COMPANION PASS PROGRAM

A Senior Project

Presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Bachelor of Science in Industrial Engineering

by

Andrew Toney

June 2013

ABSTRACT

Companion PASS Program

Andrew Toney

At the end of each quarter the students of Cal Poly prepare to sign up for their classes for next quarter. This process is more difficult for seniors who must meet very specific requirements in order to graduate on time. Students can spend hours trying to find the “perfect schedule”, that one schedule that fits their personal schedule and their graduation requirements, while not over taxing themselves during that quarter by having too much class work. This Cal Poly explores the creation and implementation of an extra scheduling tool to address the weaknesses of the current scheduling system, PASS, and help students reduce the amount of time spent looking for that one “perfect schedule”.

The companion program makes use of MS Access and MS Excel, both common Office Programs, to create a system that offers more optimization options in choosing what classes a student would like to take for the next quarter. Working with several queries in Access and then changing those queries into constraints for Excel solver this Senior Project was able to create an ‘alpha’ level companion program that is function, but not fully tested or edited for ease of use by senior level industrial engineering students at Cal Poly.

Table of Contents

List of Figures

Chapter 1: Introduction

One of the struggles of being a student is planning out what classes will be taken in what quarters. This not only relates to a student's struggle to figure out which classes can be taken, but if it is likely that those classes will be filled. Essentially a student can plan out the perfect schedule for every quarter while they are a student at Cal Poly, and it will still be possible that they will never be able to use those schedules, due to a lack of class availability or a sudden need for many students in a given year to need that particular class. Cal Poly's PASS system helps mitigate these problems by providing an easy to use software that helps students make several schedules for their classes. The system Excels in a few areas:

- PASS Creates multiple schedule options based on the number of classes that it can fit.
- PASS allows users to optimize their choices based on a preference for what time of day they would like to be at school
- PASS allows for students to optimize based on interactions between other classes
- PASS allows students to optimize classes based on a given day or time across all days.

While these are useful things for students to be able to optimize, PASS is truly at its most effective when a user has a vast number of options. Having more classes, with more times, and more sections means that pass can make more schedules and narrow them down to the best choices based on the optimizations.

PASS does however have some weaknesses. The PASS system is most useful when a student begins his/her career at Cal Poly, and when they are in the middle. This is because at the start of a college career students have many more options in what classes they can take, as most General education classes, and introductory major classes are available. When a student is nearing the end of their sophomore year, and the beginning of their junior year, they will then also have a lot of potential choices as they will be open to more of the upper division classes in

their given major, while still having the option to take on a wide variety of General Education courses as well. Therefore: PASS becomes least effective when a student has very few choices or few options. Senior level students have limited path options when choosing which classes to take as the class progression becomes more and more linear the closer a student nears to graduation. At that point, having a greater number of schedules becomes less important and having a schedule that fits the needs of the users preferences for a select class become more important. This is where PASS becomes less useful.

That is why this author has worked on building PASS Companion Program, to address these weaknesses and help students find their “perfect” schedule faster.

The PASS companion program will address the issues with PASS by including the following functions:

- Students will be able to select nine classes and assign preference values to each class based on how much the student wants to see that class in the optimization
- Students will be able to form groups amongst their chosen classes

Creating a companion program to meet these two points will allow students to select more classes than they would with PASS and select form among the few they need most, while filling the spare time with a class instantly, as opposed to using PASS where a student would have to continually re-edit the classes in order to look at different schedules with different classes.

This Senior Project will cover some prior literature on working with optimization scheduling problems and other online software scheduling systems like PASS before explaining the methodology on how to solve this problem.

Chapter 2: Literature Review

Overview

The difficulty students have in both getting classes and choosing which classes are best for them has long been a topic of complaint and debate amongst students and their peers. “I have to stay an extra quarter because I can’t get this class!”, “Did you try everything?”, “Well I tried several different schedules with the 7 or so classes I need, but none of them seem to work with the one class I have to take.”

The problem with creating schedules for the university’s classes has been the topic of a few senior projects already, but they have focused mostly on the faculty side of things, helping professors gain preferable time slots for their classes in the classrooms they’d most like to use.

There is still room for improvement on the student’s side of the scheduling system.

The theory behind electronic scheduling for a university’s students has been studied and documented in several pieces of literature, and can be broken down by the methods used to solve this a scheduling problem like this one, better known as a scheduling under hard constraints and having an NP degree of complexity. The methods used are divided between optimization and heuristic techniques.

Optimization:

Integer Programming

Dynamic Programming

Goal Programming

Heuristic:

Genetic algorithms (GA)

Local Search

Simulated Annealing

Evolutionary approach

The practice of making successful scheduling processes is a large field of study that has been explored in many papers, particularly in discovering the use of combining algorithms. Studies look into the efficacy of combining the uses of the genetic algorithm or the Evolutionary algorithm combined with local search or simulations [Abdullah and Turabieh (2008), Wang et al. (2008), Irene et al. (2009)]

It is perhaps easiest to see the differences in the methodologies when reading the paper "A comparison of Course Scheduling Methods" by Ojha and Walker (2000). The authors compare four types of methodologies in attempting to solve course scheduling problems. They reach more than a few conclusions on each. Their findings were as follows:

Hill climbing – Flexible, and gives better results, but ultimately takes more time to process

Linear programming - Able to generate good results based on some samples data they authors used for their study, but proved rigid in its structure, and was difficult for those not already trained in LP.

Tabu search – is both flexible and produces the best results.

Research has covered a lot of the subject in other papers.

Related literature abstracts

In "Linear programming, a concise introduction" by Thomas Ferguson covers the basics of linear programming, a paper discussing several types of potential problems linear programming can solve, things like transportation

problems where you are trying to find the minimal transportation costs when trying to send a fixed amount of material to factory i through route j. More importantly however, and something which is more applicable to the work in this project it covers what is commonly called, the optimal assignment program.

The optimal assignment program basically says that I have x people for y jobs and each worker x has a preference for job y. The essential issue in developing a program to help students get the classes they want based on preference is exactly the same. There are "x" classes a student can take, and the student has a preference for each class, but the classes are restricted by their time slots "y".

In "Decision Support System for University Course Scheduling." by Matthew Pesesnti 2002. A former Cal Poly student goes over the methodologies in developing a decision based aid system using Microsoft Access to help schedule classes. The tool was, while designed to help the scheduler, it was not designed to replace them. So it did not make schedules based on preference, but it did instead create information in a tabular format to help the scheduler determine all the effects of the schedule they were choosing. Pesesnti did not include an automated scheduling feature, and in specific circumstances his paper highlighted complications with the program where it wouldn't detect scheduling conflicts. The inability of his decision support system to detect these conflicts kept his final product from being implemented by Cal Poly, and left it as a Senior Project. The following conditions could occur in the system without error messages warning the user that the change they just made was infeasible:

> Multiple classes could have the same teacher teaching the classes at the same time. One class can have multiple teachers
>
> A single class can be taught in multiple lab rooms.

In a study done by Ghaemi et al. in 2007 researchers applied a modified Genetic algorithm and Cooperative Genetic algorithm to help them solve the University Time Table issue. In the study Ghaemi et al. work with two different approaches to the problem of minimizing conflicts in the university's time table. The first approach is the use of a normal genetic algorithm, which is very

effective, but requires a large amount of time to complete. With that in mind the researches tried to find a better way and noticed a trend. Those that integrate a two-stage strategy that first evaluates the feasibility of the plan and then optimizes the soft constraints of the problem using operators that restrict the search function to feasible areas will typically outperform the algorithms that will evaluate both sub-problems at the same time by using the weightings from the evaluation functions. The study also points out that there are plenty of studies on researching algorithms specializing in the optimization of soft constraints; however, the validity of these studies is questioned because there has been very little focus on producing algorithms that specialize in looking for the feasibility of a system in the first place.

In "Interactive Timetabling System Using Knowledge- Based Genetic Algorithms" by Hitoshi Kanoh and Yuusuke Sakamoto their research covers proposing a new solution to the university timetabling problem based on using genetic algorithms with an installed knowledge base, which would be, as described by Ghaemi in the previously covered study "effective but slow". Kanoh and Sakamoto propose that the use of a predetermined knowledge base could make the amount of time necessary to implement a genetic algorithm, lower by using a series of partial solutions and the algorithm to make a full solution much more quickly. The proposed method includes domain specific knowledge, but can be applied to a variety of other real life large scale combination optimization problems, thereby having applications outside the specific problem they work on.

In "University Course Scheduling Using Evolutionary Algorithms" Mohammed aldasht et al work on a new heuristic based evolutionary algorithm and apply it to what the paper calls "the university course scheduling problem," which is defined as the problem in finding a feasible and comfortable timetable schedule for everyone. The research topic is unique because it uses an evolution program algorithm, which the paper further describes as a "stochastic optimization strategy similar to genetic algorithms." The main difference between the two is that while evolutionary programming insists on behavioral links between parents and their offspring as opposed to trying to emulate specific genetic operators that are observed in their environment. The paper defines the

problem and determining its constraints under which the solution should be found. The paper then describes the problem model with a set of courses, rooms, professors, and student groups. Lastly the proposed methodology is used on a real set of data from one of four departments of the university (i.e. engineering, math, english, computer science, etc.). The results from this paper show that the methodology used grants a greater amount of search space for the problem, which gives more options for optimization of the schedules than if done manually. The schedules obtained from this method also show that several potential complications in the scheduling process can be handled with ease under this new system.

In "Optimization Of University Course Scheduling Problem With A Hybrid Artificial Bee Colony" Adalet Oner, Sel Ozcan, and Derya Dengi the researchers address the course scheduling problem by using a hybrid algorithm comprised of a heuristic graphic nod coloring algorithm and what is called an artificial bee colony algorithm. The study performed is one of the few that uses the artificial bee colony algorithm to solve discrete optimization problems, and as of the studies date of completion, it is one of the first applications to use artificial bee colony to solve the university course scheduling problem . The solution begins with a basic heuristic algorithm of node coloring to develop feasible solutions for the course scheduling problem. Those feasible solutions are then run through the artificial bee colony model as food sources, where it is then used to improve the feasible solutions. Within the model the employed and onlooker bees are directed or controlled very specifically in order to avoid conflicts within the course time table. The solution is tested using real data from a Turkish university. The results from this experiment demonstrate that the proposed hybrid algorithm yields excellent schedules.

In "Course Scheduling System Design and Implementation Based on Genetic Algorithm" by Hou Ming and Chen Qi the history of the course scheduling problem is covered extensively and examined to find new methods of solving a problem long covered in other papers read by the authors. The research details a new method based on a new genetic algorithm designed based on the flaws of past systems researched in the history of the problem. The researchers, after

many trials and tests was continually improved throughout the testing, making it an improvement on the traditional genetic algorithm. This improved genetic algorithm is designed solely to solve the course scheduling problem and the test results from the researchers show that the designed system can meet the needs of several actual colleges, hopefully this redesigned genetic algorithm can be used in college scheduling systems everywhere.

Current software packages

Orologio

The Orologio class time tabling system is a software package dedicated to creating class timetables for its customers. The software package provides a unique product because unlike other scheduling software products that are dedicated to class scheduling, their product will automatically build schedules for you based on your choices. The program is treated like a linear analysis problem, you tell it what resources it has to work with, number of days in the school year, what days are holidays, how many hours you can fill each day(in Cal Poly's case this is between 7am and 9pm, as the earliest classes can start is 7am, and the latest they can end is 9pm). Then you fill in how many classes there are, how many sections there are, how long each section is, so a 4 unit class with 3 lectures and 1 lab would have the inputs of lecture 3 times a week for an hour, lab once a week for 3 hours. Finally you'd input the number of professors or teachers and what classes they could teach, and tell the program to craft a schedule. Unfortunately this process is NOT preference driven. So while it will come up with a schedule that allows for no overlap between classes taught by the same teacher, and will try to make as little overlap between classes as possible, so that students won't have too many conflicting choices, it doesn't give the professors the ability to specify when they'd like to work, or any other preference modifiers.

ScheduleWhiz®

Schedule whiz, an online scheduling program like Orologio, offers several features useful to faculty in their methods of class scheduling and organization. The program offers scheduling process for up to 1000 units of university courses, as well as consulting on how best to use their software. The software has some unique features, like the ability to follow a course flow, meaning students shouldn't be able to sign up for a class until its first prerequisite is met. It also covers the basics of course scheduling software that all the software scheduling programs seem to offer, things like teacher information, course times and days, and matching specific teachers to classes. Unlike Orologio however, schedule whiz takes into account the preferences of professors. Professors' preferences can be recorded in their profile in the program and the process takes these into account when making a schedule.

Free online software programs

The two programs mentioned above all scheduling programs for universities to schedule their class listings for professors per quarter/university and the algorithms, methods, and interfaces are useful for the purposes of this senior project, as the goal of the project is to design a companion program with an easy to use interface that will address the issues of the students trying to register for classes; however, that is just the problem with the Orologio and Schedule Whiz programs, they are targeted at universities class scheduling, but not how the students will schedule the classes based on their preferences once the college has chosen the times for each class. So while there may not be any programs for sale that addresses the student needs, most universities will provide something similar to Cal Poly's PASS program, as well as several other online sources that are free to use.

While there are several free to use websites for scheduling classes at universities, many of them are often unusable due to various small issues. Some don't have the university you are attending, some don't update in time for you

registration rotation, some will list every class, but won't tell you when a class is offered making it a painstaking chore to look up every class you could take, and then checking to see if it's offered before registering for your courses.

Since there are so many free online process that students can use of varying strengths, but few of them have the quality, easy-to-use nature that students want. Websites like www.koofers.com, which require a facebook account to sign in, is not always up-to-date, and makes constant mistakes is not the type of scheduling system this lit review is focusing on. One particular site that has garnered a lot of attention from several UCs in the past few years, and is a quality site, is courseninja.com. Started at UC Berkeley course ninja is very much like Cal Poly's PASS system, and suffers similar restrictions, but does what PASS does for students at UC Berkeley, UC Davis, UC Merced, UC Santa Barbara, and UCLA.

"It's really been kind of a niche thing to develop, sure there are some programs similar to us [ninja courses] for commercial use, but really not many quality programs for students, and at UC Berkeley you have a lot of choices, so it's useful to have." –William Li(Co-founder of ninja courses)

Ninja courses is the kind of scheduling program every student would like to have. Like PASS it will only show you the classes you have offered that semester, and will propose several viable schedules to take like PASS. Some disadvantages are that the number of preference options are fewer than PASS, while PASS allows you to schedule which days you would rather not have, whether you want more or less time between classes, and many other options, Course ninja is limited to only two potential preference modifiers:

1) Keep my classes in the evening
2) Keep my classes in the morning

While this is better than no options for preference optimization, it is not stellar either, but it's the best online free scheduling system for students this research has found.

Chapter 3: Current Process

The current process used by students to help schedule classes for quarters is to use the online software PASS. PASS is very good at what it does, bust as mentioned earlier in this Senior Project, it often works best with students who are in their freshman year, or students who are in their later sophomore, early junior years, because it allows for a wider variety of schedules. When a student begins to use PASS to sign up for classes in their senior year it will usually take much longer to find a viable schedule as the student will have to make sure that all of the classes they are required to have completed in order to meet the prerequisites for future classes, or graduate on time will have only one or two sections and therefore be much more difficult to work around because PASS will only permit a student to sign up for classes they will take, and doesn't identify the classes that create time conflicts.

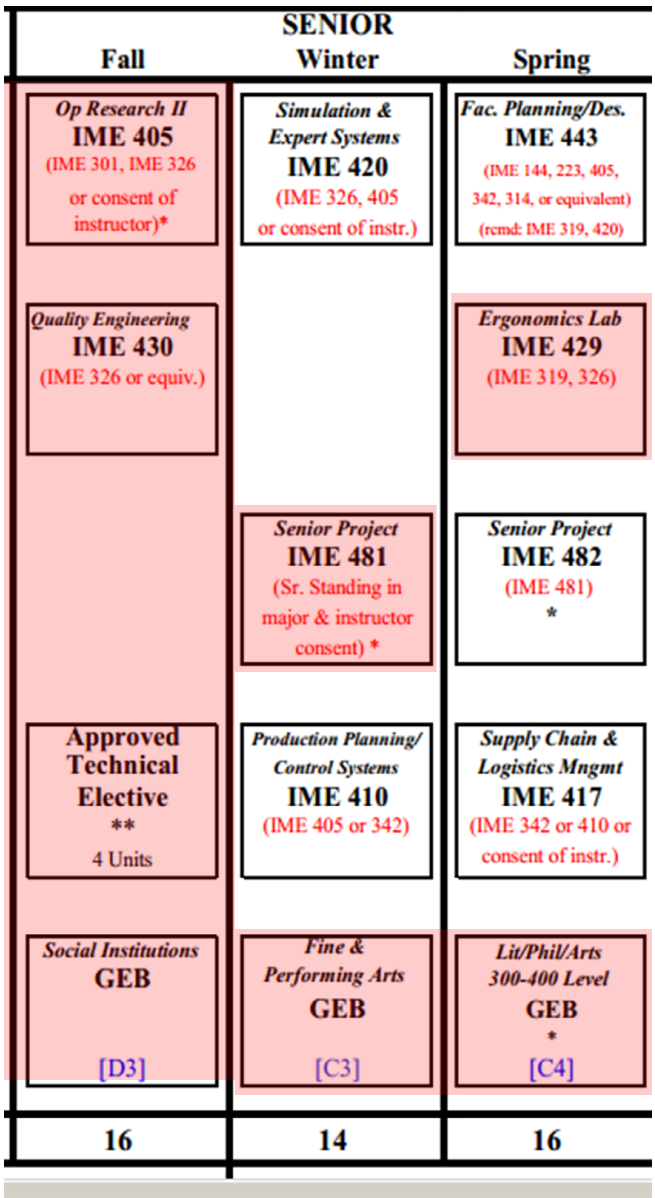


Figure 1

For example: Figure 1 details the first quarter as a senior at Cal Poly from the most recent flow chart. Classes highlighted in red are classes that a student can sign up for in their first quarter. If a student signed up for all of those classes on the hopes of forming a schedule, PASS would see that there were 8

classes and assume the user was trying to build a class schedule of 8 classes. Making a schedule of those 8 classes without time conflicts is impossible.

PASS would return the error message that there are no schedules for those 8 classes that could make a time conflict free schedule, and would leave it up to the scrutiny of the user to find out what class conflicts with what other class and refine the search for a better class schedule. This means that a student who had 8 choices, and had to choose 5, they could make a total of 56 different schedules. That's a lot of schedules to scrutinize and check to see if they are even feasible. It's entirely possible that a student would have to take a class like IME 481, which has one section, and find out that It conflicts with another class, and then the student will have to figure out a replacement class, or just take fewer units that quarter.

The whole process start to finish can take a lot of time; students will, on average look through 4 of these combinations and accept the first reasonable schedule they come across, not even considering that the 5th schedule might be even better for them. Figure 2 and Figure 3 details two questions from a survey given to senior level students on the experiences with PASS (See appendix A for full survey results and details).

The Survey not only revealed that students will only look through, on average, 4 schedules, but will also spend a lot of time going through those schedules. With almost 75% of students spending a half hour to an hour or more

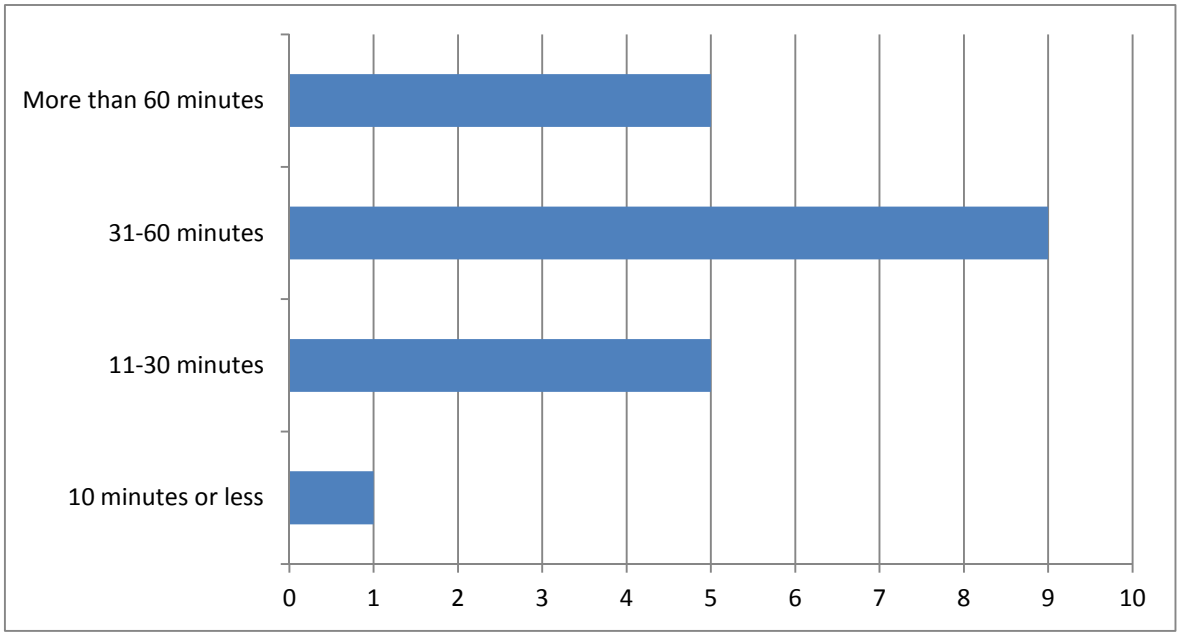
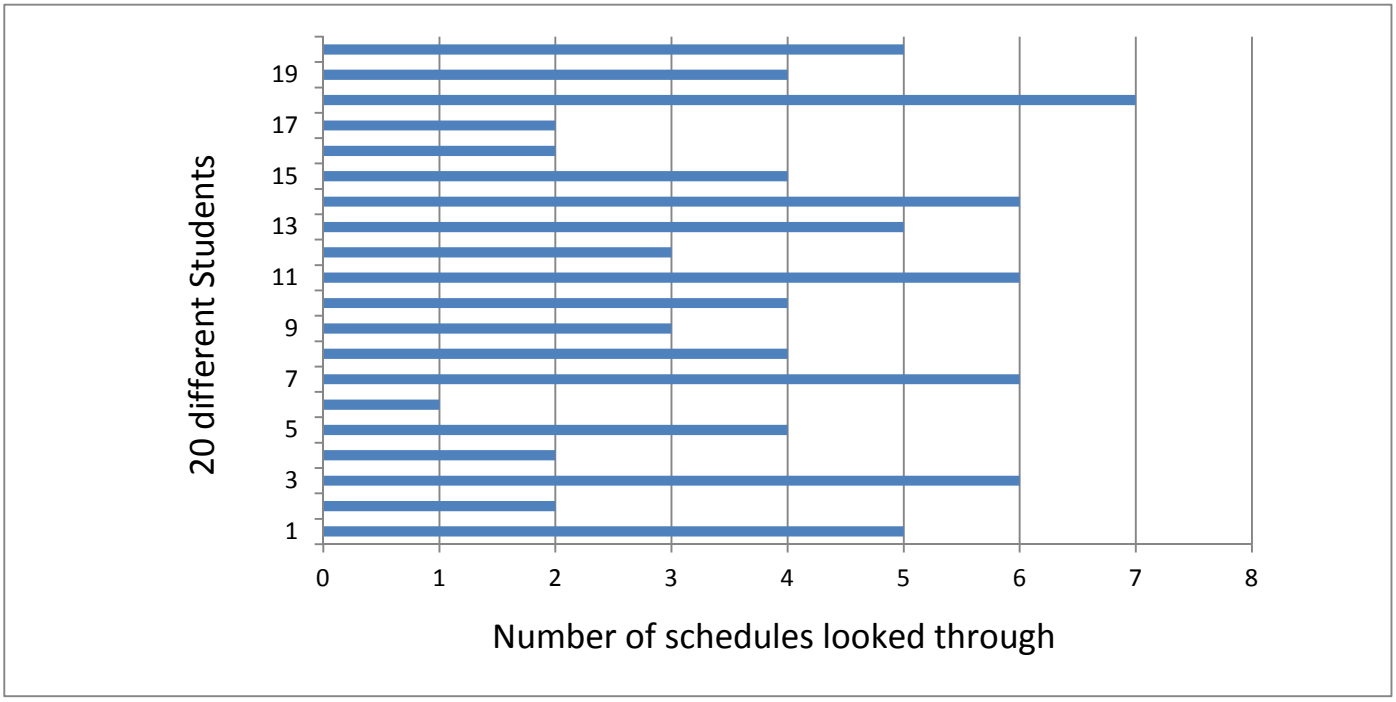


Figure 2

on their schedules, it's not surprising that they do not go through more of the 56 options they have open to them.

The survey speaks for itself. Students who use PASS only scratch the surface of the schedules open to them, and most students spend a long time looking into those schedules, and because of this designing an extra scheduling tool to help students find their "perfect" schedule faster is an important project.

Figure3



Chapter 4: Methodology

Methodology overview

When looking at this problem, one of the first questions was what medium should be used to design the companion program. It had to be a program that would most likely be on every student's computers, or at least be accessible by most students, and would have to be able to detect scheduling conflicts, and then form those conflicts into constraints for some optimization function to create the "perfect" schedule. Using principles learned here at Cal Poly in IME 312 and IME 301 this author decided to use Microsoft Access, and Microsoft Excel in a two part process to form the companion program. MS Access was chosen because it's Databasing capabilities are an excellent way to store, and quickly edit class data, and it's queries can find course time conflicts. MS Excel offers the solver Add-in, an Add-in that allows for optimization given specific constraints, which will be imported from MS Access, to identify the ideal solution. The methodology can therefore be broken into two parts, the MS Access portion of the project, and the MS Excel portion of the project. MS Access runs on a few programming sub routines and purposes:

- Create the basic structure of a linear analysis problem by marking all classes chosen by the user, including the sections for each one, and the preference values for that class.
- Create Basic constraints: Lab constraints, section constraints, unit constraints
- Utilize the queries in Access to find all time conflicts between the chosen classes by the user and store them into the schedule table as constraints.

After the above conditions are met through queries and sub routines in the VBA programming of the database, a table is created and copied into MS Excel which:

- Is edited to fix translation issues from Access and Excel
- Has the solver add-in set up to work on the table
- Produces the "perfect" schedule for the user

Methodology: MS Access: User Interface(UI)

Appendix B goes over the user interface in complete detail, explaining the function of all pieces, and object names in the form, however there is a reason for designing the UI the way it has been designed.

When design on the UI was started the idea was to include 1 more class than the maximum number of potential choice for a first quarter senior, as sometimes students will skip classes along the flow chart, and allowing for 1 more class assignment than most students would need. The grouping layout was placed next to the preference numbers because the most important part of the UI is first to choose the classes the user wants to see optimized into a schedule, the second is the preference rating of the given classes so as to know which classes are the most important, and the third is the group functionality, which unlike the other two is completely optional. Students will only use the group function if they need it for some purpose. Based on survey information from appendix A(Question 4) It seemed most likely that the group function would serve students as a 'find fill in class' option.

For example, if a student is signing up for courses and has a schedule with 3 classes already in it that they would like to have no matter what, but are looking for a fourth class, and do not care what class it is, they can select the three classes they must have, give them all a value of 10, and then create a group of classes that they do not care for and have the solver select one of those classes that happens to fit with the three they know they want.

Methodology: MS Access: Finding Course Conflicts with 135 queries

The greatest benefit to MS Access is the ability to use queries to sort through data and find important information, like course conflicts. Doing this in Access can be difficult and in this Senior Project works based on a compilation of several queries. Figure 4 shows the logic progression between these queries.

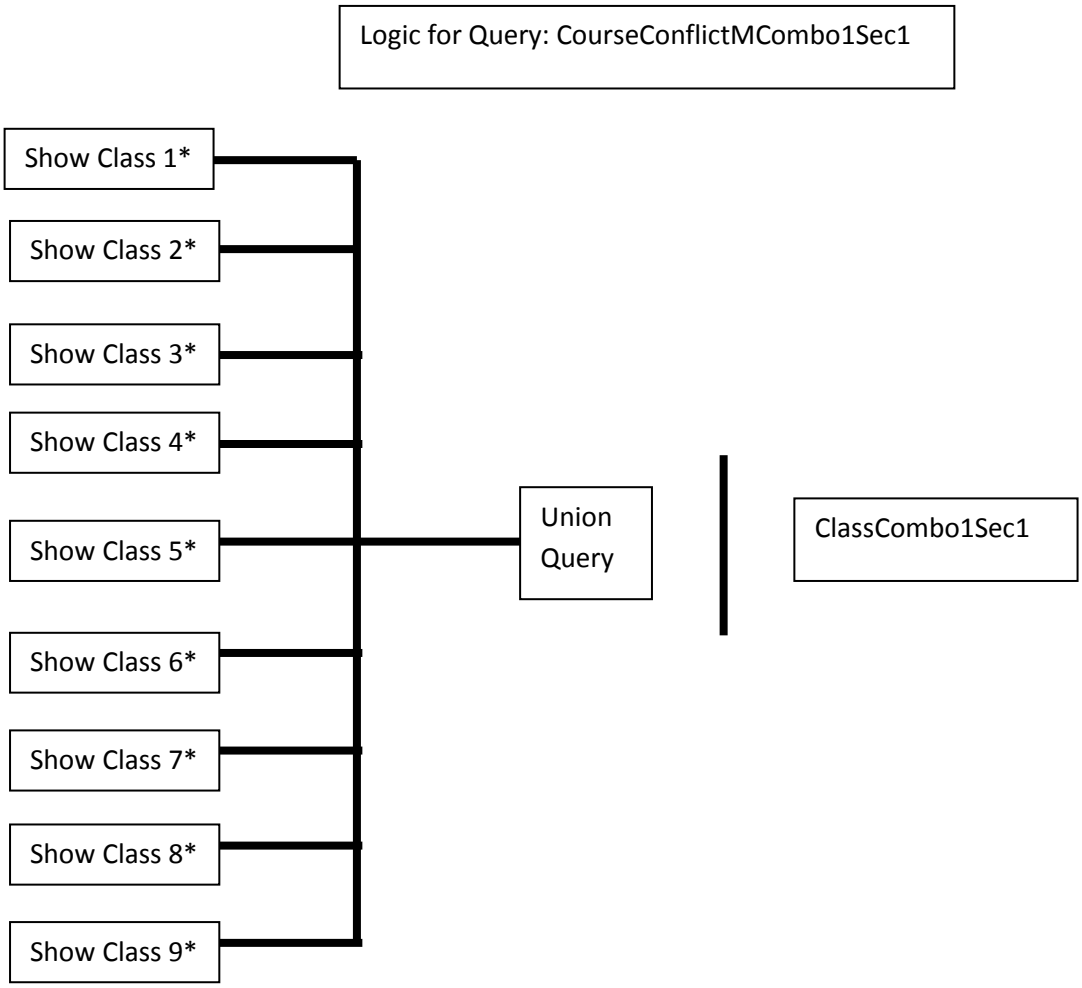


Figure 4

The show class X* query is a query that stores all the values of a class whose name is the same in combo box X e.g. If a student inputs IME 312 in Class Combo 1 then show class 1* will show all fields for the entry which has a class name of IME 312, things like start time, end time, day the of the week the class takes place, type of class, lecture or lab, see appendix B for more info on the database fields.

Union Query, which is a compilation of the show class X* queries, displays all information for all sections of the classes selected by the user. Then the query CourseConflictMClassCombo1Sec1 looks for time conflicts between the classes chosen by the user, stored in union query, and saves and conflicts between these other queries and the first section of the first class chosen by the user. This means that if IME 312 is the first class selected the query

CourseConflictMClassCombo1Sec1 will show all time conflicts between the first section of IME 312 and any other sections that are on Monday and conflict with section 1's start and end time under the following qualifiers:

Only show a class from union query, if that class' start time is less than or equal to the end time of IME 312 sec1, and if that classes end time is greater than or equal to the start time of IME 312 sec1.

This Query is then repeated twice more, once for section 2 of a class, and once for the third section of a class. Those three queries are then repeated for each of the user's choices, combo boxes 1-9, making a total of 27 queries.

The query is then remade for each day of the week, Monday through Friday as there are no classes on the weekends, giving a total of 135 queries to detect time conflicts for every section of the 9 classes chosen by the user, on every day of the week.

Methodology: MS Access: Subroutines NameClasses() and InputPrefVals()

The companion PASS program uses 135 queries to locate time constraints but it uses VBA programming in order to create the table to be transferred to Excel's solver function. The VBA coding populates the table trough a series of subroutines beginning with the subroutines NameClasses() and InputPrefVals(). The purpose of these sub routines is to create the first line of the table, which identifies which class and which section of each class is associated with what row, and to insert the preference values given by the user into the table under each respective class. So for example if a user inputs two classes, IME 405 and IME 417 with preference values of 8 and 6, the subroutines NameClasses(), and inputPrefVal() would output a table that would look something like Figure 5.

Figure 5

Schedule					
A	B	C	D	E	F
IME 405			IME 417		
Sec1	Sec2	Sec3	Sec1	Sec2	Sec3
8			6		

The values highlighted in yellow represent the table values created by the NameClasses() sun routine, and the table values highlighted in red were made by the subroutine InputPrefVal().

To better explain how the NameClasses() sub routine works, there a snippet of coding from appendix C copied below, anything with the single quote in front of it is a comment to better understand the use of each line included here to explain its purpose.

```
'Checks to make sure the combobox isn't null, then stores the name of the class if it isn't null

    If Not IsNull(ClassCombo9.Value) Then

    Y = ClassCombo9.Value

    End If

'Inserts the name of the class in each combo box into its respective column

    CurrentDb.Execute "INSERT INTO Schedule (A, D, G, J, M, P, S, V, Y) VALUES ('" & A & "', '" & D & "', '" & G & "', '" & J & "', '" & M & "', '" & P & "', '" & S & "', '" & V & "', '" & Y & "')"

'Inserts Sections 1, 2, and 3 into their respective columns to help the user identify which classes the solver is choosing to find the optimal solution.

    CurrentDb.Execute "INSERT INTO Schedule (A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, AA) VALUES ('Sec1', 'Sec2', 'Sec3' , 'Sec1', 'Sec2', 'Sec3' ,'Sec1', 'Sec2', 'Sec3' ,'Sec1', 'Sec2', 'Sec3' ,'Sec1', 'Sec2', 'Sec3', 'Sec1', 'Sec2', 'Sec3', 'Sec1', 'Sec2', 'Sec3' ,'Sec1', 'Sec2', 'Sec3', 'Sec1', 'Sec2', 'Sec3')"

'Creates a blank row, which acts as the solver's "edit" row

    CurrentDb.Execute "INSERT INTO Schedule (A) VALUES ('')"
```

To better explain how the InputPrefVal() sub routine works, there a snippet of coding from appendix C copied below, anything with the single quote in front of it is a comment to better understand the use of each line included here to explain its purpose.

```
'If the class chosen in ComboBoxZ is ES 241, or ES242, or ENGL333, which are all classes with 2 sections of lecture, then store the preference value given to class Z(that combo box is names PrefValZ, more details in appendix B) in two different variables.

    if ClassCombo9.Value = "ES 241" Or ClassCombo9.Value = "ES 242" Or
    ClassCombo9.Value = "ENGL 333" Then

    CC9S1 = PrefVal9

    CC9S2 = PrefVal9

    CC9S3 = ""

'Otherwise store the preference value for class Z in only one variable for use later.

    Else

    CC9S1 = PrefVal9

    CC9S2 = ""

    CC9S3 = ""

    End If

'Then insert those variables into the tables respective columns , e.g. if classcombo 9 is ES 241 and the preference value is 3, then CC9S1, and CC9S2 are both equal to 3, and that number will show up in the section 1 and section 2 column of the ES 241 TriColumns.
```

```
CurrentDb.Execute "INSERT INTO Schedule (A, B, C, D, E, F, G, H, I, J, K, L, M, N, O,
P, Q, R, S, T, U, V, W, X, Y, Z, AA) VALUES ('" & CC1S1 & "', '" & CC1S2 & "', '" &
CC1S3 & "', '" & CC2S1 & "', '" & CC2S2 & "', '" & CC2S3 & "', '" & CC3S1 & "', '" &
CC3S2 & "', '" & CC3S3 & "', '" & CC4S1 & "', '" & CC4S2 & "', '" & CC4S3 & "', '" &
CC5S1 & "', '" & CC5S2 & "', '" & CC5S3 & "', '" & CC6S1 & "', '" & CC6S2 & "', '" &
CC6S3 & "', '" & CC7S1 & "', '" & CC7S2 & "', '" & CC7S3 & "', '" & CC8S1 & "', '" &
CC8S2 & "', '" & CC8S3 & "', '" & CC9S1 & "', '" & CC9S2 & "', '" & CC9S3 & "')"
```

One thing that can be noticed about the input preference value for this code, is that it doesn't include labs. The reason for this is that the labs have to be chosen with lectures, which have their own preference values. What this means, is that if a lab section is given a preference value, that value is added to the value of the lecture, effectively doubling the value of any class. So if a student were to select 10 for their value of a class like 420 which has a lecture and a lab section, the solver would know that selecting IME 420 would not just be worth the 10 points for the lecture, but would be worth another 10 points for the lab value that it has to sign up for when it picks 420, making it worth 20 points, twice what a class like ENGL333 would be worth if a student wanted to take that class at a 10 value as well.

Methodology: MS Access: Subroutines SecConstraint and UnitConstraint

For the Companion PASS program to make a fully constrained table it needs more than just the class name, and preference values. The table needs to also be constrained by the units of each class, and by the sections of each class. A student should not, for example, be able to have the companion program produce a schedule that tells the student to take two lecture sections of IME 405. A student should also not get a schedule from the companion program that requires a student to sign up for more than 16 units, as it is impossible to do so until after every student has gone through their rotation, and most schedules would be obsolete by that point.

For that purpose the Companion Program uses two subroutines :

Secconstraint() - which ensures that a class is only signed up for once

UnitConstraint() – Makes sure 16 or fewer units are selected

Looking at figure 6 the add-ins from SecConstraint are highlighted in Yellow, and the add-ins from UnitConstraint are highlighted in Red.

Figure 6

Schedule					
A	**B**	**C**	**D**	**E**	**F**
ENGL 333			IME 417		
Sec1	Sec2	Sec3	Sec1	Sec2	Sec3
8	8		6		
1	1				
4	4	0	4	0	0

To better explain the way the Section constraint works there's a snippet of the code form appendix C written below with some comments to explain the logic the function has to go through to pass:

'If the class chosen is either ES 241, ES242, or ENGL 333 create a constraint that only allows one of the classes to be selected, and this is done for every combo box.

```
If ClassCombo1.Value = "ES 241" Or ClassCombo1.Value = "ES 242" Or
ClassCombo1.Value = "ENGL 333" Then

CurrentDb.Execute "INSERT INTO Schedule (A, B, AC) VALUES ('1', '1', '1')"

End If
```

To better explain the way the Unit constraint works there's a snippet of the code form appendix C written below with some comments to explain the logic the function has to go through to pass:

'Because the values stored in the user selections, the combo boxes, are string values in Access they need to be examined by their length to see if the 'exist'.

'So in this first if statement the code checks to see that the combo box is not empty and checking to see if it is the same as one of the 3 double lecture classes in the database. If it is, it looks up the unit value for that particular class and applies it to the appropriate column, twice In the case of the 3 double lecture classes, once for each lecture section, and once for each other class.

```
If Len(ClassCombo1 & vbNullString) <> 0 And ClassCombo1.Value = "ES
241" Or ClassCombo1.Value = "ES 242" Or ClassCombo1.Value = "ENGL
333" Then

C1name = ClassCombo1.Value

C1U = DLookup("Units", "ClassList", "Classnum = '" & C1name & "'")

C1U2 = DLookup("Units", "ClassList", "Classnum = '" & C1name & "'")

C1U3 = 0

ElseIf Len(ClassCombo1 & vbNullString) <> 0 Then

C1name = ClassCombo1.Value

C1U = DLookup("Units", "ClassList", "Classnum = '" & C1name & "'")

C1U2 = 0

C1U3 = 0
```

‘if the box is blank, nothing happens, the VBA simply refreshes the page, a harmless action.

Else

Me.Refresh

Methodology: MS Access: Subroutine Lab Constraint

The last two subroutines are the most complicated by far and address the biggest problem when scheduling classes. The first is the issue of making sure that classes which should have a lecture and a lab will have only one lecture and one lab section selected by the Excel solver and that the constraints are made correctly In Access.

When working in Excel solver, the solver can only read equations. This is an issue, because the constraint for a lab requirement means that the class value should be equal to the lab value, i.e if LecSec = 1 then LabSec = 1. But saying that one cell must be equal to a Changing cell in the Excel solver breaks the solver, so after much tinkering and experimentation, the author reached this conclusion.

Cij→ class combo i, section j, where if j = 2,3 then class is a lab section.

C11*PrefValue – C12 – C13 =<0 and cannot be negative.

The next hurdle was making sure that this formula would remain consistent throughout the table, as putting one constraint per row means that the row number in the formula would change. To clarify this, there’s a snippet of code from appendix C printed below with comments indicated by beginning with the single quote symbol(‘) to better explain the logic to the code.

‘This code uses a unique tool called a counter, the counter(dubbed ‘cntr’ below) is designed to keep track which row the lab constraints are in. Because of the way the constraints are formed, namely, the section constraints come before the lab constraints, the cntr starts at a negative 2 and is ticked up by one for every user selection that has 2 sections, or is one of these lab sections. So every time it gets called up the subroutine knows to add one to the cntr and to change the formula Excel will be reading by moving it down one row.

```
If ClassCombo1.Value = "IME 416" And cntr = -2 Then

    CurrentDb.Execute "INSERT INTO Schedule (A, B, C, AB, AC) VALUES ('1', '1',
    '1', '=A3*A5-B3*B5-C3*C5', '0')"

    cntr = 1 + cntr


    ElseIf ClassCombo1.Value = "IME 443" And cntr = -2 Or ClassCombo1.Value
    = "IME 420" And cntr = -2 Or ClassCombo1.Value = "IME 410" And cntr = -2
    Or ClassCombo1.Value = "IME 405" And cntr = -2 Then

    CurrentDb.Execute "INSERT INTO Schedule (A, B, AB, AC) VALUES ('1', '1',
    '=A3*A5-B3*B5', '0')"

    cntr = 1 + cntr


    ElseIf ClassCombo1.Value = "IME 416" And cntr = -1 Then

    CurrentDb.Execute "INSERT INTO Schedule (A, B, C, AB, AC) VALUES ('1', '1',
    '1', '=A3*A6-B3*B6-C3*C6', '0')"

    cntr = 1 + cntr
```

```
ElseIf ClassCombo1.Value = "IME 443" And cntr = -1 Or ClassCombo1.Value = "IME 420" And cntr = -1 Or ClassCombo1.Value = "IME 410" And cntr = -1 Or ClassCombo1.Value = "IME 405" And cntr = -1 Then

CurrentDb.Execute "INSERT INTO Schedule (A, B, AB, AC) VALUES ('1', '1', '=A3*A6-B3*B6', '0')"

cntr = 1 + cntr


ElseIf ClassCombo1.Value = "IME 416" And cntr = 0 Then

CurrentDb.Execute "INSERT INTO Schedule (A, B, C, AB, AC) VALUES ('1', '1', '1', '=A3*A7-B3*B7-C3*C7', '0')"
```

Methodology: MS Access: Subroutine Group Constraint

The PASS Companion program allows for users to also make constraints In the form of class groupings, allowing students to pick groups of classes and ask that only one among the group be chosen, this is ideal if a user has three classes they must have, and are simply looking for a class that can fit in with the time constraints.

To better understand how that works there is a small sample of the code for the group constraint below with some explanations marked by a single quote(‘)

```
‘ The first statement here checks to see if the yes/no box for class combo 1 is group 1 is checked, and if it is stores that value to be later used as a constraint. This step is repeated 8 more times with each of the yes/no boxes for a  user selection and turned into a constrain that must = 1 in order to get 1 class out of it for the user, it stores two values for a class which has two lectures, and one otherwise.
```

```
If (G1C1.Value = -1 And ClassCombo1.Value = "ES 241") Or (G1C1.Value = -1
And ClassCombo1.Value = "ES 242") Or (G1C1.Value = -1 And
ClassCombo1.Value = "ENGL 333") Then

c1 = 1

c12 = 1

ElseIf G1C1.Value = -1 Then

c1 = 1

End If
```

Methodology: MS Access: Subroutine TimeConstraint(Query)

The most important subroutine in the Access portion of the project is the time constraints. The 135 queries explained on page 19 are crucial to this subroutine.

In simplest terms this subroutine has to have a query for its criteria, and then reads that table for every possible entry that could in the table, so it looks for a total of 27 values, the first selection and its first 3 sections, the second selection and its 3 sections, …, and the ninth selection and its 3 sections. If it finds a value the Sub routine will store that value and then when it is done looking at that query it will insert what it has found as a constraint into the table. To make this easier to understand there's an example along with a piece of the code from the subroutine located in appendix C.

If, for example, a student were to look at the conflicts between two classes, IME 416, and ES 241, that student could check the tables listed in appendex D and see that both classes have conflicts on Monday in section 1 from 2:00-4:00pm. Here's the code and how it logics its way through the query to make the constraint:

‘First thing the code does is look to make sure the entries from the user are not empty, and then if they are not empty the value gets stored as a variable, in this case the variable are C1name, and C2name.

```
If Len(ClassCombo1 & vbNullString) <> 0 Then

C1name = ClassCombo1.Value

End If

If Len(ClassCombo2 & vbNullString) <> 0 Then

C2name = ClassCombo2.Value

End If
```

‘below in figure 7 is the results of the query this subroutine is running under in this example, the query “CourseConflictMCombo1Sec1”, which shows all classes that conflict with the first section of the user’s first selection, which in this case is ES241.

CourseConflictMCombo1Sec1										
ClassNum	**Section**	**StartTime**	**EndTime**	**Monday**	**Tuesday**	**Wednesday**	**Thursday**	**Friday**	**ClassType**	**Units**
ES 241	1	12:10:00 PM	2:00:00 PM	-1	0	-1	0	0	Lec	4
IME 416	1	12:10:00 PM	2:00:00 PM	-1	0	0	0	0	Lec	3

Figure 7

‘These classes are on the same day and start at the same time, so the subroutine will find them, and it will find that ES 241 “conflicts” with itself, and that IME 416 conflicts with the entry for classcombo1sec1.

‘Now the subroutine looks for an entry in the table where the field Classnum is the same as the value of the first combo box, and is in section1, in this case it will of course “conflict” because that is the very same class we are looking for things to compare to, so it will essentially conflict with itself. While this seems odd, it is necessary because the constraint must show that ES241 cannot be chosen at the same time as IME 416.

‘If the value is in the table, or in other words isn’t null, meaning it must be there, a variable is changed to 1, indicating there is a class there, acting like a switch.

```
If Not IsNull(DLookup("Classnum", "" & Qn & "", "Classnum = '" & C1name
& "' And Section = '1'")) Then

CC1S1 = 1

End If


If Not IsNull(DLookup("Classnum", "" & Qn & "", "Classnum = '" & C1name
& "' And Section = '2'")) Then

CC1S2 = 1

End If


If Not IsNull(DLookup("Classnum", "" & Qn & "", "Classnum = '" & C1name
& "' And Section = '3'")) Then

CC1S3 = 1

End If


If Not IsNull(DLookup("Classnum", "" & Qn & "", "Classnum = '" & C2name
& "' And Section = '1'")) Then
```

```
CC2S1 = 1

End If


If Not IsNull(DLookup("Classnum", "" & Qn & "", "Classnum = '" & C2name
& "' And Section = '2'")) Then

CC2S2 = 1

End If


If Not IsNull(DLookup("Classnum", "" & Qn & "", "Classnum = '" & C2name
& "' And Section = '3'")) Then

CC2S3 = 1

End If
```

‘ This process of ‘switching on’ is continued through every section for each selection for the query, and then the constraint is inserted into the table, after being check to make sure there is anything to put in. If there was no such check, and there was no conflicts with any of the classes and the first section of the first selection then there would be a blank row, and there would be many more for each query that revealed no conflicts. And with a potential for 135 blank lines, that’s a lot of empty space.

```
If Len(CC1S1 & vbNullString) = 0 And Len(CC1S2 & vbNullString) = 0 And
Len(CC1S3 & vbNullString) = 0 And Len(CC2S1 & vbNullString) = 0 And
Len(CC2S2 & vbNullString) = 0 And Len(CC2S3 & vbNullString) = 0 And
Len(CC3S1 & vbNullString) = 0 And Len(CC3S2 & vbNullString) = 0 And
Len(CC3S3 & vbNullString) = 0 And Len(CC4S1 & vbNullString) = 0 And
Len(CC4S2 & vbNullString) = 0 And Len(CC4S3 & vbNullString) = 0 And
Len(CC5S1 & vbNullString) = 0 And Len(CC5S2 & vbNullString) = 0 And
Len(CC5S3 & vbNullString) = 0 And Len(CC6S1 & vbNullString) = 0 And
```

```
Len(CC6S2 & vbNullString) = 0 And Len(CC6S3 & vbNullString) = 0 And
Len(CC7S1 & vbNullString) = 0 And Len(CC7S2 & vbNullString) = 0 And
Len(CC7S3 & vbNullString) = 0 And Len(CC8S1 & vbNullString) = 0 And
Len(CC8S2 & vbNullString) = 0 And Len(CC8S3 & vbNullString) = 0 And
Len(CC9S1 & vbNullString) = 0 And Len(CC9S2 & vbNullString) = 0 And
Len(CC9S3 & vbNullString) = 0 Then

Me.Refresh

Else

CurrentDb.Execute "INSERT INTO Schedule (A, B, C, D, E, F, G, H, I, J, K, L, M,
N, O, P, Q, R, S, T, U, V, W, X, Y, Z, AA, AC) VALUES ('" & CC1S1 & "', '" &
CC1S2 & "', '" & CC1S3 & "', '" & CC2S1 & "', '" & CC2S2 & "', '" & CC2S3 & "',
'" & CC3S1 & "', '" & CC3S2 & "', '" & CC3S3 & "', '" & CC4S1 & "', '" & CC4S2
& "', '" & CC4S3 & "', '" & CC5S1 & "', '" & CC5S2 & "', '" & CC5S3 & "', '" &
CC6S1 & "', '" & CC6S2 & "', '" & CC6S3 & "', '" & CC7S1 & "', '" & CC7S2 & "',
'" & CC7S3 & "', '" & CC8S1 & "', '" & CC8S2 & "', '" & CC8S3 & "', '" & CC9S1
& "', '" & CC9S2 & "', '" & CC9S3 & "', '1')"

End If
```

This process is repeated many more times than is shown here, 27 times in total, 3 times for each combo box, and then once for each of the 135 queries. .

Methodology: MS Excel Solver

For the Excel portion of the methodology flip to Appendix D to see an example table from the PASS companion program with annotations, and references from this section. The solver has to go through a few steps in order to find the optimization for a user. Firstly the user needs to change all numeric values in the tables to numbers, because they had to be stored as string values in Access to avoid causing errors when the constraints were made or interacting, because numbers in Access can be added or subtracted, but strings cannot.

The next step is to make the product sum box, which will have be located in cell AB4, and have the value “=SUMPRODUCT($A$3:$AA$3,A4:AA4)”, this value will be copied into the entire AB4 column until it reaches the last value in the AC column, with the exception of the lab constraints, which will be easily noticed as they will have formulae already written into the cell. The last two cells in Column AB and AC are highlighted in green to represent them as group constraints, and the lab constraint cells are highlighted in red. Now it is time to make the solver function, the solver function will have a “target cell” which will serve as the cell the solver is trying to maximize, it is highlighted in blue.

The solver then needs to know 3 different constraints, the first is that the changeable cells(the solver will change these cells to different numbers to try to get the highest possible value in the target cells) need to be made binary, because the classes can either be not take, meaning the value is 0, or taken, which means the value is 1, these cells are highlighted in purple

The second constraint will affect all but the two last rows of column AB, making sure that whatever the solver decides the values in the column AB, except for the last two, will be less than or equal to, the value in the cells directly to the left of them, these cells are highlighted in orange, and their constraining cells are highlighted in Teal.

The last four cells, representing the group constraints, must have the values in the AB column be equal to the Values in the AC column.

Finally the solver options must be accessed, and the problem must be assumed non negative and linear.

Chapter 5: Future Senior Project Opportunities &Conclusions

The PASS companion Program, even after 10 weeks of work, is still far from being “complete”, although it had made great strides in that direction. It has room to grow and be more fully explored before being reliably used by the school and that is where future students can chip in and work on this project as their own senior projects. The project as of right now is constrained to the class list for senior level classes in the fall of 2013, using the methodology established already other students can alter the coding to allow for things like ‘class types’ and implement a field in the table that would make it possible for teachers to add any class, or edit any class without fear that I would create complication or difficulties in the project. Along those lines, a senior project would also include making the system easily edited en mass so that faculty can easily and efficiently update the Companion Program for each quarter.

One other issue is that the program is still in the Alpha phase of testing, after working on the first two points, establishing a class type methodology to make a class editing system viable, and then making said class editing system, the same student can run beta testing for the program, getting feedback and statistics on how long a student will use the companion program in order to optimize the program to be more efficient and more user friendly, This will be the key to the program’s success in the long run, being easy to use for someone who would not normally know how to use it. Ideally this project would be carried over quarter after quarter, adding new features, addressing issues in the program, expanding to different classes of industrial engineers, not just seniors, and finally expanding to other majors.

After 10 weeks of working with this project, it’s clear that the PASS companion program has a future for the students here at Cal Poly, so long as it continues to be improved. The author is regretful that he did not have the time to bring the project to a full completed state, but after working with it for so long, it is obvious that the required amount of testing, coding, data basing skill, Excel solver work, and consistent survey and focus group work would require the work of at least 3 senior projects to bring to a reliable state. There’s a saying that the

first step towards change is awareness, This is problem that no senior has approached in the past, and by making this a senior project that is open to more students who want to work in data basing or operations research, there will be a greater awareness of the problem, and as more students chose to work and improve the PASS companion program, it is the authors opinion that the PASS companion program will be used by the entire senior class of the industrial engineering department in the near future.

Bibliography

Abdullah, Salwani, and Hamza Turabieh. “Generating University Course Timetable Using Genetic Algorithms and Local Search.” Convergence and Hybrid Information Technology, 2008. . Busan , 2008. 254 - 260 .

<http://www.ieeexplore.ieee.org.ezproxy.lib.calpoly.edu:2048/stamp/stamp.jsp?tp=&arnumber=4682035&isnumber=4681984>.

Hou Ming; Chen Qi, "Course scheduling system design and implementation based on genetic algorithm,"*Computer Design and Applications (ICCDA), 2010 International Conference on* , vol.3, no., pp.V3-611,V3-614, 25-27 June 2010
doi: 10.1109/ICCDA.2010.5541391

Abdullah, Salwani , Edmund Burke, and Barry McCollum. “A Hybrid EvolutionaryApproach to the University Course Timetabling Problem.” Proceedings of CEC.Singapore : IEEE Congress, 2007. 1764 - 1768 .

<http://www.ieeexplore.ieee.org.ezproxy.lib.calpoly.edu:2048/stamp/stamp.jsp?tp=&arnumber=4424686&isnumber=4424446>.

Arous, Najet, Salah B. Abdallah, and Noureddine Ellouze. “Evolutionary PotentialTimetables Optimization by Means of Genetic and Greedy Algorithms.”Information Intelligence and Systems, 1999. Proceedings. Bethesda, MD : IEEE, 1999. 24 - 31.

<http://www.ieeexplore.ieee.org.ezproxy.lib.calpoly.edu:2048/stamp/stamp.jsp?tp=&arnumber=810220&isnumber=17541>.

Badri, Masood A. et al. “A Multi-Objective Course Scheduling Model: Combining FacultyPreferences for Courses and Times .” Computers Ops Res 25.4 (1998): 303-316.

Oner, A.; Ozcan, S.; Dengi, D., "Optimization of university course scheduling problem with a hybrid artificial bee colony algorithm," *Evolutionary Computation (CEC), 2011 IEEE Congress on* , vol., no., pp.339,346, 5-8 June 2011
doi: 10.1109/CEC.2011.5949638

Boronico, Jess. “Quantitative modeling and technology driven departmentalcourse scheduling.” Omega 28.3 (2000): 327-346 .69

Burke, E. K., and J. P. Newall. “A Multistage Evolutionary Algorithm for the TimetableProblem.” IEEE Transactions on Evolutionary Computation. IEEE, 1999.

<http://www.ieeexplore.ieee.org.ezproxy.lib.calpoly.edu:2048/stamp/stamp.jsp?tp=&arnumber=752921&isnumber=16264>.

Carter, Michael, and Gilbert Laporte. “Recent developments in practical course timetabling.” Practice and Theory of Automated Timetabling II. 1998. 3-19.

<http://dx.doi.org/10.1007/BFb0055878>.

Fang, Sueychyun (Roger). “University Course Scheduling System (UCSS) – A UML Application with Database and Visual Programming.” Journal of Computing

Sciences in Colleges 20.6 (2005): 160-169.

Ghaemi, Sehraneh, Mohammad Taghi Vakili, and Ali Aghagolzadeh. “Using a Genetic Algorithm Optimizer Tool to Solve University Timetable Scheduling Problem.”Signal Processing and Its Applications, 2007. ISSPA 2007. 9th International

Symposium. Sharjah , 2007. 1 - 4.

<http://www.ieeexplore.ieee.org.ezproxy.lib.calpoly.edu:2048/stamp/stamp.jsp?tp=&arnumber=4555397&isnumber=4555273>.

Hertz, Alain, and Vincent Robert. “Constructing a course schedule by solving a series ofassignment type problems.” European Journal of Operational Research 108.3(1998): 585-603.

Hinkin, Timothy, and Gary Thompson. “SchedulExpert: scheduling courses in the CornellUniversity School of Hotel Administration..” Interfaces 32.6 (2002): 45-57.70

Irene, Sheau Fen Ho, Safaai Deris , and Siti Zaiton Mohd Hashim. “A Combination ofPSO and Local Search in University Course Timetabling Problem.” Proceedingsof the 2009 International Conference on Computer Engineering and Technology. IEEE Computer Society, 2009. 492-495 .

<http://www.ieeexplore.ieee.org.ezproxy.lib.calpoly.edu:2048/stamp/stamp.jsp?tp=&arnumber=4769651&isnumber=4769538>.

Kanoh, Hitoshi, and Yuusuke Sakamoto. “Interactive Timetabling System UsingKnowledge-Based Genetic Algorithms.” Systems, Man and Cybernetics, 2004 IEEE International Conference. 2004. 5852- 5857.

<http://www.ieeexplore.ieee.org.ezproxy.lib.calpoly.edu:2048/stamp/stamp.jsp?tp=&arnumber=1401129&isnumber=30425>.

McClure, Richard H., and Charles E. Wells. “A Mathematical Programming Model forFaculty Course Assignments.” Decision Sciences 15.3 (1984): 409-420.

Ojha, Prakash, and Abigail Walker. “A Comparison of Course Scheduling Methods.”REU2000 2000. <http://www.cs.xu.edu/~lewandow/reu2000/paper/>.

“Orologio Class Timetabling System.” Antinoos - Orologio - Class Timetabling Software.

<http://www.antinoos.gr/en/orologio.htm>.

Parthiban, P et al. “Preferences Based Decision-making Model (PDM) for Faculty Course Assignment Problem.” Engineering Management Conference, 2004. Proceedings. 2004 IEEE International . 2004. 1338 - 1341.

<http://www.ieeexplore.ieee.org.ezproxy.lib.calpoly.edu:2048/stamp/stamp.jsp?arnumber=1408912&isnumber=30511>.

Pesenti, Matthew . “Decision Support System for University Course Scheduling.” 2002.

http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1339&context=theses

“ScheduleWhiz® Academic .” Thoughtimus (coourse scheduling software, timetabling

software). <http://www.thoughtimus.com/index_postsecondary.html>.

Thompson, Gary M. “Improving conferences through session scheduling.” The Cornell Hotel and Restaurant Administration Quarterly 43.3 (2002): 71-76.

Thompson, Gary M. “Using information on unconstrained student demand to improveuniversity course schedules.” Journal of Operations Management 23.2 (2005):197-208.

Wang, Yao-Te et al. “On the Application of Data Mining Technique andGenetic Algorithm to an Automatic Course Scheduling System.” Cybernetics andIntelligent Systems, 2008 IEEE Conference. Chengdu, 2008. 400 - 405.

<http://ieeexplore.ieee.org.ezproxy.lib.calpoly.edu:2048/stamp/stamp.jsp?arnumber=4670852&isnumber=4670728>.

Wang, Yen-Zen. “An Application of Genetic Algorithm Methods for Teacher AssignmentProblems.” Expert Systems with Applications 22.4 (2002): 295-302.

Wang, Yen-Zen. “Using genetic algorithm methods to solve course schedulingproblems.” Expert Systems with Applications. 2003. 39-50.

<http://www.sciencedirect.com.ezproxy.lib.calpoly.edu:2048/science?_ob=MImg&_imagekey=B6V03-47YXTVK-1
25&_cdi=5635&_user=521828&_orig=search&_coverDate=07%2F31%2F2003&72_sk=999749998&view=c&wchp=dGLbVtbzSkzV&md5=77f82fa754ee72af21e69739fefd0aba&ie=/sdarticle.pdf>.

“Ninja Courses” © 2013 William Li and Alex Sydell (Free Course scheduling website for select UCs)

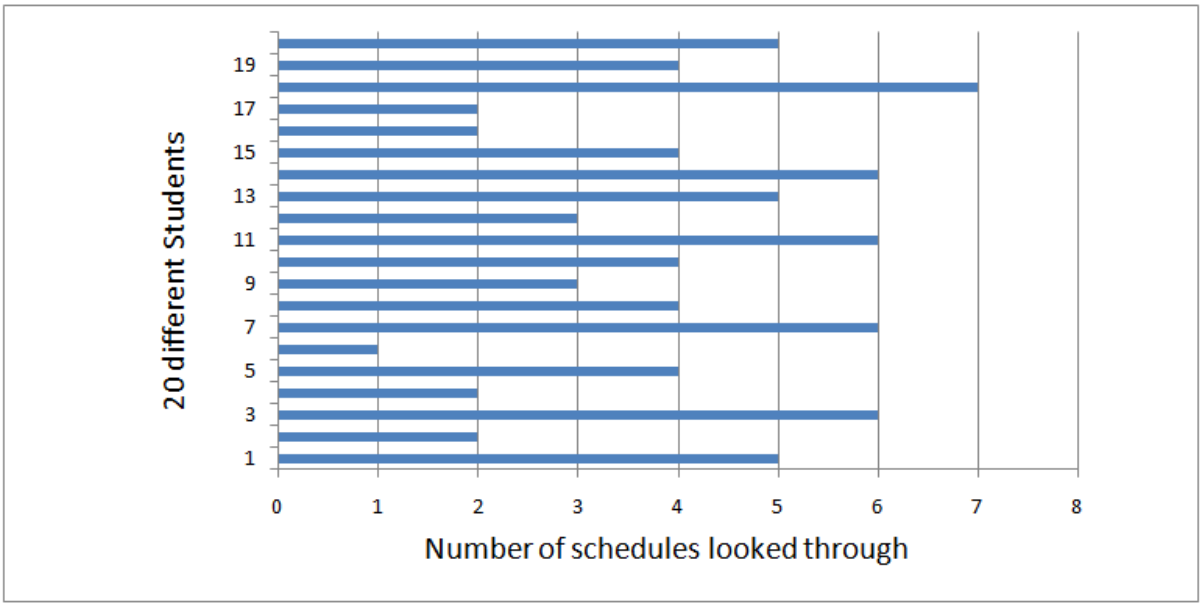
<http://www.NinjaCourses.com/>

Aldasht, M.; Alsaheb, M.; Adi, S.; Qopita, M.A., "University Course Scheduling Using Evolutionary Algorithms," *Computing in the Global Information Technology, 2009. ICCGI '09. Fourth International Multi-Conference on* , vol., no., pp.47,51, 23-29 Aug. 2009
doi: 10.1109/ICCGI.2009.15

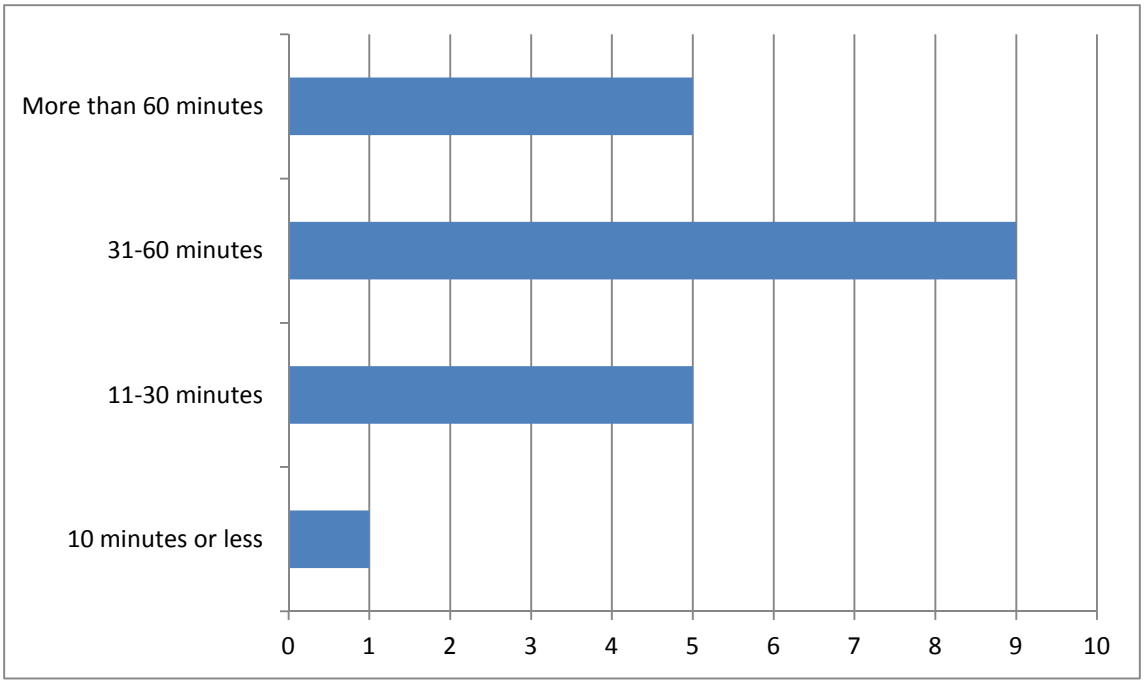
Appendices

Appendix A: Survey Results

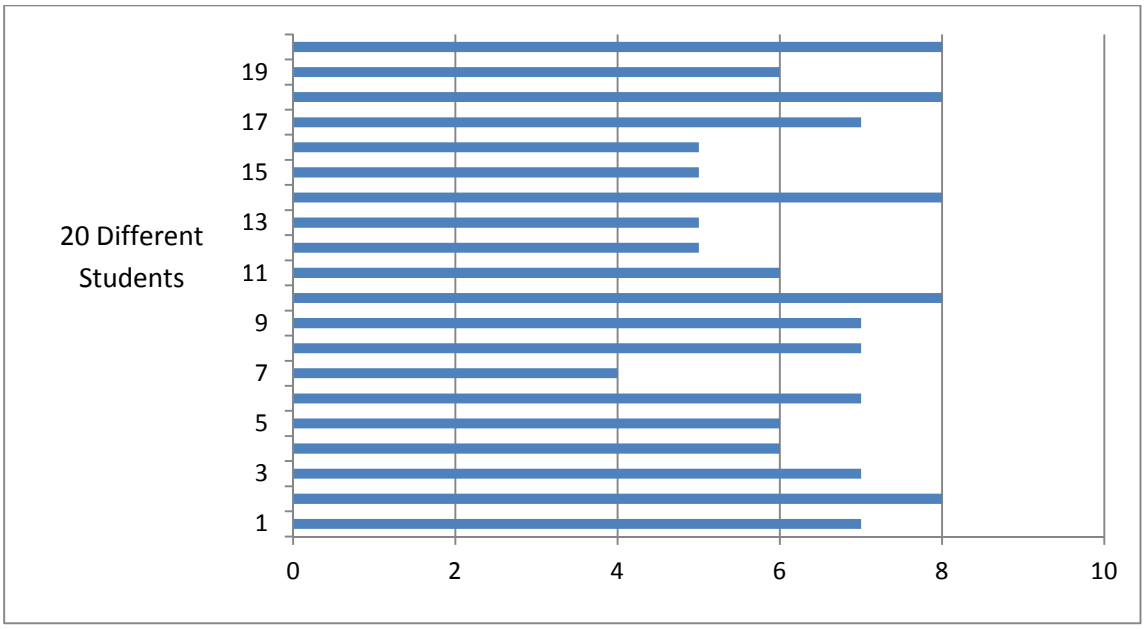
How many schedules do you look through when making a schedule for class?



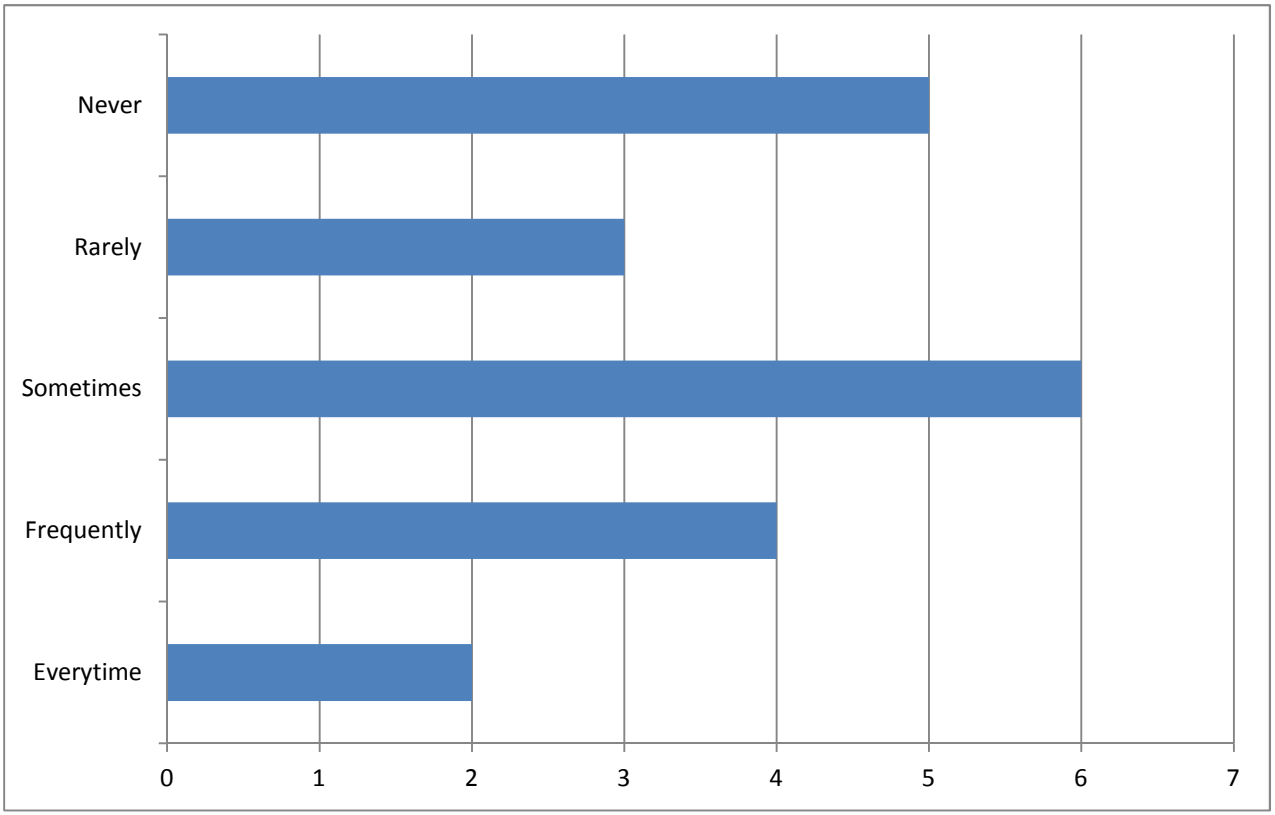
How long do you spend finding the right schedule?



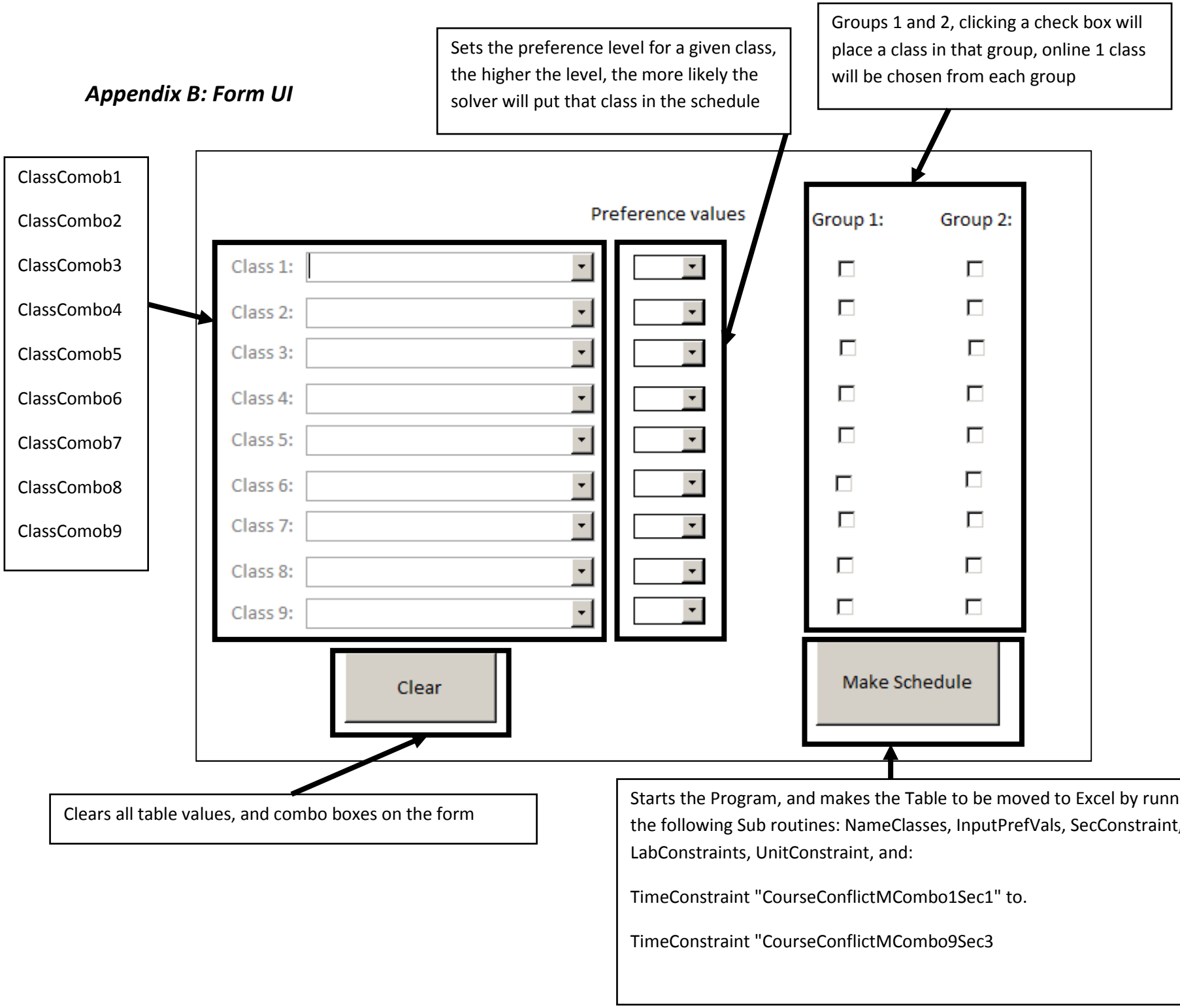
How effective would you say PASS is on a scale from 1 – 10?



Given an Access driven tool to complement PASS, how often do you think you would use it?



Appendix B: Form UI



Appendix C: Subroutines

Sub timeconstraint

```
Sub TimeConstraint(Qn As String)

Dim CC1S1 As String

Dim CC1S2 As String

Dim CC1S3 As String

‘repeats to CC2S1 – CC3S3, meaning Class Combo i section j where i= 1,2,3,4,5,6,7,8,9 and j =1,2,3


If Len(ClassCombo1 & vbNullString) <> 0 Then

C1name = ClassCombo1.Value

End If

If Not IsNull(DLookup("Classnum", "" & Qn & "", "Classnum = '" & C1name & "' And Section = '1'")) Then

CC1S1 = 1

End If

‘Repeats for CclassComob1 to ClassCombo9 and stores their name, if they aren’t null, into a string variable for each
‘section of the ComboBox being checked, for example CC1S1 will get the name of the class For Class Combo 1 if it’s
‘not null, and if that Class Has multiple sections it will be stored in CC1S2 or CC1S3 if it has 2 or 3 sections
‘accordingly.


‘checks to see if all of the combo boxes are null, and does nothing if they are

‘inputs the 1’s into the table to represent time constraints, where time constraints exist.

End Sub
```

Sub UnitConstraint

```
Sub UnitConstraint()

Dim C1U As String

Dim C1U2 As String

Dim C1U3 As String

Dim C1name As String
```

‘String variables to store the name of a class for each of its given sections, repeats from C1U – C9U3, so for C9U3, it means: Class 9 units for section 3.

```
If Len(ClassCombo1 & vbNullString) <> 0 And ClassCombo1.Value = "ES 241" Or ClassCombo1.Value = "ES 242" Or

ClassCombo1.Value = "ENGL 333" Then

C1name = ClassCombo1.Value

C1U = DLookup("Units", "ClassList", "Classnum = '" & C1name & "'")

C1U2 = DLookup("Units", "ClassList", "Classnum = '" & C1name & "'")

C1U3 = 0

ElseIf Len(ClassCombo1 & vbNullString) <> 0 Then

C1name = ClassCombo1.Value

C1U = DLookup("Units", "ClassList", "Classnum = '" & C1name & "'")

C1U2 = 0

C1U3 = 0

Else

Me.Refresh

End If

End Sub
```

‘First, checks to see if the combo box is not null, and a class with multiple lecture sections, then checks to see if that class combo is not null, and not a class with multiple sections, then stores the unit values of that class for two sections or 1 section, depending on which check it made first.

‘Inserts all of the Unit values for the classes found in the checks made above.

Name Classes

```
Sub NameClasses()

Dim A As String

 ‘creates 9 unique string variables

If Not IsNull(ClassCombo1.Value) Then

A = ClassCombo1.Value

End If

‘Stores the names of each Class Combo, given that they are not null

‘insert the name and the words ‘sec1, sec2, sec3, into a table.

End Sub
```

Sub LabConstraints

```
Sub LabConstraints()

Dim cntr As Integer

cntr = 0

cnt = 0

‘a integer variable that tells the program where to place lab constraint formulas based on previous lab checks, and multiple lecture section classes.

If ClassCombo1.Value = "ES 241" Or ClassCombo1.Value = "ES 242" Or ClassCombo1.Value = "ENGL 333" Then

   cnt = cnt + 1

End If

‘Based on the number of multiple lecture classes that are in the class selections the cnt triggers the cntr, and the cntr triggers how the lab constraints are placed in the excel sheet.


If cnt = 0 Then

 cntr = -2

End If
```

‘Based on the cntr and the class in a given combo box, the code places the lab constrains in very specific boxes to make the excel solver function work properly.

```
If ClassCombo1.Value = "IME 416" And cntr = -2 Then

CurrentDb.Execute "INSERT INTO Schedule (A, B, C, AB, AC) VALUES ('1', '1', '1', '=A3*A5-B3*B5-C3*C5', '0')"

cntr = 1 + cntr


ElseIf ClassCombo1.Value = "IME 443" And cntr = -2 Or ClassCombo1.Value = "IME 420" And cntr = -2 Or
ClassCombo1.Value = "IME 410" And cntr = -2 Or ClassCombo1.Value = "IME 405" And cntr = -2 Then

CurrentDb.Execute "INSERT INTO Schedule (A, B, AB, AC) VALUES ('1', '1', '=A3*A5-B3*B5', '0')"

cntr = 1 + cntr


End If


End Sub
```

Sub InputPrefVal

```
Sub InputPrefVals()


If ClassCombo1.Value = "ES 241" Or ClassCombo1.Value = "ES 242" Or ClassCombo1.Value = "ENGL 333" Then
   CC1S1 = PrefVal1
   CC1S2 = PrefVal1
   CC1S3 = ""
Else
   CC1S1 = PrefVal1
   CC1S2 = ""
   CC1S3 = ""
End If
‘Takes the preference values chosen by the user for a given class and stores them into a string variable,
‘inserts the preference values into the table.
End Sub
```

Appendix D: Example Excel sheet

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1	ES 241			IME 416			ART 112			ART 111			IME 481			IME			IME 410			IME 417			IME				
2	Sec1	Sec2	Sec3	Sec1	Sec2	Sec3	Sec1	Sec2	Sec3	Sec1	Sec2	Sec3	Sec1	Sec2	Sec3	Sec1	Sec2	Sec3	Sec1	Sec2	Sec3	Sec1	Sec2	Sec3	Sec1	Sec2	Sec3		
3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	0	1	0	0	1	0	0		
4	8	8		6			6			2			7			7			2			3			5			24	
5	1	1																										0	1
6				1	1	1																						0	0
7																1	1											0	0
8																			1	1								0	0
9	4	4	0	3	0	0	4	0	0	4	0	0	2	0	0	4	0	0	4	0	0	4	0	0	1	0	0	15	16
10	1			1																1								1	1
11		1			1															1								1	1
12	1			1																1								1	1
13		1			1																							0	1
14							1									1												1	1
15							1									1												1	1
16					1																							0	1
17	1	1		1																1								1	1
18						1													1									1	1
19											1																	0	1
20						1													1									1	1
21																						1						1	1
22	1																											0	1
23		1																										0	1
24							1									1												1	1
25													1															1	1
26							1									1												1	1
27																									1			1	1
28		1																										0	1
29											1																	0	1
30																			1									1	1
31																						1						1	1
32											1																	0	1
33		1																										0	1
34																	1											1	1
35											1																	0	1
36																						1						1	1
37																			1									1	1