← true
4:     status ← unknown
5:     i ← 1
6:     while status ≠ impossible OR best_solution do
7:         if consistent then
8:             i ← FC-BnB-label(i, consistent)
9:         else
10:            i ← FC-BnB-unlabel(i, consistent)[5]
11:        end if
12:        if i > n then                          ▷ a solution is found
13:            status ← solution
14:            v_best ← v
15:            i ← i − 1
16:            consistency ← false
17:        else if i = 0 then                     ▷ the entire space is traversed
18:            if status = unknown then
19:                status = impossible
20:            else if status = solution then
21:                status = best_solution
22:            end if
23:        end if
24:    end while
25:    return v_best
26: end procedure
```

2.3 Evaluation of Search Performance

We tested our approach on two real class data sets for Fall 2013 and Spring 2014, see Table 2.2. Running on a Macbook Pro with quad-core 2.7 GHz Intel i7 processor and 16 GB of RAM, our results are shown in Table 2.3. The details about the search performance relative to the optimization criteria for both data sets are provided in Appendix A. Below, we report our main observations.

Table 2.2: Description of the two data sets collected.

	Fall 2013	**Spring 2014**
Number of variables (students)	45	10
Total number of values (projects)	11	3
Average domain size	4	2
Maximum domain size	7	3
Minimum domain size	2	2

Table 2.3: CPU time for exhautive search.

	Avg	*Geo Avg*	*Maxmin*
Fall 2013			
Without h	$>$ 30 min	$>$ 30 min	$>$ 30 min
With h	1.5 min	2.1 min	20 sec
Spring 2014			
Without h	27 ms	27 ms	39 ms
With h	10 ms	10 ms	15 ms

- *Impact of the h function.* Using the heuristic function h significantly improves the performance of search, and allows us to traverse the entire search space for Fall 2013, thus guaranteeing the optimal solution. Indeed, even for Spring 2014, which is a very small data set, we find the optimal solution quicker by a factor of 1.5 times under all three optimization criteria. In the larger data set (Fall 2013), the search, without h does not terminate within 30 minutes and is interrupted. The reason why the h function is so effective is that it quickly detects and prunes solutions of inferior quality. As a result, it reduces the number of nodes visited and dramatically increases the search speed.

- *Impact of the optimization criterion.* We observe that the search is much faster on Fall 2013 when using the *Maxmin* optimization criterion. Interestingly, Figure 2.2 shows that the values of *Avg* and *Geo Avg* steadily increase, within small fluctuations, during search when the optimization criterion *Maxmin* is used.

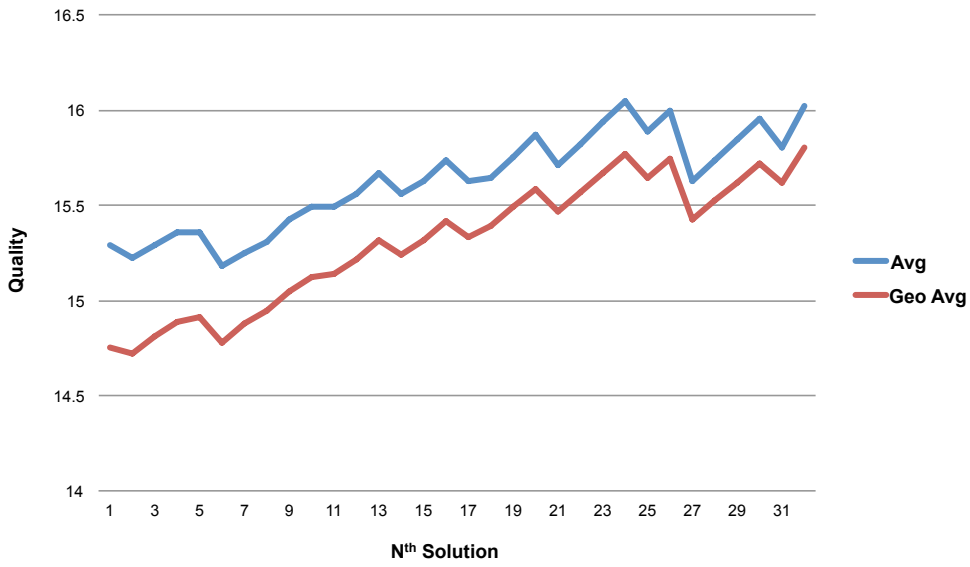

Figure 2.2: Evolution of *Avg* and *Geo Avg* with *Maxmin* optimization criterion.

In Figure 2.3, we plot the profiles of the optimal solutions found for each of the three criteria. We see that the *Avg* and the *Geo Avg* have the same optimal solution with more individuals for the higher preferences (i.e., 25, 20) but also lower preferences (i.e., 12) than the solution obtained by the *Maxmin* criterion. In conclusion, our

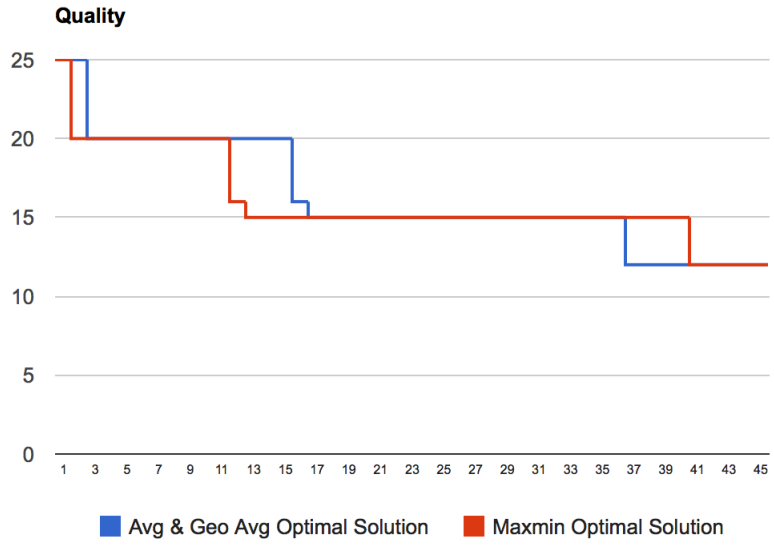

Figure 2.3: A comparison of optimal solutions.

model and methods show positive results in terms of the ability to quickly find the optimal solution. However, tests on more data sets are required to further confirm our observations.

Summary

In this chapter, we introduced the Constraint Optimization Problem, and modeled the Capstone Course Project Assignment Problem as a COP. We also discussed ordering heuristics and the local consistency methods that we use in search. Finally, we described our FC-BnB search algorithm, and evaluated its performance with two real class data sets.

Chapter 3

System Implementation and Features

In this chapter, we present the architecture and design of the software we have implemented. In addition, we describe from the user's point of view the features of our system. It is servers the purpose of a system reference for future developers and a user manual for capstone course instructors.

3.1 System Architecture

In this section, we start with an overview of the iLab infrastructure and its relation to our project. We then discuss the architecture of our system.

3.1.1 iLab Infrastructure

The iLab infrastructure has four interconnecting functional modules:

1. *The user registration module* allows the students and sponsors to be identified in the iLab system. It is mandatory for a student enrolled in the capstone course

to register on iLab. For security reasons, this module requires the student to be verified with CSE to confirm his/her identity and enrollment before it allows the student to proceed with registration[1]. After a successful verification, student information such as major (computer science or engineering) and availability[2] is collected. Students registered with iLab can log in with their CSE credentials but they can also choose to use an independent password. In addition, an organization (company or institution) can submit an application to become a sponsor for the course. An email with a log-in token is sent to the sponsor once the iLab administrator approves the application. The sponsor is then asked to enter a password for future authentication purposes.

2. *The information exchange module* provides a platform for the students and the sponsors to communicate. On the one hand, this module provides an interface for the sponsor to create projects. For each project created, the sponsor is required to specify the project name and its capacity, and he/she also needs to provide a description of the project by uploading a proposal document. On the other hand, students can view the projects proposed by the sponsors and submit applications if they are interested. A resume and a cover letter is required for each project application as well as a preference value between 0 and 5. After the deadline of student application, the sponsors can review the applicants and give their preferences (details on this process are described in Section 2.1.2).

3. *The project assignment module* is our contribution to the iLab infrastructure. It uses the information gathered in the previous two modules and assists the

[1]Through the CSE Authentication Mechanism.

[2]The availability information indicates whether the student is a 1-semester or 2-semester student. Most students are required to take the capstone course for two semesters, but certain students are allowed to choose the one-semester option.

course instructors to find a good assignment. The result is saved in the iLab database and is shared with other modules in the iLab system.

4. *The progress management module* represents an ongoing effort of the iLab system as a complete life-cycle management facility for the capstone course. As of this writing, the course uses separate software such as Blackboard[3] and CSE Web Handin[4] for posting announcements, handouts, and submitting progress reports. The diversity of repositories and sites can be the source of confusion for the students who have to remember the right place to do the right things. Therefore, this module aims to incorporate the above mentioned functionalities to provide a more consistent user experience.

Figure 3.1 shows the iLab infrastructure from a user's perspective. The user registration and information exchange modules are intended solely for the students and sponsors whereas the project assignment module serves primarily the instructors and iLab staff. The pending progress management module is designed for all users. For ease of management, the iLab administrator is granted superuser privilege, and is therefore able to overwrite other users' actions.

From a system point of view (Figure 3.2), the entire iLab infrastructure can be divided into three projects: the iLab Info Project, OPRAM, and the iLab X Project. The three projects focus on different functional responsibilities and they exchange data through a shared database (iLab DB).

3.1.2 OPRAM Architecture

In this section, we first introduce the background of a Google Web Toolkit (GWT) application. We then discuss the mechanism (RPC) of our client-server communication.

[3] www.blackboard.com
[4] cse.unl.edu/handin

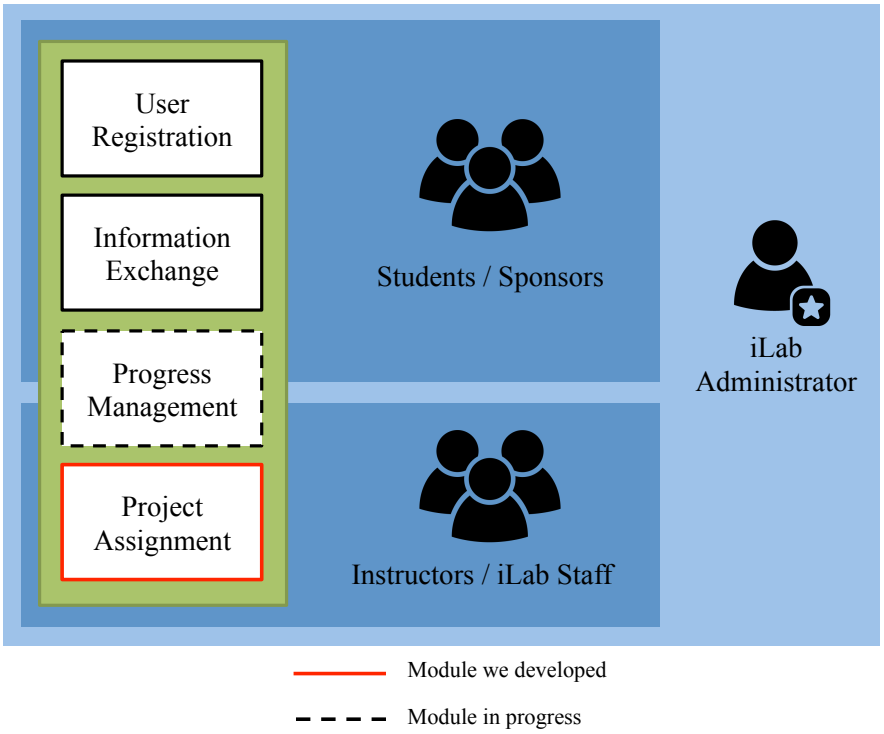

Figure 3.1: iLab infrastructure: a user's view.

Finally, we describe the architecture of our system.

3.1.2.1 Google Web Toolkit (GWT)

GWT is an opensource development toolkit for building and optimizing complex browser-based applications [GWT, 2014]. It allows for easy development of AJAX[5] based websites using the Java programming language. GWT takes a strong object-oriented approach to software architecture. When the Java code is compiled, specific AJAX code is generated for every well known browser, therefore offering great platform compatibility. In additional, GWT comes with many pre-built UI components, and also supports inclusion of third party software libraries.

Like any web application, GWT adopts a client-server model (Figure 3.3) with

[5]AJAX is short for Asynchronous JavaScript and XML.

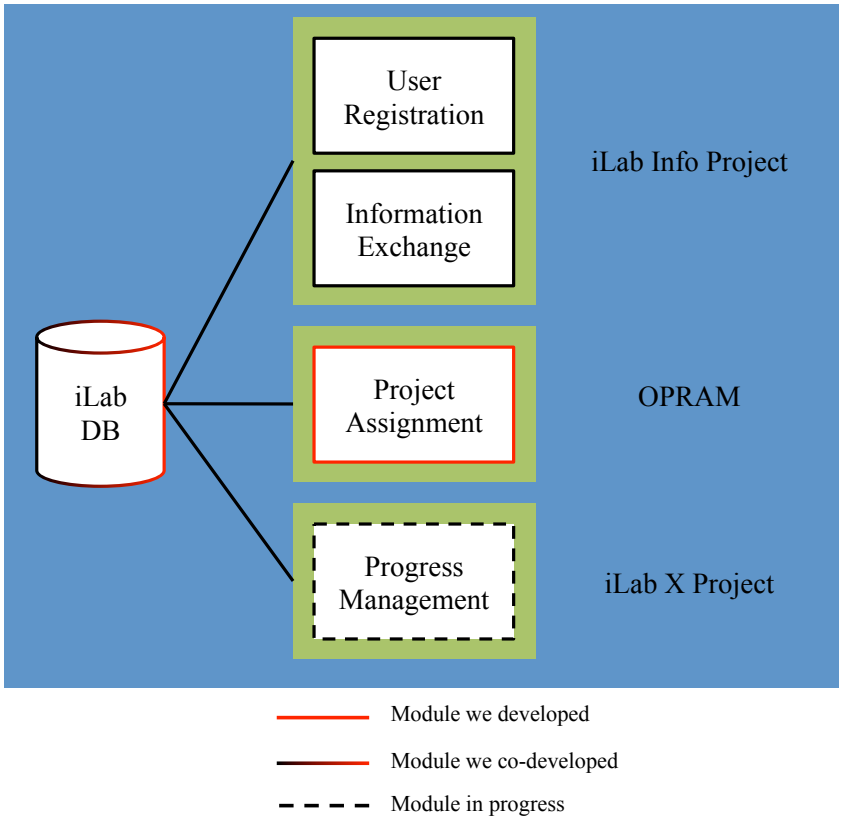

Figure 3.2: iLab infrastructure: a system view.

AJAX running on the client side in a browser and Java EE[6] running at the backend on the server. The communication between the client and the server is achieved through Remote Procedure Calls (RPC), which we discuss in the next section.

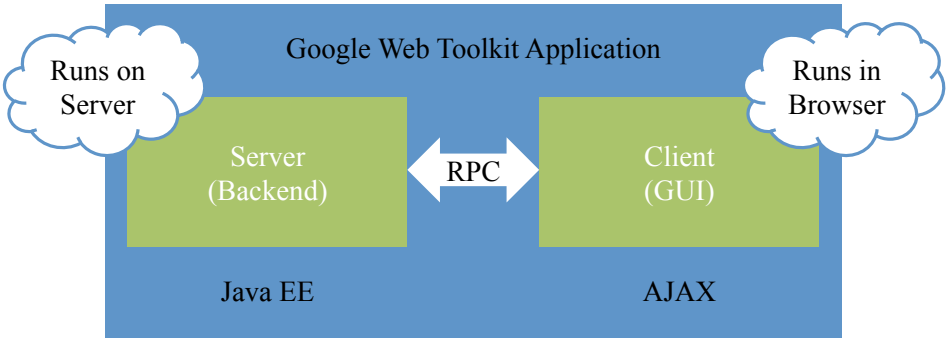

Figure 3.3: GWT application architecture.

[6]Java EE or Java Platform Enterprise Edition is a platform that provides APIs and runtime environment for developing network and web services.

3.1.2.2 Remote Procedure Call (RPC)

A Remote Procedure Call is a form of inter-process communication that enables one computer program to invoke a procedure (or function) in another computer program across the network. In GWT, RPC allows the client and server to pass data (in the form of serialized[7] Java objects) back and forth over the HTTP.[8] In Figure 3.4, the client process first initiates a RPC that invokes a server-side procedure which the server process subsequently executes. Notice that, in GWT, RPC is asynchronous. Therefore, it does not block (or informally freeze) the client process while it waits for a reply from the server. When the procedure returns, the server process calls back the client with the results for further processing. RPC is commonly used to access a database from the front-end. In our project, it is also the mechanism that our user interacts with the automatic solver.

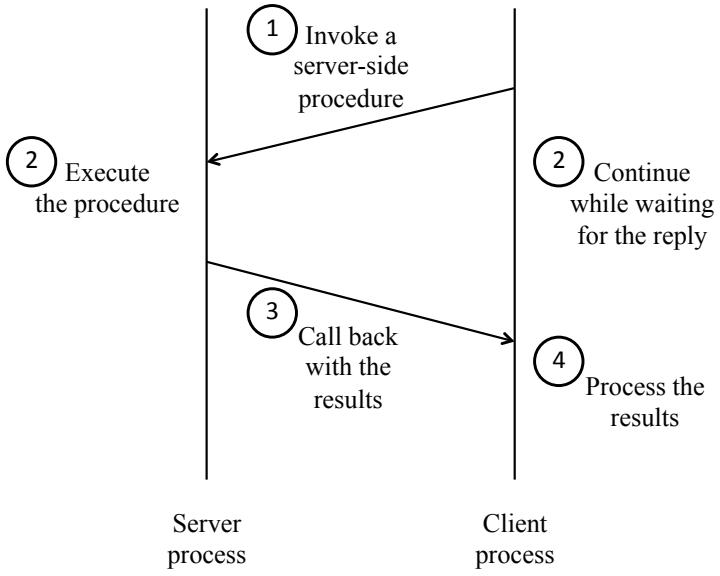

Figure 3.4: The mechanism of a Remote Procedure Call.

[7]Serialization is the process of translating data structures into a format that can be transmitted across a network connection link.

[8]HTTP is short for Hypertext Transfer Protocol, an application layer protocol widely used on the Internet.

3.1.2.3 The architecture of OPRAM

As a GWT application, our program inherits the client-server architecture. In addition, we adopted the design pattern of the Service Oriented Architecture (SOA) to enhance the moduality and extendability of our software. The key concept behind this design is the "separation of concerns" principle, which encapsulates the server-side logics into services offered to the client. In Figure 3.5, Service and Service Async. are two interfaces that provides a contract for message exchange between the client and the server. They are also the gateways for RPC requests (invocations) and replies (callbacks). Moreover, these two services have their corresponding implementations on both the client and the server. The implementation of the server-side service defines how the service requests (e.g., run the automatic solver or access the database) from the client should be handled, whereas the client implementation acts as a layer of abstraction for our graphical user interface (GUI) and the interactive solver.

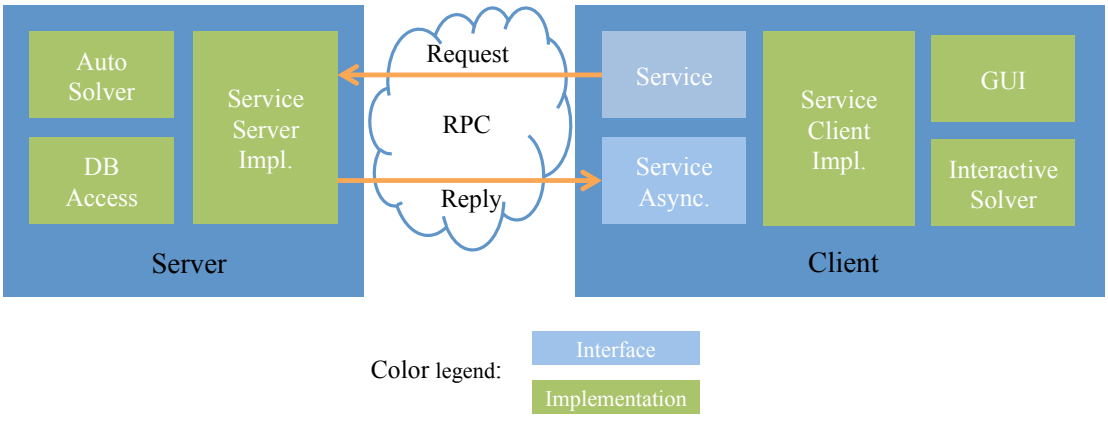

Figure 3.5: OPRAM architecture.

3.2 OPRAM's Functionalities

In this section, we describe the functionalities offered in OPRAM. We start with a bird's-eye view of the overall Graphical User Interface. Then, we describe each functionality in relation to the GUI components.

3.2.1 Web Graphical User Interface (GUI)

Our web GUI is designed with the user's convenience in mind. In addition to the legend, it consists of four communicating components: the control bar, student panel, project panel, and solution panel. Figure 3.6 shows a screen capture of the Web GUI.

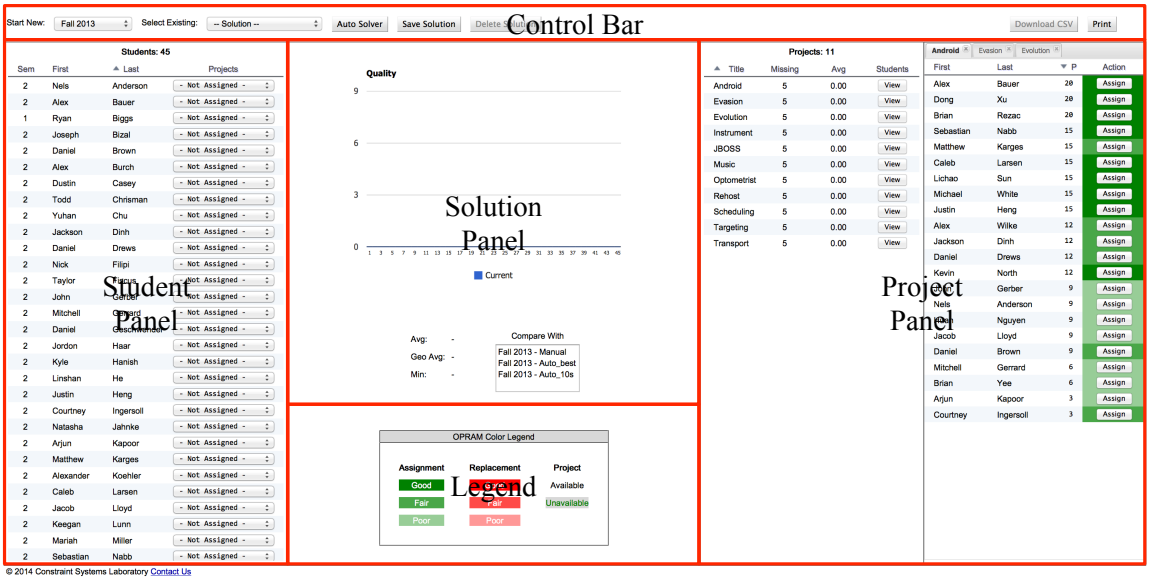

Figure 3.6: Web GUI layout.

3.2.2 The Legend's color map

The color map is intended to provide an intuitive visual guidance during the assignment process. We recommend the user to become familiar with the color conventions

we defined in the legend (Figure 3.7).

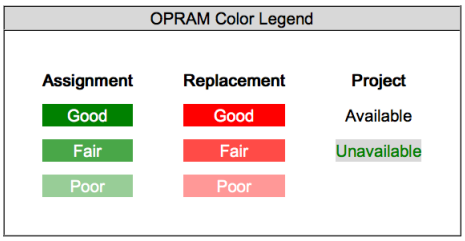

Figure 3.7: OPRAM color legend.

We use three[9] shades of green color to indicate the quality of an assignment and three[9] shades of red color to represent the quality of a replacement. The colors are mapped to the rank of the preference associated with each project among all projects applied by a student. Table 3.1 give the details of this mapping. Beside the color

Table 3.1: Definition of the quality of an assignment/replacement.

Quality	**Definition**
Good	The student is assigned to, or can be replaced[10] to, a project with a preference among the top $\frac{1}{3}$ (inclusive) of all his/her choices
Fair	The student is assigned to, or can be replaced[10] to, a project with a preference below the top $\frac{1}{3}$ but above the $\frac{2}{3}$ (inclusive) of all his/her choices
Poor	The student is assigned or can be replaced[10] to a project with a preference below the top $\frac{2}{3}$ of all his/her choices

indications for the quality of assignments and replacements (i.e., soft constraints), we also provide visual assistance for our user to keep track of the hard (i.e., capacity) constraints. Projects available for assignment are shown in normal text whereas the rest (unavailable) are displayed with a green text and a gray background (Figure 3.7).

[9]The reason we used only three shades (dark, normal, light) and not more is because they become very difficult to differentiate by human eyes as the number of shades increases.

[10] Replace means assign the student to a project different than his/her current assignment.

3.2.3 Administrative features

The control bar (Figure 3.8) is the command "center" of the program and the first GUI component our user sees when he/she visits the OPRAM website. The user can either choose to start from scratch (the "Start New Semester" option) or choose to view or revise an existing solution. In addition, the user can access our automatic solver by clicking on the "Auto Solver" button (details of this feature is discussed in Section 3.2.5). The control bar also provides several utility features such as save or delete a solution, download a solution as a CSV file, and print the current web page.

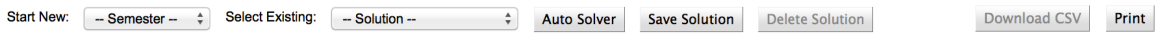

Figure 3.8: Control bar.

3.2.4 Interactive solver

The interactive solver has two working modes to offer the user extended flexibility during the assignment process. An assignment can either be made from the student perspective by selecting a project from the list of projects to which a student has applied (Figure3.9), or, from the project perspective, by choosing a student among all the students who applied for the project (Figure 3.10). Changes made in one perspective are automatically propagated and reflected in the other. In addition, a color representing the assignment quality (see Section 3.2.2) is visible in the background after each assignment being made. When a student is assigned, he/she is shown as a replacement option in all the other projects and a color reflecting the replacement quality (see Section 3.2.2) is displayed. Moreover, if the the capacity limit is reached for a project, the project is disabled for assignment and replacement, and only the "Undo" (assignment) action is active. Figure 3.9 and Figure 3.10 show

the descriptions of several other minor features of our interactive solver.

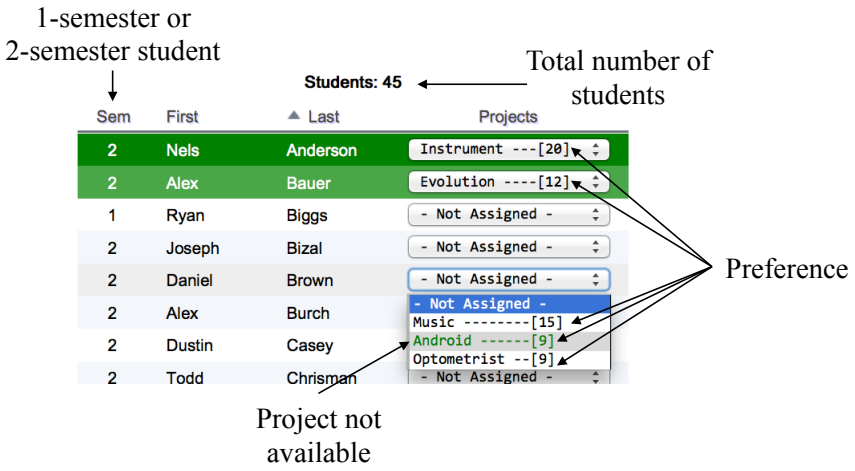

Figure 3.9: Student panel.

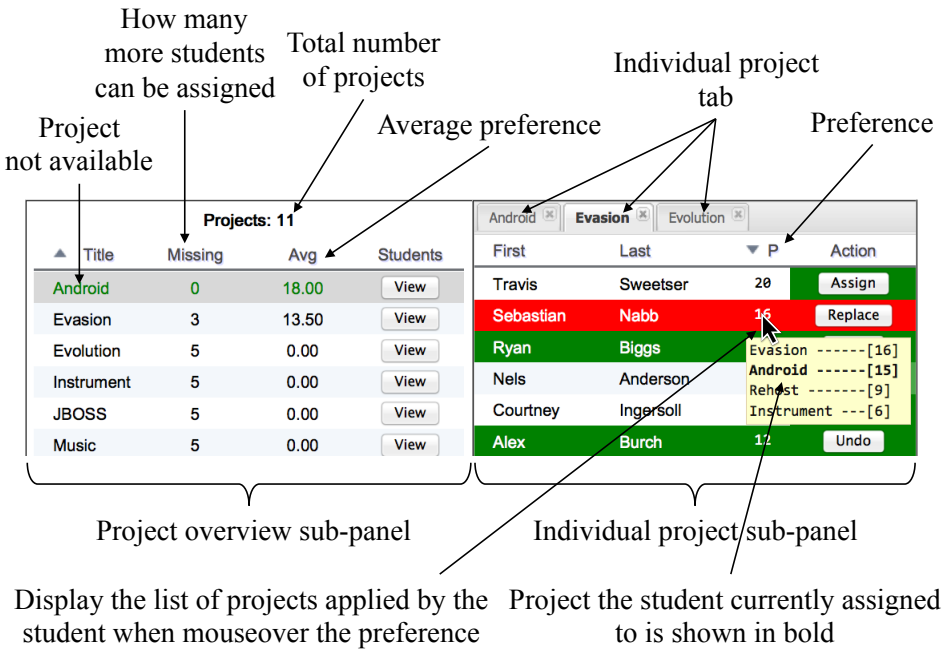

Figure 3.10: Project panel.

3.2.5 Automatic solver

The automatic solver is particularly useful when dealing with a large class (e.g., 20+ students). The technical details behind this solver is described in Chapter 2. Below, we discuss only the instructions on how to use this solver.

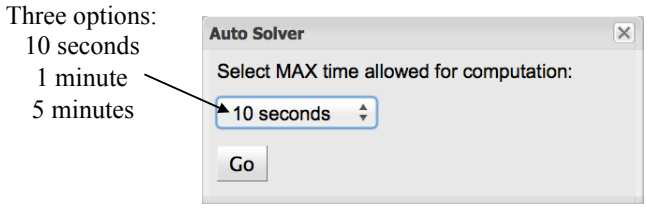

Figure 3.11: Automatic solver dialog window.

To access the automatic solver, the user needs to click the "Auto Solver" button on the control bar (see Section 3.2.3). A window (Figure 3.11) then pops up asking the user to select the maximum time duration he/she wants the solver to run. The longer the time allocated, the better the solution that can be found. The solver returns the optimal solution as soon as it is found or informs the user that no solution exists. In other cases, after the time runs out, the solver returns the best solution it finds withing the allocated time. The returned solution is displayed in the same GUI components of the interactive solver so that the user can make, at his/her discretion, further adjustments to the solution found.

3.2.6 Solution comparison

The solution comparison feature allows the user to compare the quality of different solutions. The centerpiece of this feature is a step chart that plots, in decreasing preference value, every individual assignment. The chart updates dynamically with each modification (i.e., assignment or replacement) made by the user.

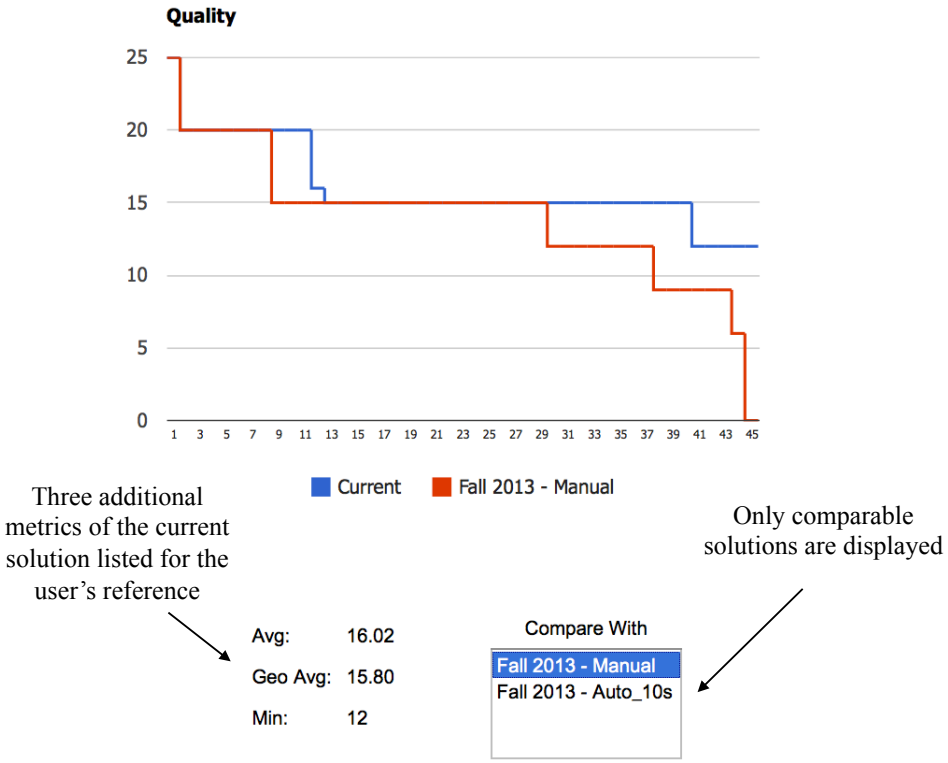

Figure 3.12: Solution panel.

Figure 3.12 shows two solutions for the class of Fall 2013. The solution labeled "current" is the optimal solution generated by our automatic solver. The solution labelled "manual" is the solution used for the real assignment.[11] Although both solutions have the same number of students with the highest preference score (i.e., 25), the solution found by our automatic solver (labeled "Current") has significantly better quality assignments at the lower end and in the upper middle range than the one made manually (labelled "Manual").

[11]The manual solution displayed here assigned only 44 out of 45 students because the last student was assigned to a different project which he did not previously apply. Nevertheless, the two solutions remain comparable to each other.

Summary

In this chapter, we presented an overview of the iLab infrastructure where our software will reside. We then introduced the system architecture of OPRAM in the context of GWT and discussed our client-server communication mechanisms (RPCs). Finally, we describe the key functionalaties of OPRAM with visual illustrations.

Chapter 4

Conclusions and Future Works

In this project, we have designed and implemented a web-based decision support system to assist the capstone course instructors with the task of assigning undergraduate seniors to work on sponsored projects in CSE. We have successfully modeled the Capstone Course Project Assignment Problem as a Constraint Optimization Problem and developed a constraint-based hybrid search algorithm (automatic solver) to look for the optimal solution. We have also built an interactive solver that keeps track of the consistency state and works side-by-side with the user to find solutions. The two solvers are integrated into a single system through a shared web graphical user interface. In addition to providing a unified user experience, this graphical user interface also offers visual hints on assignment/replacement at local (the color map feature) and global levels (the chart feature).

Experiments with real class data (Fall 2013 and Spring 2014) of our system have demonstrated a significant improvement on efficiency in assignment decision making process, reducing the time needed to minutes from, previously, hours when done manually on paper. In addition, the results showed a substantial increase in the quality of the solution found by our solver as depicted in the chart (in Figure 3.12)

that compares the automatically generated solution with the manual solution used in Fall 2013. Although the results from our experiments are by no means conclusive, the current instructors of the capstone course and the iLab manager are impressed by the capabilities of OPRAM and have decided to take advantage of this system in the upcoming semesters.

In preparation for the next stage of development, we summarize below our thoughts on the directions for further improvements:

1. We started with the assumption that the Capstone Course Project Assignment Problem is *NP-Hard* and therefore did not follow the path to find a polynomial search algorithm. Though assignment problems of this kind generally[1] do not belong to the class of *P*, a formal proof is required.

2. We adopted the idea of "maximizing the minimal happiness" (i.e., maxmin) as our objective function for optimization (Section 2.1.2). Alternative choices such as the concept of "stable matching" [Gale and Shapley, 1962] may also be used.

3. In our model, the project capacity limit is the only type of hard constraint that must be satisfied. We consider it useful to introduce a minimum size for each project as a potential hard constraint to avoid understaffing.

4. So far, our system allows the user to refine the solution generated by the automatic solver within the interactive solver. It would be ideal for the system to support the reverse work flow and enable the automatic solver to complete or improve the partial assignment made by the user with the interactive solver.

[1] However, there are exceptions [Gale and Shapley, 1962].

Appendix A

Search Results

This appendix contains the results of our search algorithm performed on the data from Spring 2014 and Fall 2013 classes. The Spring 2014 data set contains 10 student and 3 projects and the Fall 2013 data set contains 45 students and 11 projects. In each data set, we experimented three different optimization criteria: *Avg*, *Geo Avg* and *Maxmin*. An envaluation of the search results is discussed in Section 2.3. Below, we describe the table headers:

- Sol #: The solution number. (The quality, based on the given optimization criterion, of a solution increases with the solution number.)
- *Avg*: The average all the preferences.
- *Geo Avg*: The geometric average of all the preferences.
- *Maxmin*: The quality of the *Maxmin* optimization criterion represented by counting the number of individual assignments with the same preferences. For example,

$$9 : 2, 15 : 3$$

means 2 assignments with preference 9 and 3 assignments with preference 15.

- #NV: The number of nodes visited. This also equals the number of times the *label* function (Procedure 1) is called.
- SB: The shallowest backtracking level. When the entire search space is traversed, SB equals to 1.
- CPU: The CPU time spent in milliseconds.

A.1 Results of Spring 2014

Table A.1: Search results on data set Spring 2014 with *Avg* optimization criterion.

Sol #	*Avg*	*Geo Avg*	*Maxmin*	#NV	SB	CPU
1	5.90	5.14	2:1,3:3,6:2,9:4	10	10	2
2	6.20	5.50	2:1,3:2,6:3,9:4	13	8	7
End				21	1	10

Table A.2: Search results of Spring 2014 with *Geo Avg* optimization criterion.

Sol #	*Avg*	*Geo Avg*	*Maxmin*	#NV	SB	CPU
1	5.90	5.14	2:1,3:3,6:2,9:4	10	10	2
2	6.20	5.50	2:1,3:2,6:3,9:4	13	8	6
End				22	1	10

Table A.3: Search results of Spring 2014 with *Maxmin* optimization criterion.

Sol #	*Avg*	*Geo Avg*	*Maxmin*	#NV	SB	CPU
1	5.90	5.14	2:1,3:3,6:2,9:4	10	10	3
2	6.20	5.50	2:1,3:2,6:3,9:4	13	8	8
3	6.00	5.35	3:4,6:2,9:4	25	4	12
End				33	1	15

A.2 Results of Fall 2013

Table A.4: Search results of Fall 2013 with *Avg* optimization criterion.

Sol #	*Avg*	*Geo Avg*	*Maxmin*	#NV	SB	CPU
1	15.29	14.76	6:2,9:2,12:6,15:23,16:1,20:10,25:1	45	45	6
2	15.36	14.89	6:1,9:3,12:6,15:23,16:1,20:10,25:1	69	33	17
3	15.42	14.98	6:1,9:2,12:7,15:23,16:1,20:10,25:1	189	31	36
4	15.47	14.94	6:2,9:1,12:7,15:22,16:1,20:11,25:1	263	30	49
5	15.60	15.16	6:1,9:2,12:6,15:23,16:1,20:11,25:1	271	30	57
6	15.67	15.31	9:2,12:8,15:22,16:1,20:11,25:1	715	28	90
7	15.78	15.33	6:1,9:2,12:5,15:23,16:1,20:12,25:1	1417	24	128
8	15.82	15.35	6:1,9:2,12:6,15:21,16:1,20:13,25:1	7790	22	234
9	15.89	15.51	9:2,12:8,15:20,16:1,20:13,25:1	11255	22	278
10	15.91	15.57	9:2,12:6,15:23,16:1,20:12,25:1	13451	21	309
11	15.93	15.43	6:1,9:2,12:6,15:21,16:1,20:12,25:2	20428	19	388
12	16.00	15.50	6:1,9:2,12:5,15:22,16:1,20:12,25:2	20713	19	396
13	16.02	15.69	9:1,12:8,15:21,16:1,20:13,25:1	37160	19	599
14	16.04	15.53	6:1,9:2,12:6,15:20,16:1,20:13,25:2	67461	19	908
15	16.07	15.67	9:2,12:7,15:21,16:1,20:12,25:2	140651	19	1716
16	16.13	15.74	9:2,12:6,15:22,16:1,20:12,25:2	141507	19	1730
17	16.18	15.77	9:2,12:7,15:20,16:1,20:13,25:2	291298	18	3351
18	16.20	15.84	9:1,12:7,15:22,16:1,20:12,25:2	301567	18	3469
19	16.27	15.95	12:8,15:22,16:1,20:12,25:2	2511940	7	19191
20	16.31	15.97	12:9,15:20,16:1,20:13,25:2	2581994	7	19569
End				12524279	1	90376

Table A.5: Search results of Fall 2013 with *Geo Avg* optimization criterion.

Sol #	*Avg*	*Geo Avg*	*Maxmin*	#NV	SB	CPU
1	15.29	14.76	6:2,9:2,12:6,15:23,16:1,20:10,25:1	45	45	8
2	15.29	14.81	6:1,9:3,12:7,15:22,16:1,20:10,25:1	68	35	21
3	15.36	14.89	6:1,9:3,12:6,15:23,16:1,20:10,25:1	92	33	33
4	15.36	14.91	6:1,9:2,12:8,15:22,16:1,20:10,25:1	131	31	44
5	15.42	14.98	6:1,9:2,12:7,15:23,16:1,20:10,25:1	329	31	68
6	15.60	15.16	6:1,9:2,12:6,15:23,16:1,20:11,25:1	517	30	87
7	15.56	15.22	9:2,12:8,15:23,16:1,20:10,25:1	843	30	112
8	15.67	15.31	9:2,12:8,15:22,16:1,20:11,25:1	1420	28	144
9	15.78	15.33	6:1,9:2,12:5,15:23,16:1,20:12,25:1	2374	24	173
10	15.73	15.39	9:2,12:7,15:23,16:1,20:11,25:1	3170	24	200
11	15.78	15.41	9:2,12:8,15:21,16:1,20:12,25:1	5171	24	231
12	15.82	15.43	9:2,12:9,15:19,16:1,20:13,25:1	15592	22	359
13	15.89	15.51	9:2,12:8,15:20,16:1,20:13,25:1	16943	22	379
14	15.91	15.57	9:2,12:6,15:23,16:1,20:12,25:1	22133	21	441
15	15.91	15.59	9:1,12:8,15:22,16:1,20:12,25:1	25260	21	479
16	15.96	15.61	9:1,12:9,15:20,16:1,20:13,25:1	45992	19	749
17	16.02	15.69	9:1,12:8,15:21,16:1,20:13,25:1	46014	19	754
18	16.13	15.74	9:2,12:6,15:22,16:1,20:12,25:2	100164	19	1404
19	16.04	15.77	12:8,15:23,16:1,20:12,25:1	111045	19	1538
20	16.09	15.79	12:9,15:21,16:1,20:13,25:1	185659	18	2444
21	16.20	15.84	9:1,12:7,15:22,16:1,20:12,25:2	225702	18	2900
22	16.22	15.92	12:7,15:24,16:1,20:11,25:2	2679948	7	21838
23	16.27	15.95	12:8,15:22,16:1,20:12,25:2	2726942	7	22146
24	16.31	15.97	12:9,15:20,16:1,20:13,25:2	2863167	7	23067
End				15386646	1	125599

Table A.6: Search results of Fall 2013 with *Maxmin* optimization criterion.

Sol #	*Avg*	*Geo Avg*	*Maxmin*	#NV	SB	CPU
1	15.29	14.76	6:2,9:2,12: 6,15:23,16:1,20:10,25:1	45	45	15
2	15.22	14.72	6:1,9:4,12: 6,15:22,16:1,20:10,25:1	57	39	27
3	15.29	14.81	6:1,9:3,12: 7,15:22,16:1,20:10,25:1	80	35	40
4	15.36	14.89	6:1,9:3,12: 6,15:23,16:1,20:10,25:1	104	33	50
5	15.36	14.91	6:1,9:2,12: 8,15:22,16:1,20:10,25:1	143	31	62
6	15.18	14.78	9:4,12: 8,15:22,16:1,20: 9,25:1	171	31	72
7	15.24	14.88	9:3,12: 9,15:22,16:1,20: 9,25:1	204	31	83
8	15.31	14.95	9:3,12: 8,15:23,16:1,20: 9,25:1	422	31	114
9	15.42	15.04	9:3,12: 8,15:22,16:1,20:10,25:1	625	30	135
10	15.49	15.12	9:3,12: 7,15:23,16:1,20:10,25:1	708	30	147
11	15.49	15.14	9:2,12: 9,15:22,16:1,20:10,25:1	991	30	172
12	15.56	15.22	9:2,12: 8,15:23,16:1,20:10,25:1	1152	30	186
13	15.67	15.31	9:2,12: 8,15:22,16:1,20:11,25:1	2136	28	231
14	15.56	15.24	9:1,12:10,15:22,16:1,20:10,25:1	2351	28	241
15	15.62	15.31	9:1,12: 9,15:23,16:1,20:10,25:1	2385	28	246
16	15.73	15.41	9:1,12: 9,15:22,16:1,20:11,25:1	5120	22	299
17	15.62	15.34	12:11,15:22,16:1,20:10,25:1	5978	22	320
18	15.64	15.39	12: 9,15:25,16:1,20: 9,25:1	7398	21	346
19	15.76	15.49	12: 9,15:24,16:1,20:10,25:1	8112	21	360
20	15.87	15.59	12: 9,15:23,16:1,20:11,25:1	11856	19	419
21	15.71	15.47	12: 8,15:26,16:1,20: 9,25:1	15595	19	476
22	15.82	15.57	12: 8,15:25,16:1,20:10,25:1	16296	19	490
23	15.93	15.67	12: 8,15:24,16:1,20:11,25:1	16411	19	497
24	16.04	15.77	12: 8,15:23,16:1,20:12,25:1	19119	19	536
25	15.89	15.64	12: 7,15:26,16:1,20:10,25:1	52563	17	988
26	16.00	15.74	12: 7,15:25,16:1,20:11,25:1	70206	17	1237
27	15.62	15.42	12: 6,15:30,16:1,20: 7,25:1	177025	15	2500
28	15.73	15.52	12: 6,15:29,16:1,20: 8,25:1	177307	15	2507
29	15.84	15.62	12: 6,15:28,16:1,20: 9,25:1	180823	15	2553
30	15.96	15.72	12: 6,15:27,16:1,20:10,25:1	181295	15	2563
31	15.80	15.62	12: 5,15:29,16:1,20:10	497685	9	5717
32	16.02	15.80	12: 5,15:28,16:1,20:10,25:1	926010	7	9169
End				2174731	1	19611

Appendix B

Integration in iLab Infrastructure

In the appendix, we discuss the integration requirements that needs to be met to accommodate and the interfaces we have designed for integration support.

B.1 Database integration

Because OPRAM shares a common database with other applications within the iLab infrastructure, the ilab database must incorporate the schema defined by OPRAM (Figure B.1). While it is not necessary to create the tables mentioned in the schema as stand-along entities (i.e., fields used by other applications may also be added to these tables), the relations among them must be exact. For smooth integration, we have separated the implementation details of the actual database into a configurable XML file (Listing B.2) which can be loaded dynamically during program execution. This separation offers the iLab database developers the freedom to name the tables and fields however they see fit without having compatibility issues with OPRAM and after they finalize the database design, they simply map the final table names and field names in the configuration file.

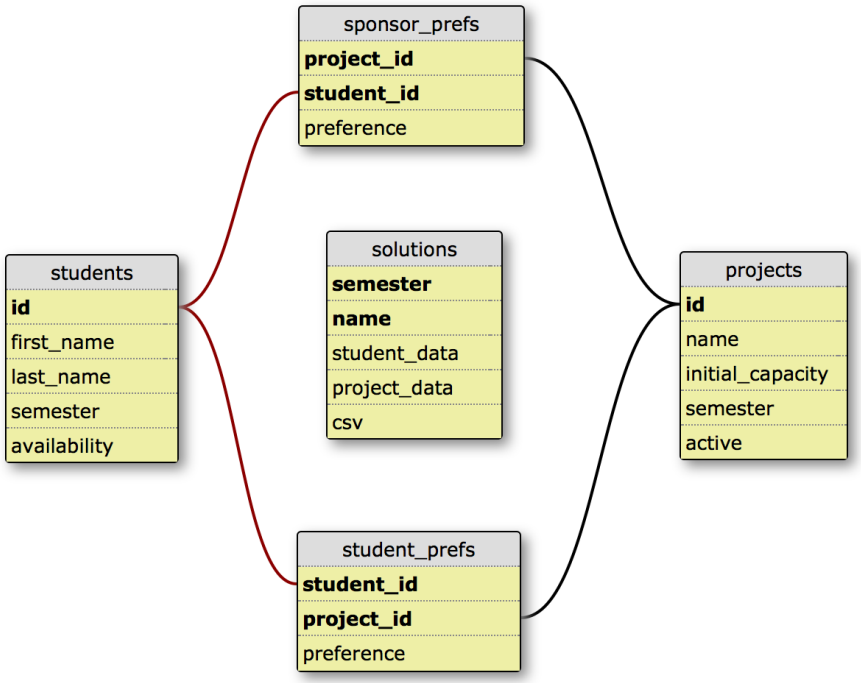

Figure B.1: OPRAM database schema (discussed in Appendix C).

Listing B.1: Sample database schema configuration file `db_schema.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<database>
   <students tb_name="students">
      <id>id</id>
      <f_name>first_name</f_name>
      <l_name>last_name</l_name>
      <avail>availability</avail>
      <sem>semester</sem>
   </students>
   <projects tb_name="projects">
      <id>id</id>
      <name>name</name>
      <cap>initial_capacity</cap>
```

```
14        <sem>semester</sem>
15        <act>active</act>
16    </projects>
17    <student_prefs tb_name ="student_preferences">
18        <student_id>student_id</student_id>
19        <project_id>project_id</project_id>
20        <pref>rating</pref>
21    </student_prefs>
22    <sponsor_prefs tb_name="sponsor_preferences">
23        <student_id>student_id</student_id>
24        <project_id>project_id</project_id>
25        <pref>rating</pref>
26    </sponsor_prefs>
27    <solutions tb_name="solutions">
28        <sem>semester</sem>
29        <name>name</name>
30        <student_data>student_data</student_data>
31        <project_data>project_data</project_data>
32        <csv>csv</csv>
33    </solutions>
34    <auth_tokens tb_name="auth_tokens">
35        <token>token</token>
36    </auth_tokens>
37 </database>
```

B.2 Access control integration

OPRAM is designed with the sole purpose to solve the Capstone Course Project Assignment Problem and it must rely on other iLab component to authenticate its

users. We have developed an token-based authentication mechanism as a way for access control in OPRAM. Figure B.2 illustrates the work flow of this authentication.

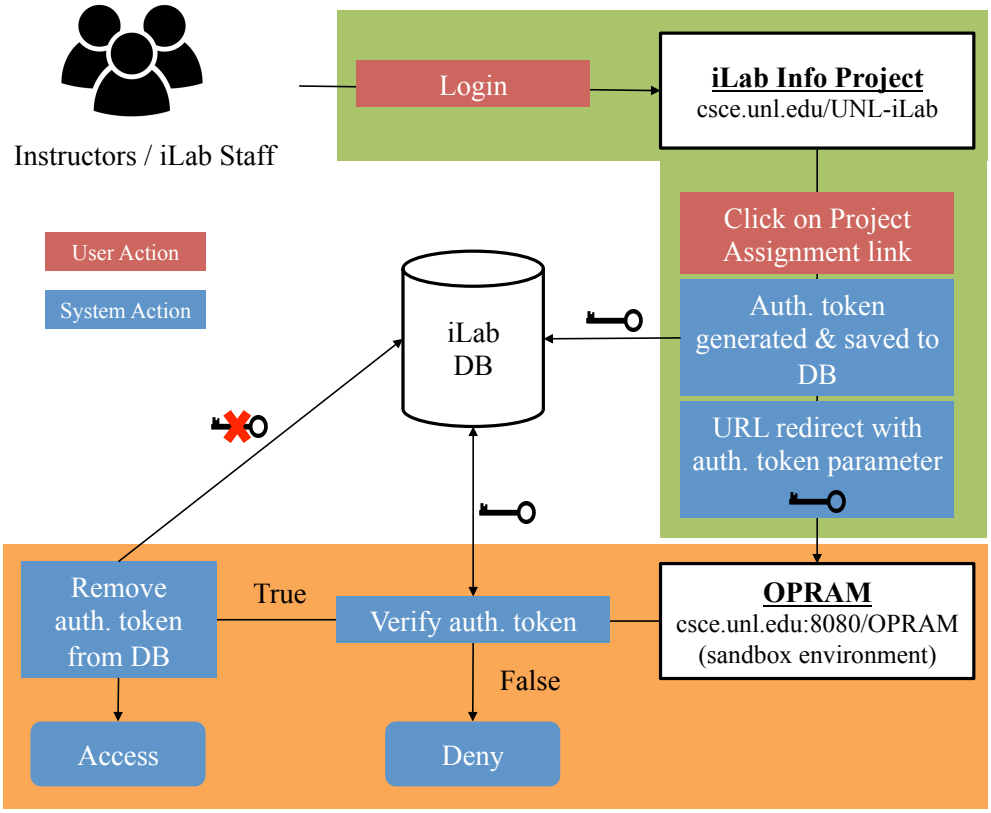

Figure B.2: OPRAM user authentication work flow.

B.3 Deployment integration

OPRAM is a Java EE application and can be deployed as a WAR file on any Java web server. We have successfully tested the deployment of OPRAM on the CSE Glassfish server (in the sandbox environment[1]) and we have discussed with the CSE system

[1]http://cscs.unl.edu:8080/OPRAM

administrator to deploy our application in the production environment[2] in the future. Below is sample a system properties file with parameters for deployment .

Listing B.2: Sample system properties file `system.properties`

```
1 root_path=# path to the location where OPRAM is deployed
2 db_schema_path=# path to the db_schema.xml file
3 db_path=mysql://cse.unl.edu:3306/ilab
4 db_user=ilab
5 db_pass=******
```

[2]http://cse-apps.unl.edu/OPRAM

Appendix C

Documentation of Database and MySQL Queries

In this appendix, we provide low-level descriptions of the database and MySQL queries. It is beyond the scope of this appendix to describe the entire iLab database schema. We list below only the tables and queries that are relevant to our project.

C.1 Description of database tables

In this section, we describe the database tables. For simplicity, certain tables are abridged to focus on the fields only related to OPRAM. For each table, the highlighted field(s) (shown in bold italic) is the primary key of the table and it uniquely identifies each record in the table.

C.1.1 `students`

id: This is the unique identifier for the records in the `students` table.

first_name: This is the student's first name.

Table C.1: Table `students` (partial).

Field	Type	Null	Default
id	int	No	
first_name	varchar(32)	No	
last_name	varcahr(32)	No	
semester	varchar(16)	No	
availability	tinyint	No	2

last_name: This is the student's last name.

semester: This is the semester when the student starts the capstone course. The format of `semester` is: SEASON YEAR (e.g., Fall 2013, Spring 2014).

availability: This is the duration (in semesters) the student is enrolled in the capstone course.

C.1.2 `projects`

Table C.2: Table `projects` (partial).

Field	Type	Null	Default
id	int	No	
name	varchar(32)	No	
semester	varchar(16)	No	
initial_capacity	smallint	No	
active	tinyint	No	1

id: This is the unique identifier for the records in the `project` table.

name: This is the name of the project.

semester: This is the semester in which the project is offered. The format of `semester` is: SEASON YEAR (e.g., Fall 2013, Spring 2014).

initial_capacity: This is the capacity limit of the project.

active: This is the project activity status. The values should be either 1 or 0.

- 1: The project is active and available for assignment.
- 2: The project is inactive and not available for assignment.

C.1.3 `student_prefs`

Table C.3: Table `student_prefs` (partial).

Field	Type	Null	Default
student_id	int	No	
project_id	int	No	
preference	int	No	3

student_id: This is the identifier for the student. It is associated with the field "id" in `students` (Table C.1) as the foreign key.

project_id: This is the identifier for the proejct. It is associated with the field "id" in `projects` (Table C.2) as the foreign key.

preference: This is the student's preference for the project. Its value ranges from 0 (absolutely disagree) to 5 (strongly agree). The details about the preference scale is discussed in Chapter 2.

C.1.4 `sponsor_prefs`

Table C.4: Table `sponsor_prefs` (partial).

Field	Type	Null	Default
project_id	int	No	
student_id	int	No	
preference	int	No	3

project_id: This is the identifier for the proejct. It is associated with the field "id" in `projects` (Table C.2) as the foreign key.

student_id: This is the identifier for the student. It is associated with the field "id" in `students` (Table C.1) as the foreign key.

preference: This is the project sponsor's preference for the student. Its value ranges from 0 (absolutely disagree) to 5 (strongly agree). The details about the preference scale is discussed in Chapter 2.

C.1.5 `solutions`

Table C.5: Table `solutions`.

Field	Type	Null	Default
semester	int	No	
name	int	No	
student_data	longblob	No	
project_data	longblob	No	
csv	longtext	No	

semester: This is the semester of the solution. The format of `semester` is: SEASON YEAR (e.g., Fall 2013 and Spring 2014).

name: This is the name of the solution.

student_data: This is the Java object used to store and transfer student data.

project_data: This is the Java object used to store and transfer project data.

csv: This is the plain-text representation of a solution in Comma-separated values (CSV) format with one assignment (FIRST NAME, LAST NAME, PROJECT NAME) per row.

C.1.6 auth_tokens

Table C.6: Table auth_tokens.

Field	Type	Null	Default
token	text	No	

token: This is the authentication token that determines whether or not a user has the privilege to access OPRAM.

C.2 MySQL queries

In this section, we document the MySQL queries used by our program to access the database. The queries are divided into two categories by the software component within which they are executed. A variable in a query is denoted with a "$" sign in front of the variable's name (e.g., $variable).

C.2.1 Queries by solvers

Below are the SQL queries used by the solvers:

- This query fetches the assignment information of a given semester from the student perspective. It is used in both solvers. Node consistency (refer to Chapter 2) checks are performed in this query to remove zero preferences.

```
SELECT student.id, student.first_name, student.last_name,
student.availability, project.name, student_prefs.preference,
sponsor_prefs.preference FROM student
LEFT JOIN student_prefs ON student_pref.student_id = student.id
AND student_pref.preference != 0
```

```
LEFT JOIN project ON project.id = student_prefs.project_id
LEFT JOIN sponsor_prefs ON sponsor_prefs.project_id = project.id
AND sponsor_prefs.student_id = student.id
AND sponsor_pref.preference != 0
WHERE student.semester = "$semester" AND project.active = 1;
```

- This query fetches the project information. It is used in the automatic solver.

```
SELECT project.name, project.initial_capacity FROM project
WHERE project.active = 1 AND project.semester = "$semester";
```

- This query fetches the assignment information of a given semester from the project perspective. It is used in the interactive solver. Node consistency (refer to Chapter 2) checks are performed in this query to remove zero preferences.

```
SELECT project.id, project.name, project.initial_capacity, student.id
FROM project
LEFT JOIN student_prefs ON student_prefs.project_id = project.id
AND student_prefs.preference != 0
LEFT JOIN student ON student.id = student_prefs.student_id
LEFT JOIN sponsor_prefs ON sponsor_prefs.student_id = student.id
AND sponsor_prefs.project_id = project.id
AND sponsor_prefs.preference != 0
WHERE project.semester = "$semester" AND project.active = 1;
```

C.2.2 Queries by web GUI

Below are the SQL queries used by the web GUI:

- This query verifies whether a user has the proper right to access the OPRAM website.

```
SELECT * FROM auth_tokens WHERE token = $token;
```

- This query removes the token after a user has been authenticated.

```
DELETE FROM auth_tokens WHERE token = $token;
```

- This query fetches the solution specified by (the semester and name) from the database. It corresponds to the "select existing solution" feature provided by the web GUI.

```
SELECT student_data, project_data, FROM solutions WHERE
semester = "$semester" AND name = "$name";
```

- This query removes the solution specified by (the semester and name) from the database. It corresponds to the "delete solution" feature provided by the web GUI.

```
DELETE FROM solutions WHERE semester = "$semester" AND name = "$name";
```

- The following two queries fetches the list of semesters and populates the "start new semester" drop down list in the web GUI.

```
SELECT COUNT(DISTINCT semester) FROM students;
```

```
SELECT DISTINCT semester FROM students;
```

- The following two queries fetches the list of solutions and populates the "select existing solution" drop down list in the web GUI.

```
SELECT COUNT(*) FROM solutions;
```

```
SELECT semester, name FROM solutions;
```

- The following two queries fetches the list of solutions of the current[1] semester and populations the "compare with" selection box in the solution panel.

```
SELECT COUNT(*) FROM solutions WHERE semester = "$semester";

SELECT semester, name FROM solutions WHERE semester = "$semester";
```

[1]The semester of which the assignment data is currently displayed in the web GUI.

Appendix D

Documentation of Software API

In this appendix, we list the software dependencies necessary for our program to function. We then provide a descriptions of the packages and files.

D.1 Software Dependencies

- Java EE Development Kit (JDK) and Java Runtime Environment (JRE) version 1.6 (1.7 compatible).
- Google Web Toolkit (GWT) version 2.4.0.
- MySQL Java Connector driver version 5.1.22.
- Sencha GXT (gxt-3.0.1.jar).
- GWT Chart (gwt-chart-0.9.9.jar).

D.2 Description of Packages and Files

D.2.1 Package `opram.server`

This package contains the Java classes for server side operations.

D.2.1.1 `OPRAMServiceServerImpl.java`

This class represents the services implemented on the server side.

D.2.2 Package `opram.server.db`

This package provides dynamic name mappings of fields in iLab database tables. The mapping is specified in the `db_schema.xml` file.

D.2.2.1 `AuthTokens.java`

This class represents a partial mapping of the database table that contains the authentication token information.

D.2.2.2 `Projects.java`

This class represents a partial mapping of the database table that contains the project information.

D.2.2.3 `Solutions.java`

This class represents a partial mapping of the database table that contains the solution information.

D.2.2.4 `SponsorPrefs.java`

This class represents a partial mapping of the database table that contains the sponsor preference information.

D.2.2.5 `StudentPrefs.java`

This class represents a partial mapping of the database table that contains the student preference information.

D.2.2.6 `Students.java`

This class represents a partial mapping of the database table that contains the student information.

D.2.3 Package `opram.server.resources`

This package contains the resources used on the server side.

D.2.3.1 `db_schema.xml`

This file contains the mapping of the names and fields of iLab database tables.

D.2.4 Package `opram.server.data`

This package contains the data structures used on the server side.

D.2.4.1 `StudentData.java`

This class contains the data structures representing the student (variable) and is used by the solver algorithms.

D.2.4.2 `ProjectData.java`

This class contains the data structures representing the project (value) and is used by the solver algorithms.

D.2.5 Package `opram.server.algo`

This package contains algorithms and methods used by the automatic solver.

D.2.5.1 `FC_BnB_Solver.java`

This class implements the FC_BnB algorithm (see Chapter 2) we developed.

D.2.5.2 `FC_Solver.java`

This class implements the FC algorithm [Haralick and Elliott, 1980].

D.2.5.3 `ValueOrdering.java`

This class implements the *most promising first* value ordering heuristic.

D.2.5.4 `VariableOrdering.java`

This class implements the *least domain* variable ordering heuristic.

D.2.6 Package `opram.server.util`

This packages contains the utilities necessary for server side functionalities.

D.2.6.1 `DataSerializer.java`

This class converts/serializes the server side data structures into corresponding shared data structures that can be transferred over the network to the client side.

D.2.6.2 `BD.java`

This class provides various query methods to the database.

D.2.6.3 `FileHandler.java`

This class generates CSV files for corresponding solutions that can be downloaded on the user's computer.

D.2.7 Package `opram.client`

This package contains the Java classes for client side operations.

D.2.7.1 `OPRAM.java`

This class the entry point of the OPRAM web application.

D.2.8 Package `opram.client.service`

This package contains the definition of services provided by the server to the client.

D.2.8.1 `OPRAMService.java`

This interface contains the definitions of service invocations (RPC invocations).

D.2.8.2 `OPRAMServiceAsync.java`

This interface contains the definitions of service callbacks (RPC callbacks).

D.2.8.3 `OPRAMServiceClientImpl.java`

This class represents the services implemented on the client side.

D.2.8.4 `OPRAMServiceClientInt.java`

This interface defines the services to be implemented on the client side.

D.2.9 Package `opram.client.gui`

This package contains the classes that control and render the graphical user interface on the web.

D.2.9.1 `AutoSolutionDialog.java`

This class represents the pop-up dialog that guides the user to interact with the automatic solver.

D.2.9.2 `ControlPanel.java`

This class represents the control bar on the top of the web GUI.

D.2.9.3 `DeleteSolutionDialog.java`

This class represents the pop-up dialog that confirms with the user before he/she deletes a solution.

D.2.9.4 `DynamicButtonCell.java`

This class represents the button widget on the individual project panel (Figure 3.10).

D.2.9.5 `DynamicSelectionCell.java`

This class represents the drop down list on the student panel (Figure 3.9).

D.2.9.6 `Legend.java`

This class represents the OPRAM color legend (Figure 3.7).

D.2.9.7 `MainGUI.java`

This class represents the main GUI container where other GUI components reside.

D.2.9.8 `ProjectTable.java`

This class represents the project overview panel (Figure 3.10 left).

D.2.9.9 `ProjectTable2.java`

This class represents the individual project panel (Figure 3.10 right).

D.2.9.10 `SaveSolutionDialog.java`

This class represents the pop-up dialog that guides the user to save a solution.

D.2.9.11 `SolutionChartPanel.java`

This class represent the chart on the solution panel (Figure 3.12).

D.2.9.12 `SolutionQualityPanel.java`

This class represents the three additional quality metrics on the solution panel (Figure 3.12).

D.2.9.13 `StudentTable.java`

This class represents the student panel (Figure 3.9).

D.2.9.14 `TableResources.java`

This class specifies the CSS styling resources to render the table widget.

D.2.10 Package `opram.client.resources`

This package contains the resources used on the client side.

D.2.10.1 `table.css`

This file defines the style and appearance of the table widget.

D.2.11 Package `opram.shared`

This package contains the common data structures and information shared by the client and the server.

D.2.11.1 `DataEnvelope.java`

This class represents the data envelope that is used to exchange data between the client and the server.

D.2.11.2 `DataInterface.java`

This class defines the actions the client wants the server to perform.

D.2.11.3 `Pair.java`

This class represents a generic implementation of a pair (i.e., $(item1, item2)$) and is used in many other data structures we defined.

D.2.11.4 `ProjectDataS.java`

This class represents the serializable version of the project (value) data structure.

D.2.11.5 `StudentDataS.java`

This class represents the serializable version of the student (variable) data structure.

Bibliography

Fahiem Bacchus and P. van Run. Dynamic Variable Ordering in CSPs. In *Principles and Practice of Constraint Programming, CP'95. Lecture Notes in Artificial Intelligence 976*, pages 258–275. Springer Verlag, 1995.

Rina Dechter. *Constraint Processing*, chapter Constraint Optimization, Chapter 13. Morgan Kaufmann, 2003.

Rina Dechter. *Constraint Processing*, chapter Consistency-Enforcing and Constraint Propagation, Chapter 3. Morgan Kaufmann, 2003.

D. Gale and L.S. Shapley. College Admissions and the Stability of Marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.

Pieter Andreas Geelen. Dual Viewpoint Heuristics for Binary Constraint Satisfaction Problems. In *Proceedings of the Tenth European Conference on Artificial Intelligence*, pages 31–35, Vienna, Austria, 1992.

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

Venkata Praveen Guddeti. Empirical Evaluation of Heuristic and Randomized Backtrack Search. Master's thesis, Department of Computer Science & Engineering, University of Nebraska-Lincoln, December 2004.

Google Web Toolkit Project. http://www.gwtproject.org, April 2014.

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

Grzegorz Kondrak and Peter van Beek. A Theoretical Evaluation of Selected Backtracking Algorithms. In *Proceedings of the 14 th International Joint Conference on Artificial Intelligence*, pages 541–547, Montréal, Québec, Canada, 1995.

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.

Ryan Way Hoong Lim. GTAAP: An Online System For Managing and Assigning Graduate Teaching Assistants to Academic Tasks. Master's Project. Department of Computer Science & Engineering, University of Nebraska-Lincoln, 2006.

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.

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

Venkateshwar Rao Thota. Online Interactive Problem-Solving. Master's Project. Department of Computer Science & Engineering, University of Nebraska-Lincoln, 2004.

Hui Zou. Iterative Improvement Techniques for Solving Tight Constraint Satisfaction Problems. Master's thesis, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE, December 2003.