Al Akhawayn University in Ifrane, Morocco

CSC 5309 Artificial Intelligence

Part Time Job CSP

By: **Ali El Amri**

Bassma Ammine

Mohammed Tahri-Sqalli

Supervised by: Dr. Vioetta Cavalli-Sforza

December,18 2017

Contents

Introduction:

This report summarizes our attempt to solve the problem of assigning TAs/graders to professors. The problem falls under constraints satisfaction problem, as this problem has different constraints from the point of view of both students and faculty.

The problem consists of having a number of students that need to work part time, either as part of a payed part time job or a contractual part time job, where they need to work for a minimum of hours and shall not exceed a maximum number of hours. The professors on the other hand can be granted up to a certain number of graders depending on the course load they are teaching. Professors and students usually can specify their preferences in terms of who to choose as a faculty or as a student. These are the major highlights of the problem, and the report will present additional constraints that need to be respected for an adequate solution.

The solution to be presented by this report uses a constraints satisfaction problem solver in order to find a suitable and adequate solution that tries to satisfy as much as possible the constraints described. The output of the software consists of assignments of students to faculty where the preferences of both entities are met as much as possible.

Survey of Tools:

Before diving in into the solution attempts, we have tried to compile a list of tools that solve constraint satisfaction problems. The tools that we have identified and analyzed use different languages, different technologies and different internal data structures. We have tried to be aware of these different tools and what makes each one different from the other in order to focus on attention into only one tool that we will be using in order to tackle the problem. The list of tools that we have identified is the following:

- OptaPlanner
- Python-constraint
- Choco-solver

The three tools above are the tools that we have analyzed as each one of us has tried to analyze and understand one tool and come up with a digest of the tool in order to come up with the final decision about what tool is best for us to use in order to approach the problem.

- OptaPlanner: https://www.optaplanner.org/
  This tool has been analyzed by Bassma Ammine. Based on the documentation, OptaPlanner is described to be a constraint solver that optimizes business resource planning use cases, such as Vehicle Routing, Employee Rostering, Cloud Optimization, Task Assignment, Job Scheduling, Bin Packing and other tasks. This organization faces such scheduling puzzles: assign a limited set of *constrained* resources (employees, assets, time and money) to provide products or services. OptaPlanner delivers more efficient plans to improve service quality and reduce costs. What

characterizes this tool is that it has been mostly used for industry purposes. It means that most users try to solve a real situation problem rather than a puzzle or a game.

- Python Constraint: https://labix.org/python-constraint Mohammed tried to look at the Python constraint library as he tried to use the same tool in the team formation constraint problem. The reason for not choosing this tool was mainly due to the fact that the data set is large
  The Python *constraint* module offers solvers for Constraint Solving Problems (CSPs) over finite domains in simple and pure Python. CSP is class of problems which may be represented in terms of variables (a, b, ...), domains (a in [1, 2, 3], ...), and constraints (a < b, ...). Python constraints is more focused on optimizing solutions.
- Choco solver: http://www.choco-solver.org/ Ali took care of looking at Choco Solver.
  Choco is a Free Open-Source Java library dedicated to Constraint Programming. The user models its problem in a declarative way by stating the set of constraints that need to be satisfied in every solution. Then, the problem is solved by alternating constraint filtering algorithms with a search mechanism.

The following reasons have encouraged us to choose Choco solver:

- Thousands of downloads per month
- Tens of thousands of website visitors per month
- 350+ pages of documentation
- 15+ realistic examples
- A very high automated test coverage with unit tests, integration tests, stress tests and performance regression tests
- A very permissive license (Apache License)
  and it's used it across the globe for small to medium to large to huge use cases.
- Oriented more towards industry problems
- Has different methodologies to specify the constraints.

We have tried to try each work with all the tools but only with few small examples, among these the 8 queens example. We tried to look at how the variables and constraints are represented and what are the rules needed to represent them. We have discovered that some tools use regular expressions like python solver, while other tools have long description of variables like OptaPlanner, and OptaPlanner is more oriented towards large industry constraint problems so it was a bit complex to understand. We found Choco Solver to be the most adequate to our task for the reasons above and for the ease of integration with excel sheets.

Concerning the memory usage and the CPU usage, we could not find any accurate information, but based on forums like stack overflow, the tools do not use extensively the resources and can work on the average computer, all depends on the size of the problem and how the constraints are formulated.

Data Formats and Files:

the following technologies were used in order to make this application possible:

Component	Technologies
Constraint Satisfaction Solver	• Java • Choco Solver
Database	• Excel Sheets provided in the project description (.xlsx) • MySQL
Web App Presentation	• Laravel, • VueJs

It is important to mention that while the form of the data provided in the excel sheets attached to the project description was not changed except the uniformity of the course codes (added a spade between the course letters and the numbers across all the courses as some did not have that space)

Important Links:

The following table summarizes the important link that is needed to have the application function properly: It is important to note that this report is included in the GitHub repository of the project.

Information	Link
Web App Link	http://part-time-job-csp.herokuapp.com
GitHub Repository containing the Java App	https://github.com/Shoodey/PartTimeJobCSP
Demonstration Video	https://www.youtube.com/watch?v=TlJR2McRbjI

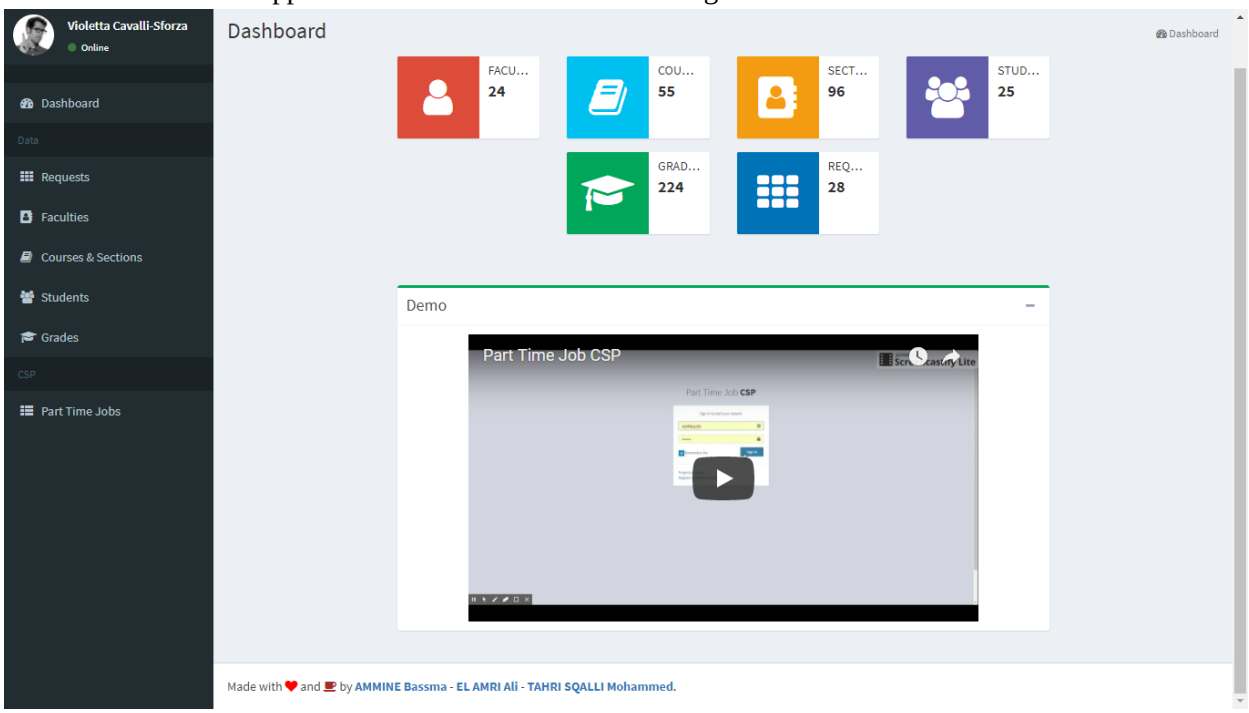
Execution Instruction:

Having both of us benefited from the Human Computer Interaction class, we have decided that our Constraint Satisfaction Problem will have an appealing interface. And, based on the continuity mentality, we have started from last semester project and have decided that the interface would be the one from last semester project.

No need for a compiler, or complicated execution instructions, the application, its tools, libraries and databases are all deployed on the Heroku server. Therefore, in order to access the application please click on the following link: http://part-time-job-csp.herokuapp.com

You will notice that the link directs you to the application web page. If you are not logged in, the first page that appears is the login page as it can be seen in figure 1:

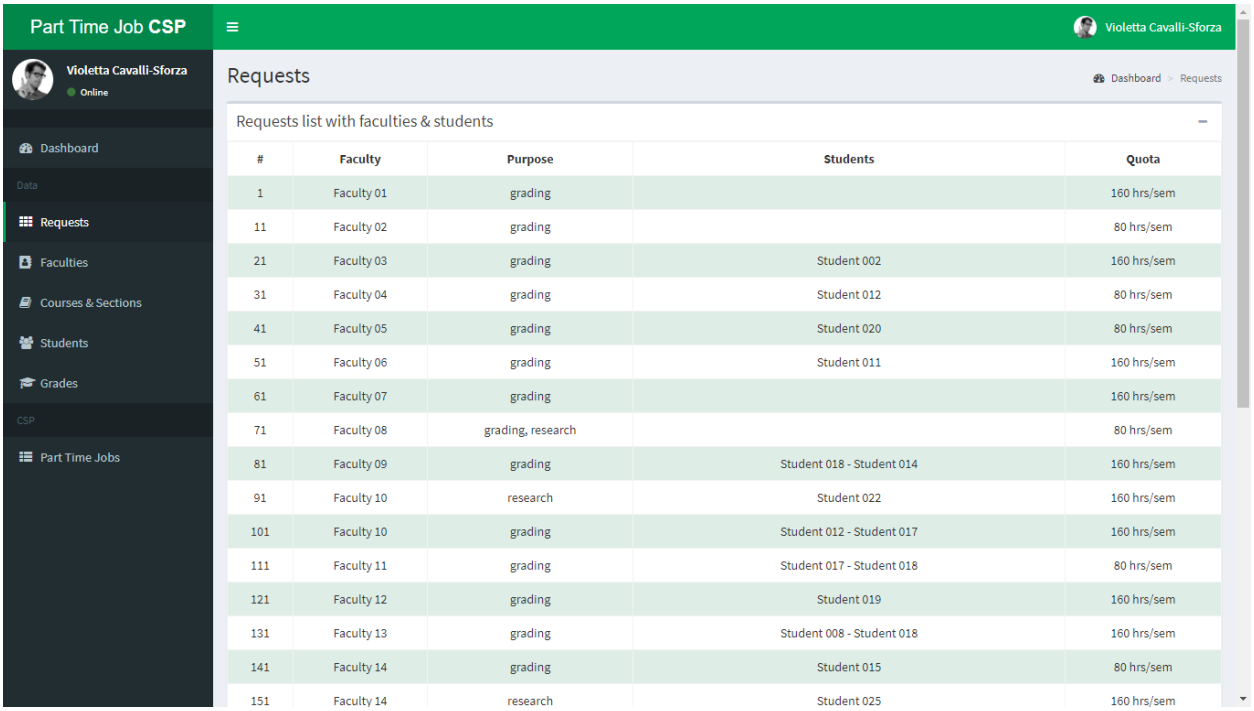
The login page contains the sign in credentials mainly the email and password. To login please input: vcs@aui.ma as the email, and for the password use “password” and then you can proceed in signing in. The dashboard of the application will look like the following:



On the left, the user can find the name of the person logged in, the different functionalities related to data, mainly that are: the list of requests made by faculty, the list of faculty who made the requests along with their course code and appropriate sections, the courses and sections that are relevant to the problem, the students that are in the list of part time job (whether contractual or payed) and finally their appropriate grades that they got in the classes they took, that are potential to be graded by them.

The dashboard contains the main information about the problem, mainly the number of faculty requesting a grader, the number of courses to be graded, the students available for grading positions, and finally the courses they took and their grades. Below this information, the user can find the demo video in case he wants to understand how the CSP application works.

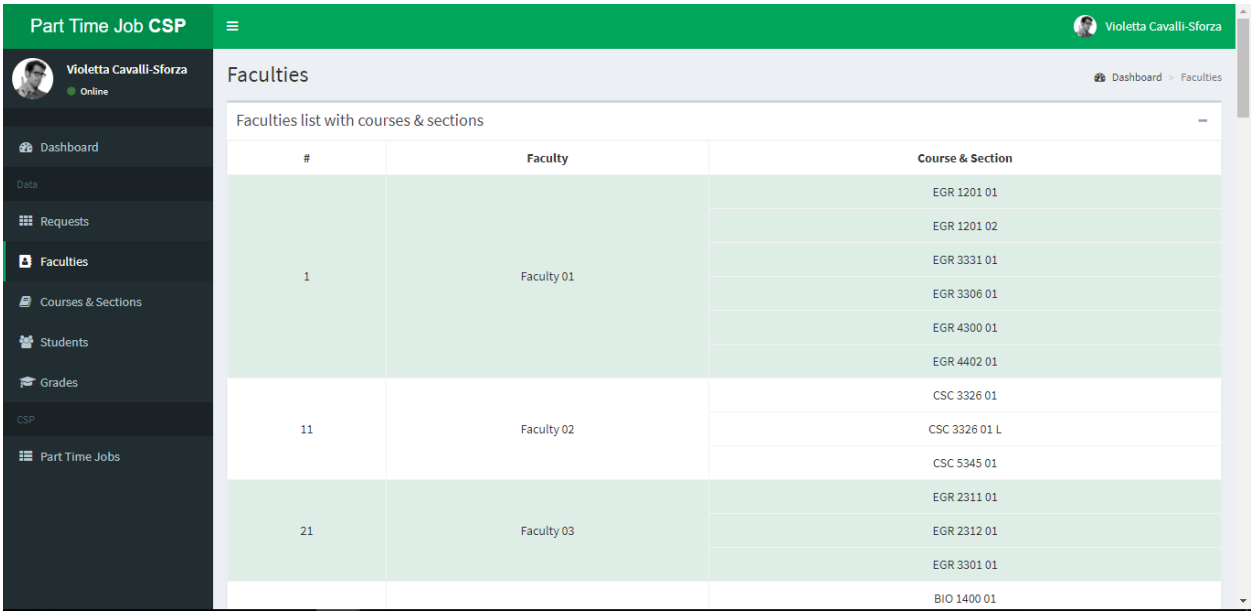
Once the user clicks on requests, the following page appears:



#	Faculty	Purpose	Students	Quota
1	Faculty 01	grading		160 hrs/sem
11	Faculty 02	grading		80 hrs/sem
21	Faculty 03	grading	Student 002	160 hrs/sem
31	Faculty 04	grading	Student 012	80 hrs/sem
41	Faculty 05	grading	Student 020	80 hrs/sem
51	Faculty 06	grading	Student 011	160 hrs/sem
61	Faculty 07	grading		160 hrs/sem
71	Faculty 08	grading, research		80 hrs/sem
81	Faculty 09	grading	Student 018 - Student 014	160 hrs/sem
91	Faculty 10	research	Student 022	160 hrs/sem
101	Faculty 10	grading	Student 012 - Student 017	160 hrs/sem
111	Faculty 11	grading	Student 017 - Student 018	80 hrs/sem
121	Faculty 12	grading	Student 019	160 hrs/sem
131	Faculty 13	grading	Student 008 - Student 018	160 hrs/sem
141	Faculty 14	grading	Student 015	80 hrs/sem
151	Faculty 14	research	Student 025	160 hrs/sem

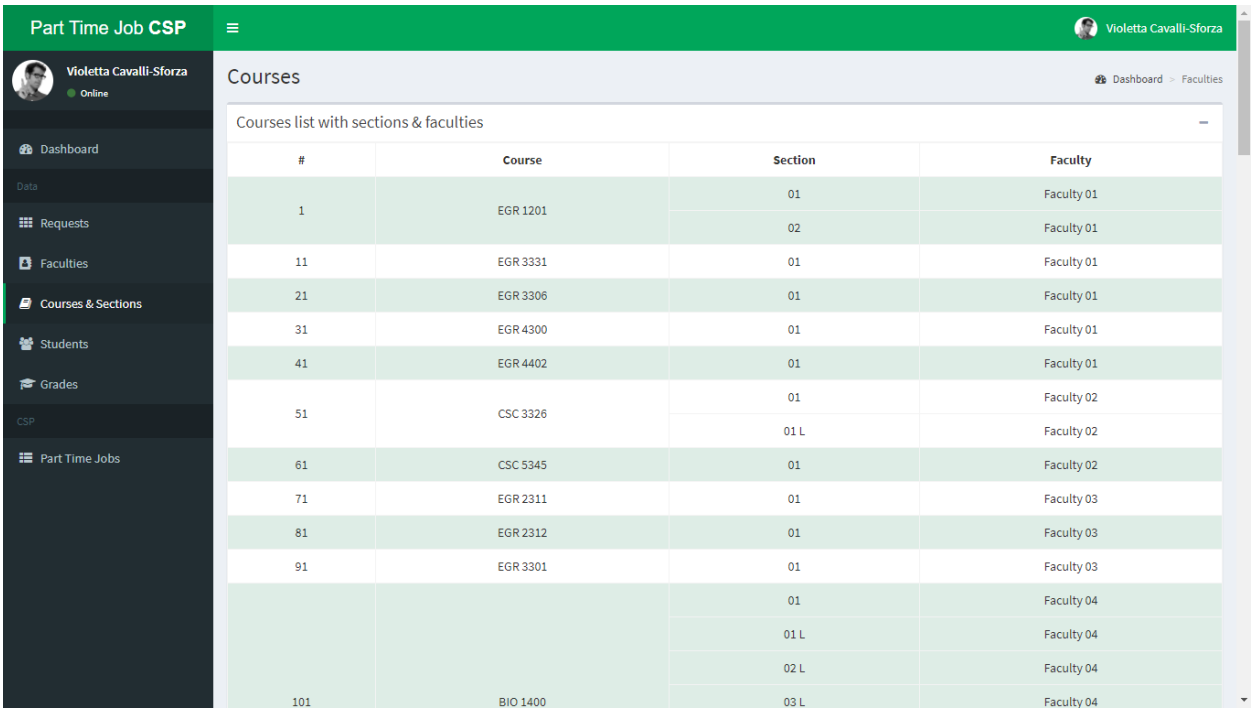
The requests page contains all the requests made by faculty inside a list. The request list logic and order is taken from the sample data provided along with the project description, it is stored the same format as the excel sheet inside MySQL database. The table contains the request ID, the faculty who made the request, the purpose of having a TA, a specific student, and the quota in terms of hours per semester.

When the user clicks on Faculties, the user can see all the specific information about the faculty, their classes they are teaching along with their appropriate sections as seen in the next figure.



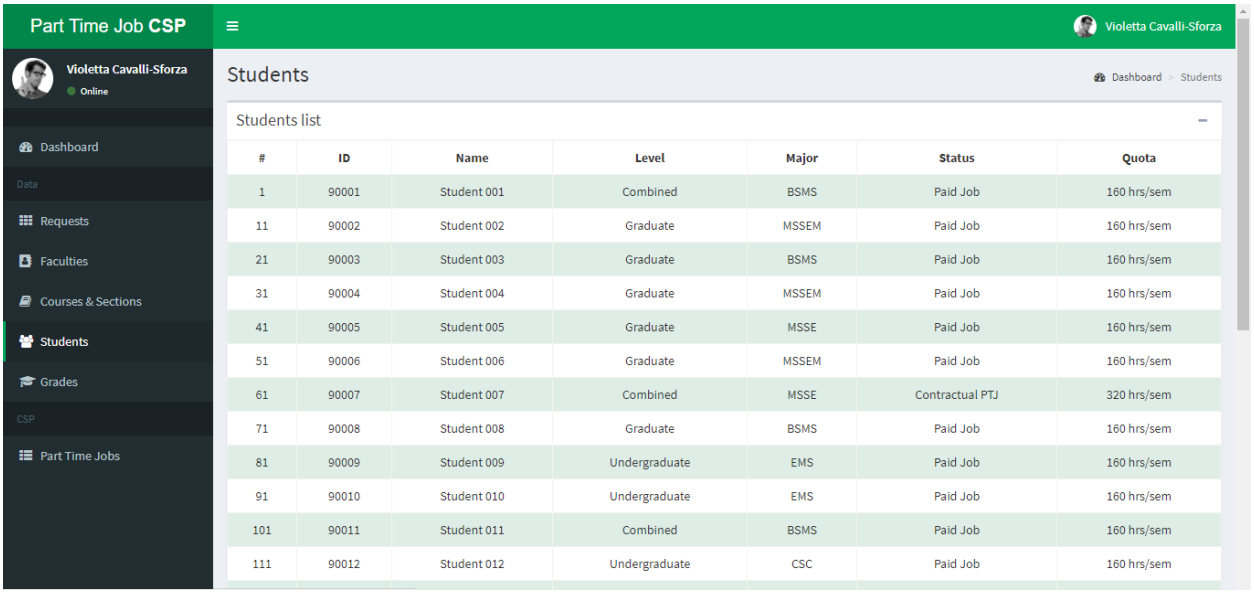
#	Faculty	Course & Section
1	Faculty 01	EGR 1201 01
		EGR 1201 02
		EGR 3331 01
		EGR 3306 01
		EGR 4300 01
		EGR 4402 01
11	Faculty 02	CSC 3326 01
		CSC 3326 01 L
		CSC 5345 01
21	Faculty 03	EGR 2311 01
		EGR 2312 01
		EGR 3301 01
		BIO 1400 01

For the courses and sections, it associates all the courses along with their faculty as it can be seen in the following screenshots.



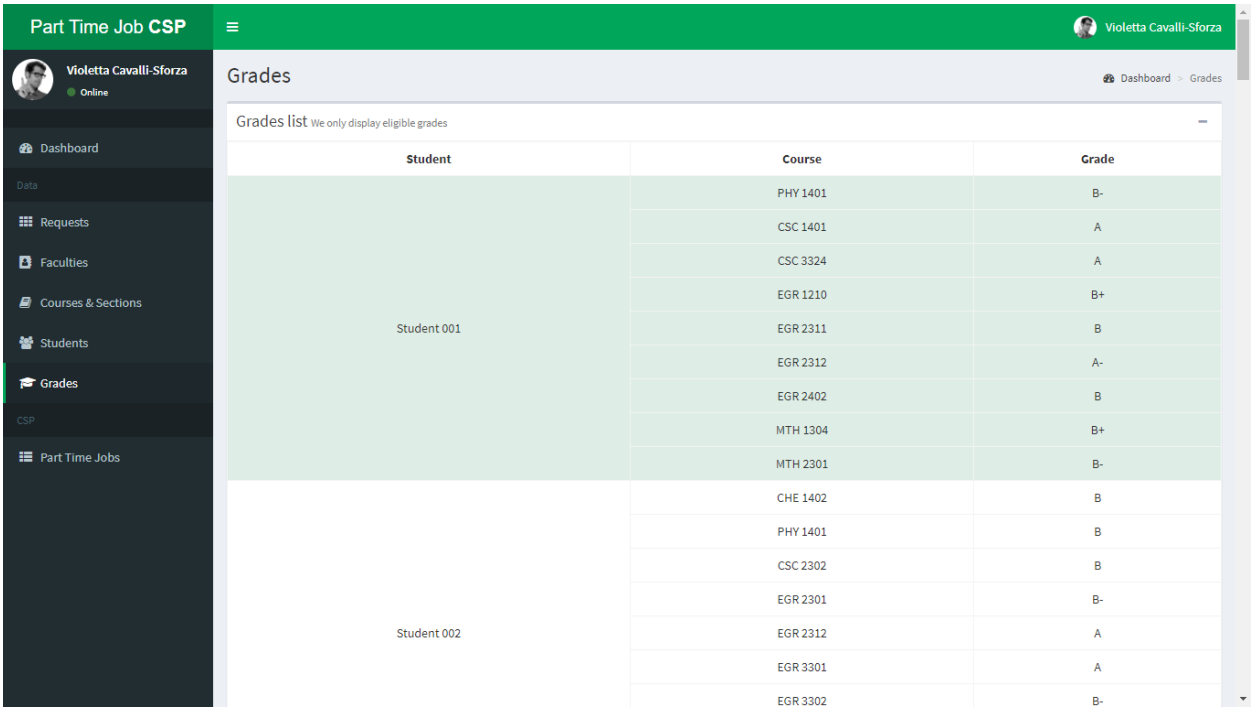
#	Course	Section	Faculty
1	EGR 1201	01	Faculty 01
		02	Faculty 01
11	EGR 3331	01	Faculty 01
21	EGR 3306	01	Faculty 01
31	EGR 4300	01	Faculty 01
41	EGR 4402	01	Faculty 01
51	CSC 3326	01	Faculty 02
		01 L	Faculty 02
61	CSC 5345	01	Faculty 02
71	EGR 2311	01	Faculty 03
81	EGR 2312	01	Faculty 03
91	EGR 3301	01	Faculty 03
		01	Faculty 04
		01 L	Faculty 04
		02 L	Faculty 04
101	BIO 1400	03 L	Faculty 04

The students section contains the students that will take the position of a grader/research assistant. In the table the user can find information about the following: the student ID, the name of the student, the seniority/ division of the student, the major, the status of the job (payed part time job, or contractual), the quotas, meaning the maximum allowed number of hours per semester that the student can work with.



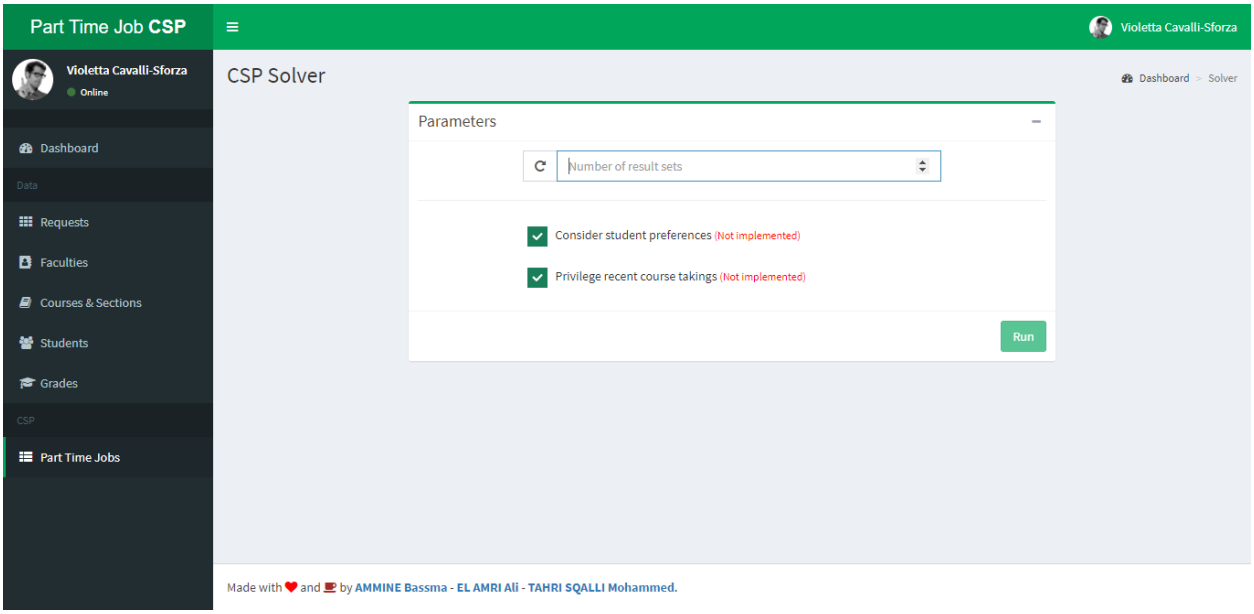
#	ID	Name	Level	Major	Status	Quota
1	90001	Student 001	Combined	BSMS	Paid Job	160 hrs/sem
11	90002	Student 002	Graduate	MSSEM	Paid Job	160 hrs/sem
21	90003	Student 003	Graduate	BSMS	Paid Job	160 hrs/sem
31	90004	Student 004	Graduate	MSSEM	Paid Job	160 hrs/sem
41	90005	Student 005	Graduate	MSSE	Paid Job	160 hrs/sem
51	90006	Student 006	Graduate	MSSEM	Paid Job	160 hrs/sem
61	90007	Student 007	Combined	MSSE	Contractual PTJ	320 hrs/sem
71	90008	Student 008	Graduate	BSMS	Paid Job	160 hrs/sem
81	90009	Student 009	Undergraduate	EMS	Paid Job	160 hrs/sem
91	90010	Student 010	Undergraduate	EMS	Paid Job	160 hrs/sem
101	90011	Student 011	Combined	BSMS	Paid Job	160 hrs/sem
111	90012	Student 012	Undergraduate	CSC	Paid Job	160 hrs/sem

The final list that is needed for the constraint solver is the grades list, the grade list contains the courses along with their respective grades of each student potential to be placed as a grader/researcher. The list looks like the following:



Output Interpretation:

Once the user consults the data needed for the constraint satisfaction solver, he can click on the last tab the "Part time job tab" that leads the user to the following settings page.



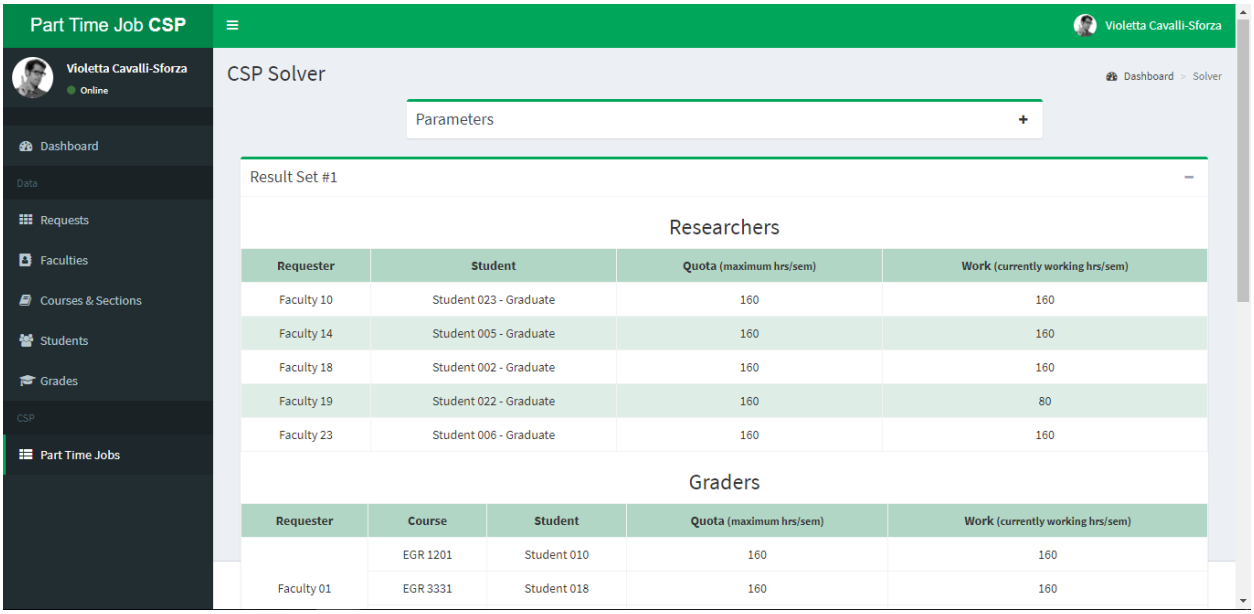
The settings page is actually the page where the user specifies whether the constraint satisfaction solver needs to satisfy only hard constraints or soft constraints. They are represented in the parameters by:

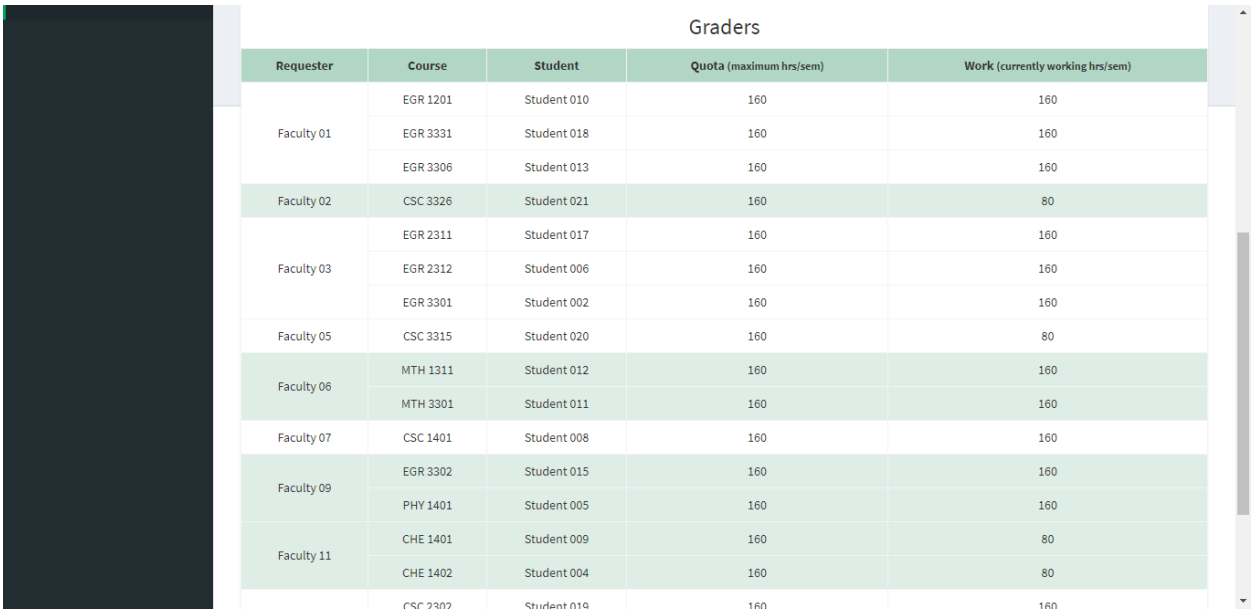
- The number of solutions that the user wants to have
- Whether the CSP needs to take into consideration the preferences of the faculty
- Whether the CSP needs to make assign students with the most recent course they took (privileging recent course takings)

Once the user has specified these parameters, he can proceed by running the CSP that runs in the background.

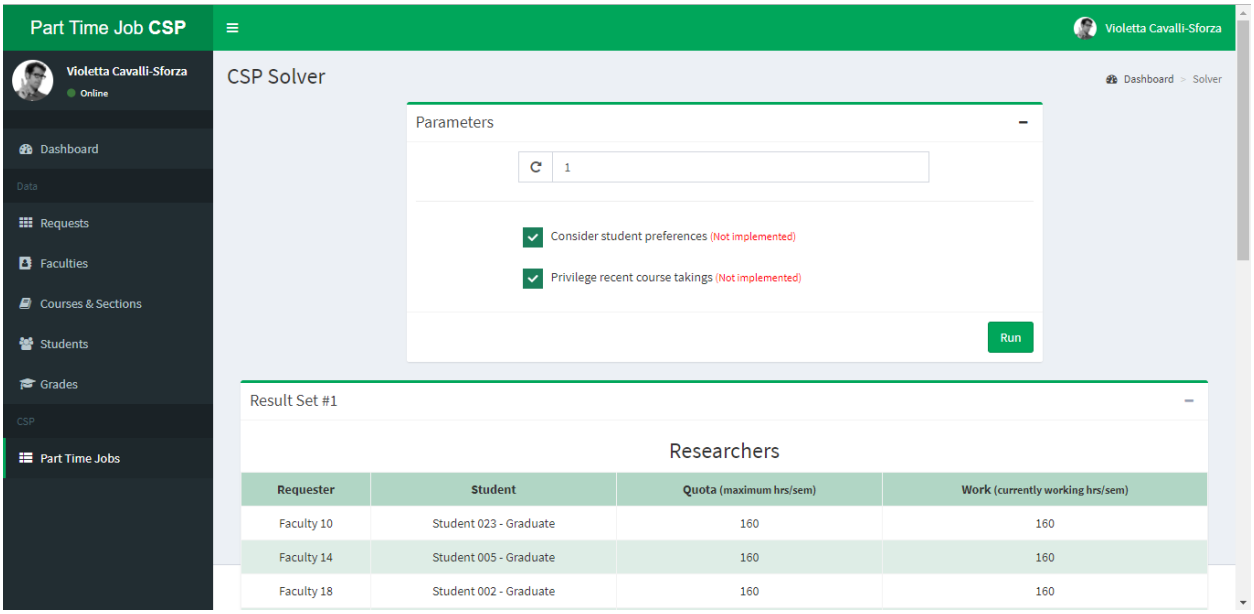
The processing might need some time depending on the constraints that the user has set, and depending on the difficulty of the data the solution usually takes 30 seconds to 2 minutes of processing so please be patient when running the CSP. The following is the solution page.

The solution page displays two aspects of the solutions, the first aspect is the researchers assignments. The researchers are separated from the graders assignments as shown in the following figures.





In case the user has forgotten what were the constraints that he has provided as input, he can expand the constraints tab by clicking on the + sign above to show the constraints tab as it can be seen in the following figure:



The results can be interpreted as follows: the table contains 4 tabs. The tab of the requester, meaning the faculty that requested the grading, the student assigned to him, the quota (represented as the maximum number of hours that the student can work), and the number of hours that the student will work for that specific faculty. If the amount of the work tab is half of the quota, this means that the student will be splitted between two faculties or he will be working only half the time needed.

Description of results:

The results that we get most of the time satisfy the hard constraints that we have set based on the instructions of the assignment and some of the soft constraints. In the case of not being able to satisfy the soft constraints, Choco Solver works with the idea that it needs to satisfy all the requests no matter what so it might sacrifice the preferences of the soft constraints. Following is our guess why some constraints are not satisfied:

- Inconsistency of data: we have encountered a lot of inconsistencies when trying to analyze how Choco Solver attacks the problem, among these inconsistencies: Professors requesting students that do not exist on the list, professors requesting graders for some classes where no student has taken that class, the count of students compared to the count of faculty is considered unbalanced for Choco Solver which has created a difficult dilemma that made the solver go running for a long period of time without finding a solution.
- The internal recusing process of Choco Solver. We have noticed that Choco Solver does not take the data provided to it from the database linearly, but does a different research process as it might be starting from a specific order different that taking into consideration the request list.
- No set execution time for the constraint satisfaction solver. Sometimes the CSP comes with the solution in less than 10 seconds however, sometimes it takes up to 3 minutes and the CSP keeps running before finding a solution. Following is a table describing the average execution time that we have tested needed to get a solution for the problem:

Result Set	1	5	10	100	500	1000
Time	0 Seconds	0.5 seconds	1 second	37 seconds	2 min 23 seconds	Infinity