**Artificial Intelligence Final Project: Write-up**

Intelligent Course Planner ([GitHub repo](https://github.com/adamliningerwhite/course-planner))

Adam Lininger-White

Douglas Webster

Project Summary:

Over the course of the semester we have been incrementally developing an intelligent course planner which is accompanied by a database of Pomona College course listings. Our project was broken up into three overarching categories: data collection, constraint-based satisfaction, and user interface design.

Our plan with regard to data collection morphed over the course of the semester as obtaining Pomona College course listing directly from the registrar and ASPC proved tougher than expected. Due to the difficulties of obtaining a large clean data set we decided to resort to the published listings on the ASPC website. To narrow our database to something reasonable for manual input we researched the most popular majors at Pomona College. With a list of popular majors – some of which include computer science, math, economics, and politics – we began inputting classes from the ASPC website, taking into consideration the following course related data: course code, name, professor, department, type, prerequisites, credits, meeting day/time/location, workload, and rating.

Once we had a concrete representation of a class, we began to implement the constraints which would form the foundation of our intelligent course planner. We started off small with constraints that did not depend on user input. Such constraints included not choosing a schedule with duplicate courses or overlapping times. Next, we needed to understand what the most important aspects of a schedule were to students. To do this we informally spoke with colleagues and got their opinions on the most essential aspects of a well-designed schedule. We found that it was very important to students to make sure that they took a specific number of core courses each semester to incrementally work towards completion of their major, a schedule which did not adequately meet this need would lead to backloading hard courses and unwanted stress. Additionally, students voiced a need for a tool to help them schedule classes around their extracurricular commitments and make sure that they did not have too many classes on one day. Lastly, students mentioned the annoyance of searching through listings to find courses which they had the appropriate prerequisites for. With these suggestion in mind we made sure to implement these constraints as the backbone of our solver and later extrapolated on student desires to include constraints such as the number of elective courses within a student’s major, maximum course workload, and minimum course rating.

Upon completion of the backend of our application we then moved to the user interface which would allow students to quickly and easily dictate their scheduling needs and desires. In order to avoid user error, we made sure to provide drop downs for as many fields as possible, only resorting to direct user input when absolutely necessary. Additionally, to decrease error we provide examples of valid input and if the user makes a mistake we have implemented our parsing with numerous try-catch blocks which will try to complete parsing of a given fields and if unable will report an error to the user in the unproperly formatted field and once again provide an example of proper input. Once the user has filled out the questionnaire a simple click of the submit button will transfer the information to a file and a sub-process will be started to generate a schedule. Once the schedule has been generated it will appear on the screen and in the case the where the preferences were unsatisfiable, we inform the user and prompt them to relax their demands.

Goals:

When proposing this idea as our final project we were interested in working on a project which was directly applicable to student life. We wanted to use the techniques we learned in this course to help solve a real-world problem. The importance of scheduling extends beyond the domain of course scheduling and we wanted to use this project to investigate a topic which we have not touched on in our other computer science courses. The goals of this project included: exploring the versatility of Prolog, learning how to connect multiple languages though the command line, contributing to student utility, beginning a larger project dealing with scheduling, and making sure our project was scalable. After the conclusion of the semester we plan on continuing this project and expanding upon it to consider more user defined constraints and fancy optimizations.

Differences From Original Project:

When we first proposed the project, we were under the assumption that obtaining data from the registrar would be a few emails away. However, after numerous emails and in-person meetings we had to abandon our hope for a complete database of courses listings. We had to settle for a manually inputted list of courses which contain many courses from the most popular majors at Pomona College. Additionally, due to not being able to obtain a proper set of course listings we had to abandon the idea of making our scheduler work for students at other schools. We had originally thought that it would be useful to meet the needs of students at a consortium but we quickly realized that course codes, prerequisites, and differing major criteria were a major road block.

Key Accomplishments:

With regard to the user interface we were able to develop a well formatted, easily understood, and interactive environment where students could quickly detail the most important aspects of their schedule. It was important to us to get as much valuable information as possible in a short questionnaire because we wanted to take away work from students, not add to it. Additionally, another key accomplishment was connecting our Java built interface with our Prolog schedule solver through the command line. After gathering student input, we create a sub-process detailing the files we want our solver to draw from. After we receive an answer, we parse through the results to find the most optimal schedule or notify the user that their constraints are overbearing and that they should consider relaxing a few of their demands. On the backend, we effectively harnessed the tools and strengths of ASP to structure the class database, utilize student inputs in checking constraints, deal with pieces of missing information, and smartly optimize results. For accomplishing all these goals, we’re very proud of our final project.

What We Learned:

Throughout this process we learned how to develop a semi-original plan from start to finish. We learned the struggles of obtaining comprehensive data and the inevitable unknowns of starting a problem from scratch. Over the course of the semester we had to learn how to design good Prolog facts which would lead to a well-designed database. Additionally, we gained confidence and competence working with Prolog and came away with an appreciation for the language an its versatility. Lastly, we learned how to connect our front-end user interface with our backend constraint solver which were written in two different language. While not a monumental feat it was something that neither of us had done before.

Evaluation Criteria (same as before):

* Research: 1
* Engineering: 3
* Analysis: 1

Research Portion (1/5) :

Scheduling is necessarily a constraint satisfaction problem, so we knew there would be papers and forums on the topic of using Prolog to implement scheduling tools. In fact, there were so many resources on the topic that we barely waded into the pool of available, potentially helpful materials. We quickly recognized a pattern in the few papers we read: each scheduling problem is quite unique, so the papers contained too much complexity and low-level detail to apply to our project.

Instead, the majority of our research focused on comparative analysis of existing scheduling tools and conversations with students to assess demand for certain features/functionalities.

Before diving into our project, we needed to develop a thorough understanding of the existing landscape. We want our project to be truly useful for students, so our tool needs to occupy a new role in the market and cater to currently unmet needs. The three most popular existing resources for Pomona students in this space are the [Pomona Portal](https://my.pomona.edu/ICS/Through_the_Gates/), [Hyperschedule](https://hyperschedule.io/), and the [ASPC Course Planner](https://pomonastudents.org/login). Each of these tools provides lots of information for each class and can help students visualize their schedules, but all demand serious effort from students to search the list of available classes and manually filter out inviable ones. As a result, we decided our tool needed to give more power to students to reduce the search space from the outset. Additionally, these tools are completely reactive to student inputs, which motivated us to create a bolder tool that attempts to anticipate a student’s desires by outputting what we think is the best class schedule given their preferences.

To accurately guess what schedules will best satisfy students, we obviously need to know what they care about most. Also, giving students more power in shaping the search process means nothing if they’re articulating trivial preferences rather than important ones. We informally presented our idea to nearly 50 students and sought their feedback, using the results to shape the inputs options in our UI application and the backend optimizations we carry out unseen.

Engineering Portion (3/5) :

Herein lies the bulk of our project and the fruits of our labor. Part 1 of this section offers a brief overview of the repository as a whole with descriptions of the files and directories inside. In Part II, we discuss the frontend Java GUI application where students express their scheduling preferences and provide important info on previous classes. This is followed by an explanation of how our Prolog files utilize this input to perform backend operations and search for the optimal result in Part III. Finally, we close with a manual for using our tool in Part IV that includes basic instructions and some notes.

Part I: Overview

Our GitHub repo contains several files/directories where people interested in our project should direct their attention. At the heart of our project is the “clingo” executable, an answer set solver which processes constraint problems and prints out solutions, which, in our case, are optimal schedules which we recommend to students. The prolog file “classes.lp” should be thought of as our class database. It contains information on several hundred classes, which the program searches through when seeking to build schedules. Our other prolog file “chooser.lp” processes user input to create constraints telling our solver which schedules are unallowable. Finally, the “ui-info.txt” file gives a brief description of each field in our Java GUI application and how we translate between student input and prolog constraints. The “/src” folder contains the java code that forms our UI, which we’ll describe in more detail below.

Part II: Java GUI Application

Inside our repository’s “/src” folder, you will find three Java files: ChoiceInputContainer, TextInputContainer, and InputWindow. The first two are simple helper classes, which ensure consistent appearance and performance across all text and choice fields. These classes are deployed in “InputWindow.java,” which performs three important roles in our program. First, it builds the GUI and creates fields where students can pass in search parameters. Next, InputWindow parses the information in each field and either (1) translates it into prolog facts/predicates which it then prints to a file “student-input.lp” or (2) prints an error message describing the proper input format if there is a problem in parsing. In the absence of errors, InputWindow then executes clingo and passes in the class database (“classes.lp”), constraint file (“chooser.lp”), and student preferences (“student-input.lp”) as arguments. Finally, it redirects the clingo output to a file “results.txt,” which we then parse to find the final answer set. This optimal schedule is displayed to the user in a splash page listing the classes selected by our backend processes. If the user passed in unsatisfiable inputs, then the splash page displays a message telling the user to soften their scheduling constraints.

Part III: Prolog on the Backend

When InputWindow creates a subprocess executing clingo, our Prolog files take control and perform the heavy lifting of choosing a schedule and making sure it satisfies the user’s preferences. Most of the work is performed by “chooser.lp,” with “student-input.lp” and “classes.lp” mostly passing in relevant prolog facts. First, “chooser.lp” picks a set of classes from the list that will be offered next semester; the exact number to choose is specified by a fact in “student-input.lp” called class\_num/1. It then checks this set of picked classes against a number of constraints to ensure that our selected schedule doesn’t violate the student’s articulated preferences. The basic formula for checking one of these constraints is:

1. Grab some piece of information about a picked class from the database in “classes.lp”
2. Use this piece of information in one of two ways
   1. As a stand alone datapoint
   2. As part of sum to compute details about our schedule as a whole
3. Retrieve the user’s preference for this constraint from “student-input.lp”
4. Check that our data point or summation doesn’t violate this preference

We hope the code for each individual constraint is straightforward and not too difficult to understand. To this end, we tried to use descriptive names and thorough commenting wherever possible. Finally, after checking all constraints, our “chooser.lp” seeks to maximize the average rating of all picked classes. Class rating is a terrific proxy for student satisfaction, so we feel confident that optimizing for this parameter will cause our program to return better schedules more often than not. Some classes in our “classes.lp” file do not have rating or workload data on the ASPC website, so we used weak constraints to prefer classes with this info without making it a demand.

Part IV: How To Manual

There are two ways to use our program.

Option 1: Running from Eclipse

* Use Eclipse to clone our [git repository](https://github.com/adamliningerwhite/course-planner) somewhere on your computer.
* Make sure the clingo executable and prolog files are located in the main project folder
* Open the InputWindow class in the src folder
* Run the program
* An interactive Java GUI should pop up where you can enter preferences and receive our program’s output

Option 2: Running from the Command Line

* Use git to clone our repository somewhere on your computer
* Cd into this new directory
* From there, cd into the /src directory
* Make sure the clingo executable and prolog files are located in this /src directory
* Execute “javac ChoiceInputContainer.java InputWindow.java TextInputContainer.java”
* Then run “java InputWindow”
* An interactive Java GUI should pop up where you can enter preferences and receive our program’s output

**Note:** Currently, our program only works with Mac and Linux file systems (i.e. those using the “/course-planner/src/example” file navigation syntax).

Analysis Portion (1/5):

Testing whether our program was correctly working and respecting both the user’s preferences and our hardcoded constraints proved more difficult than we anticipated. We employed a rather brutish method for testing each constraint. At the top of the “chooser.lp” file, you can see some of the predicates that we wrote to print out information about our picked classes. To test each constraint, we looked at it in a vacuum, printing out the relevant pieces of schedule information and commenting out all other constraints. We then ran the program with 5-10 unique, satisfiable inputs to make sure it would always find a valid solution. Next, we would give the program 3-5 different, unsatisfiable inputs to make sure it didn’t return any false positives. After testing each constraint individually, we gradually pieced them together and continued employing the same testing protocol along the way.

In terms of validating that our program is working for grading, I would recommend the following strategy:

1. Test each field in isolation to make sure they function correctly on their own (Note: the class number field must always be filled or else there will be no output)
2. Start testing combinations of constraints together. The “ui-info.txt” file gives a description of how inputs are translated to prolog facts, and our self-defined predicates are listed in “chooser.lp.” To make sure that our prolog predicate calculations are correct, open “results.txt” and cntrl-f search for the predicate of interest. You can then cntrl-f search for the user-defined fact and make sure our predicate doesn’t violate their input.
3. Give unsatisfiable inputs to our program and make sure it doesn’t return a schedule.

Analyzing the correctness of our program was difficult even for us, and we’re intimately familiar with the system’s innerworkings. Part of the difficulty is attributable to the nature of Answer Set Programming. Another great way to analyze the program’s correctness is by looking at the constraints in “chooser.lp” with a microscope to make sure their airtight.

Conclusion:

We are under no illusions that we’ve created a perfect scheduling tool. In fact, we are very aware of its limits: lack of full 5C dataset, missing pieces of information, difficult to use without some CS background. However, given the timeframe of this project and the limited access to necessary resources, we feel proud of the final product.

To quickly boast about our most impressive accomplishments, we…

* Created a fully functional Java GUI that robustly handles user input
* Expanded the list of scheduling tools at students’ disposal.
  + For students majoring in one of the departments included in our database, this program actually works for generating the “best” schedule possible given your constraints.
* Implemented a bunch of new constraints that had never been considered by other scheduling tools and will give student’s more power in the class search process
* Real students who tested our program gave positive reviews and expressed excitement about the possibility of a similar, more complete tool.

Over the course of this project, we’ve become passionate and invested in bringing our idea to life. We plan to continue working on the project over winter break, investigating how we could translate this project into a fully functional Web application. We also hope to leverage our existing program in future discussions with the Registrar and ASPC Office about finally obtaining comprehensive course data.