Stanford Class Picker

Sébastien Robaszkiewicz
Stanford University
robinio@stanford.edu

James Whitbeck
Stanford University
jamesbw@stanford.edu

ABSTRACT

In this paper we describe Class Picker, a web-based application that helps Stanford Computer Science Master's students fill in their program sheets and schedule their classes. Class Picker brings information from many sources into a single application. We demonstrate how interactivity, data display and algorithmic schedule building can drastically reduce the effort needed to complete a program sheet that conforms to the CS department's requirements.

INTRODUCTION

The problem we are solving is as follows: in order to graduate with a Master's in Computer Science, students have to fulfill a whole set of requirements, spelled out in detail in the Stanford Bulletin [1]. The main requirements consist of:

- 5 foundation courses, which can be waived if similar coursework has already been completed.
- 27 units from a chosen specialization. There are 10 such specializations.
- 3 breadth courses from a list specific to the chosen specialization.

There are variations of these requirements, such as picking dual specializations, but the bottom line is that in addition to the foundation courses, at least 36 units have to come from specific lists. Each specialization is itself subdivided into separate requirements. Since a Master's degree corresponds to 45 units, careful planning is needed once a specialization is chosen, if one wants to fulfill all requirements in close to 45 units.

We have spent tens of hours ourselves figuring out the right sequence of courses we will take. We also conducted a need-finding survey with classmates and found that they all used very different approaches: some opened dozens of tabs in browsers to get all the information they needed; others built Excel spreadsheets with basic algorithms to build their schedules; others still tried to do everything with pen and paper. All had in common that they spent a lot of time in the process.

Here are some of the difficulties faced when building a schedule:

- Some courses are offered several times during the year, others only once a year.
- Some courses are not offered every year, but information is scarce on when they will be offered.
- Often, two required courses are offered at the same time.
- Courses can often be picked for a variable number of units. Students generally want to maximize the number of units, while remaining under a certain threshold (10 per term) for tuition purposes.
- Students often have their own constraints or preferences, such as not wanting courses on Fridays, needing to free up 2 days a week to do part-time work off campus, or graduating in a given number of terms.

As we will see in the Related Work sections, the current information is in many places. Moreover, it is hard to figure out whether a particular sequence of courses is actually possible given schedule conflicts and personal constraints. Therefore, we define 3 main goals that our application must fulfill:

a) It must provide information on the specializations and their requirements.

b) It must provide information on each course, when it is offered, with whom, for how many units.

c) It must allow experimentation: incrementally picking courses and checking for conflicts, integrating personal constraints, viewing possible schedules.

RELATED WORK

In this section, we will describe the existing resources and tools available to students to find information and build their course schedules.

CS department website

The Computer Science department maintains a list of computer science courses offered during the current academic year [2]. It provides days and times, as well the names of the instructors for each course. It can therefore be seen as verifying goal b).

However, this resource has many shortcomings. Firstly, each page displays only one term, hence four browser tabs are needed to display the four terms in the academic year. This makes it difficult to figure out if a course is offered in several terms. Secondly, no unit information is provided. Thirdly, there is no information on what requirements each course is fulfilling. Finally, some courses for the Master's degree are taught in other departments (ME, MS&E, COMM, EE, PSYCH, BIO, GENE, ENGR), so students must look elsewhere for information on those courses.

Stanford Bulletin: explorecourses
The Stanford bulletin has a website called explorecourses [3] that allows searching for particular courses. One can search for all CS courses too. It gives information on a course by course basis, instead of a term by term basis as with the CS department website. Therefore, it can list all the offerings for a particular course, which is quite useful. It also informs on the units and instructors for each course. So it does quite well with regard to goal b).

However, it has drawbacks too: there are about 200 CS courses listed in the system, however the web interface only displays 10 at a time. So to list them all, one would need 20 browser tabs. Moreover, it does not link to information about specializations (goal a)) or provide the ability to pick a course and experiment (goal c)).

Stanford Bulletin: exploredegrees
The Stanford bulletin also has a website called exploredegrees [1] which contains all the requirements for all degrees delivered by the University. In particular, it contains all the requirements for each specialization for CS Master's degrees. It fulfills goal a), and is the go-to reference for such information. However it is lacking on course information (goal a)) and no picking is possible (goal c)).

Courserank
Courserank [4] is a service not directly affiliated with Stanford but provides valuable course information such as reviews, ratings and Q&As. It provides basic course information such as times and instructors. However, this information is not always as accurate as the Stanford resources are. Nonetheless, it does a decent job on goal a).

The more interesting feature of Courserank is that it allows students to pick courses and build schedules. This is very helpful for planning, because one can see what courses conflict and what days are free for example. Students can also plan all the way through the Spring term. Courserank is therefore a good resource for goal c).

It does not, however, give any information on specializations and requirements (goal a)).

Axess
Axess is Stanford's official "Human Resources" website [5]. For students, it is the place where they register for courses, pay tuition, check grades, request transcripts, update personal information etc. Axess has recently deployed as system called SimpleEnroll. It is a graphical interface for enrolling in classes for the current term. Through a search feature, one can figure out the times, instructors and units for a course. A 'Plan' feature lets students place the course on a schedule. So it fulfills goal b) and c) to some extent.

However, it is not possible to list all courses that are available. Moreover, it is only available for the upcoming term, meaning that longer term planning is not possible, and no information about requirements and specializations is given (goal a)).

Summary
The above paragraphs should have made clear that, while the information is available, it is difficult to access it all at the same time, and impossible to do so in the same place.

	Requirement info	**Course info**	**Interaction and Feedback**
CS Depart ment	✗	✓	✗
explore courses	✗	✓	✗
explore degrees	✓	✗	✗
Course rank	✗	✓	✓
Axess	✗	✓	✓

Table 1. Matrix of goals met by available resources

The following matrix summarizes the available resources:

CONTRIBUTIONS
Class Picker provides a simple interface in which to select courses. It has information on specializations, courses and allows for building schedules and checking for conflicts, thereby meeting goals a), b) and c).

In particular, Class Picker has the following contributions:

1) Allowing the student to focus on the essential: picking courses to satisfy requirements. The application automatically calculates all valid schedule permutations, so the user doesn't have to mentally keep track of scenarios. The application also automatically allocates units for each valid schedule and reports on the maximum units obtained.

2) A step by step process: the users first select their specializations, then select courses. In the end, they can visualize the resulting schedules proposed by the application. Additionally, in the course

picking phase, the users can step through each requirement, in whatever order.

3) Varying level of detail: a lot of information is present in the app. At a high level, we offer overviews of what requirements are satisfied so far. The users can then drill into particular courses and pull up all available information. The proposed schedules also allow zooming.

4) Immediate feedback: every time a selection is made, the application recalculates the progress on each requirement. It also signals courses that are no longer available because of conflicts.

5) An overview tab that lists all the picked courses, grouped by requirements. This is almost exactly the program sheet that needs to be filled in for the department.

6) User constraint integration: users can pick the terms they will study in, set maximum number of units and pick the days they want courses on.

The application is also useful beyond the first term because it allows users to mark which courses have already been taken (and for how many units). Moreover, users can mark courses as waived: those will still be counted toward requirements when possible.

Figure 1 gives an overview of the contributions.

METHODS

In this section, we will explore the main visual and algorithmic methods.

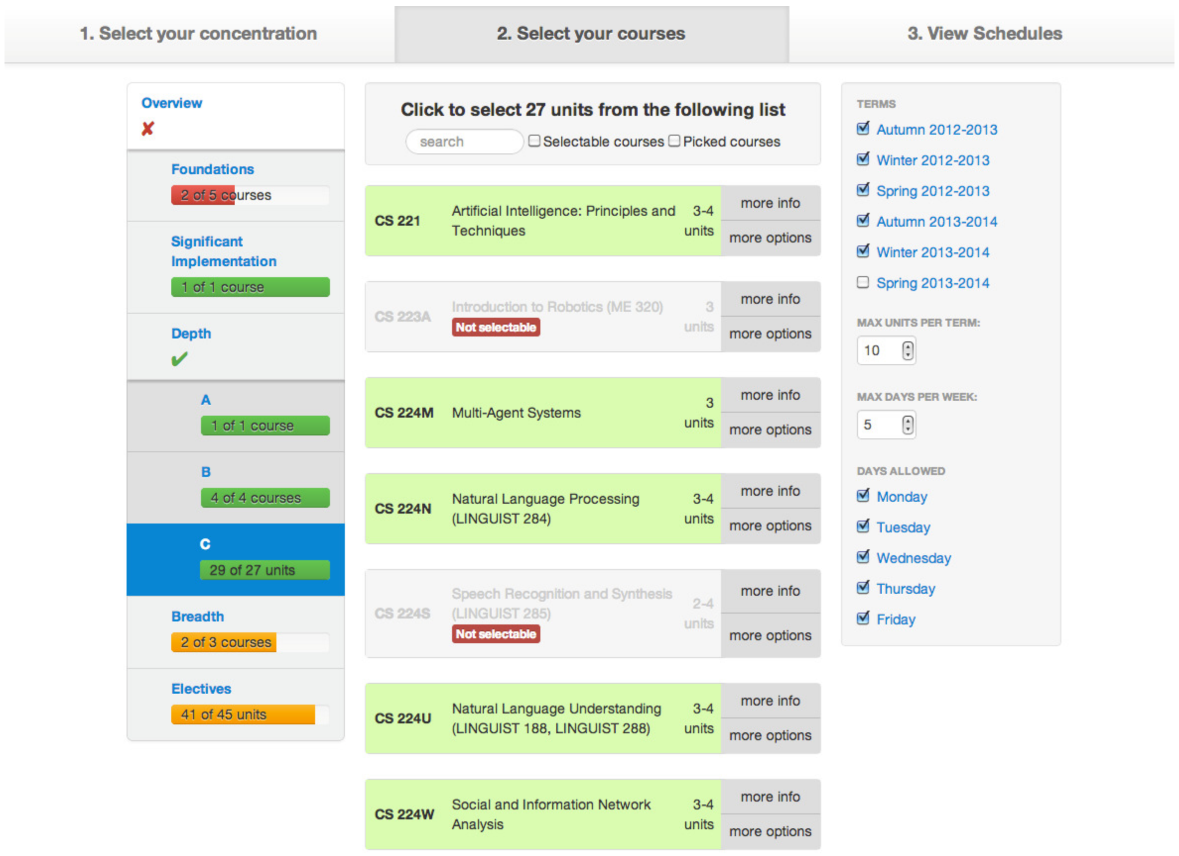
Levels of detail

One of the main challenges of the application was to condense all the information from all the sources, while making it easily accessible to the user. To give an idea of the quantity of information, the HTML file from explorecourses[3] that was parsed for course information was over 400,000 lines long.

Therefore we tried to organize this information hierarchically. One of our inspirations was the Degree of Interest trees [6] where navigation through the tree surfaced deeper nodes, while others we being rolled-up and collapsed. In Class Picker, during the course-picking phase, the left column shows all the requirements that need to be fulfilled. By clicking on the "Overview" tab, the user can see all the selected classes and where they fit in with the requirements.

Each requirement tab contains a progress bar and progress text that shows how much of the requirement is fulfilled. Red to green coloring of the progress bar provides redundant encoding of the progress.

The user can drill down into each requirement, to see the list of courses that can be picked. For each course, a "more info" tab brings up a screen with the course description, times, instructors, units and grading basis.



Foundations: 2 of 5 courses
Significant Implementation: 1 of 1 course
Depth ✓
A: 1 of 1 course
B: 4 of 4 courses
C: 29 of 27 units
Breadth: 2 of 3 courses
Electives: 41 of 45 units

CS 221	Artificial Intelligence: Principles and Techniques	3-4 units
CS 223A	Introduction to Robotics (ME 320) Not selectable	3 units
CS 224M	Multi-Agent Systems	3 units
CS 224N	Natural Language Processing (LINGUIST 284)	3-4 units
CS 224S	Speech Recognition and Synthesis (LINGUIST 285) Not selectable	2-4 units
CS 224U	Natural Language Understanding (LINGUIST 188, LINGUIST 288)	3-4 units
CS 224W	Social and Information Network Analysis	3-4 units

TERMS
☑ Autumn 2012-2013
☑ Winter 2012-2013
☑ Spring 2012-2013
☑ Autumn 2013-2014
☑ Winter 2013-2014
☐ Spring 2013-2014

MAX UNITS PER TERM: 10

MAX DAYS PER WEEK: 5

DAYS ALLOWED
☑ Monday
☑ Tuesday
☑ Wednesday
☑ Thursday
☑ Friday

Figure 1: The course picking view. The header shows the three steps: picking a concentration, selecting classes, viewing resulting schedules. The left hand column shows the progress on each requirement. The right hand bar shows the selected terms and constraints. Course selection is done by clicking on each course. Filtering options are at the top.

The user can also adjust the amount of information displayed with the filter bar at the top. A search field filters the courses according to the entered text, while two checkboxes allow filtering on only available courses or courses already selected.

In the schedule view (third tab), we display up to 9 valid schedule proposals that meet all the constraints. The miniature views give a preview for each term. These miniatures are vertically aligned so that days match. By clicking on any miniature, a pop-up shows a more detailed view. The color palette was picked to maximize differences in hue between courses (a categorical variable). We initially used a palette from ColorBrewer [7] but had to swap out the yellow because it contrasted poorly with white text.

Figure 2 shows the zoomed version of a schedule for a particular term.

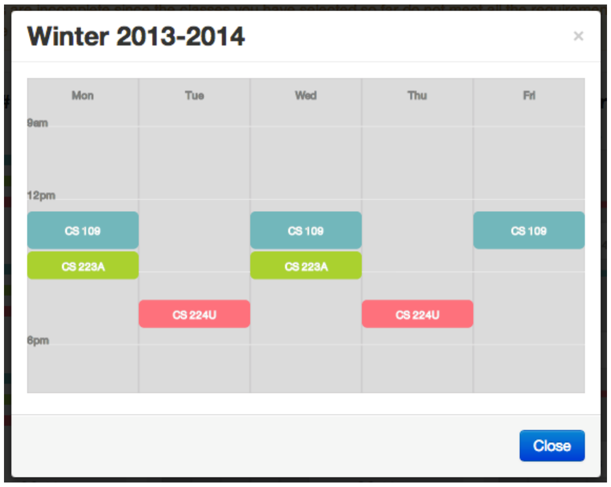


Figure 2: A schedule popup, shown after clicking on a miniature in the schedules view

Immediate feedback

An interactive tool is only really useful if it provides fast feedback. This is what Class Picker attempts to do. For instance, when a course is selected, all the requirements are checked and progress on each is updated.

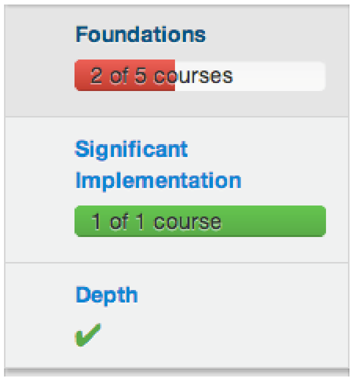


Figure 3: These progress bars are updated whenever a change is made (picking or dropping course, changing terms or constraints)

Another main concern when building a schedule without Class Picker is the "what if?" question: if I pick a certain course, is it going to make other courses not available because of time conflicts, or because courses can't all fit in the same term?

A main goal in Class Picker is to make these dependencies more obvious. The application only lets users pick courses that will actually fit in a valid schedule. That means that if two courses are offered at the same time, and one is picked, the other will be marked as not available. For each disabled course, a tooltip will give feedback on why the course can't be selected. Sometimes, the course is not offered at all, or not offered on the days and terms selected by the user. Other times, there are schedule conflicts with other courses. In that case, the tooltip will suggest courses to remove to make the course available once again (Figure 3).

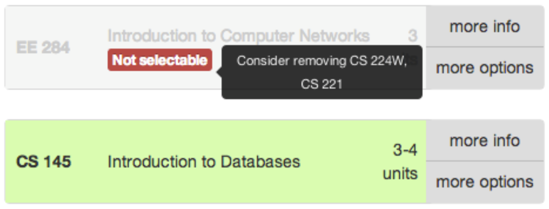

EE 284
Introduction to Computer Networks
3
Not selectable
Consider removing CS 224W, CS 221
more info
more options
CS 145
Introduction to Databases
3-4 units
more info
more options

Figure 3: EE 284 cannot be picked, because it conflicts with CS 224W and CS 221. Removing either of these courses will likely allow picking EE 284.

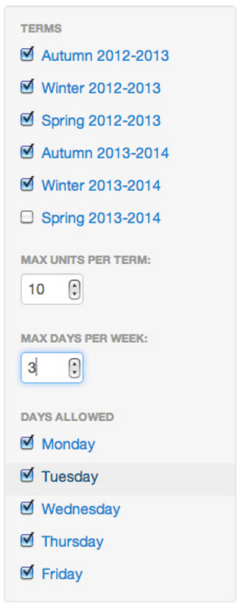
The availability of every course is checked every time a change is made: picking a new class or changing constraints or terms. If ever the user makes the constraints tighter – for instance decides to not have classes on Fridays – then the application will remove whatever courses can no longer be fit into valid schedules. Again, feedback in the form of a warning is provided to the user if this happens.

Integration of constraints

A feature that is not present in any of the current tools is an integration of user constraints. The most common constraint is that student in Computer Science try to stay below 10 units per term. This corresponds to about 3 courses, which is already a significant load in the CS department. It is also a threshold in tuition. Above 10 units, tuition is 50% higher.

Another constraint is what terms the student wants to study in. Stanford offers courses in the summer but few students actually attend, as many are doing internships then. Furthermore, a student may want to graduate in fewer quarters than the standard 5 or 6. Class Picker allows the student to specify exactly what terms to study in.

Some students also have constraints or preferences on what days to study on. We provide both the ability to restrict course to certain days – for example only Monday through Thursday – or to limit the number of days with courses per term. This case comes up when a student is trying to work part-time off-campus, and needs to free up two days a week (Figure 4).


TERMS
- ☑ Autumn 2012-2013
- ☑ Winter 2012-2013
- ☑ Spring 2012-2013
- ☑ Autumn 2013-2014
- ☑ Winter 2013-2014
- ☐ Spring 2013-2014

MAX UNITS PER TERM: 10

MAX DAYS PER WEEK: 3

DAYS ALLOWED
- ☑ Monday
- ☑ Tuesday
- ☑ Wednesday
- ☑ Thursday
- ☑ Friday

Figure 4: The user can constrain what terms to schedule courses in, what days, how many days, how many units for each term. Here, 5 terms, 3 days max per week, 10 units per term.

Algorithms

Under the hood, the application is generating all possible schedule permutations with the courses selected by the user. This has the advantage for the users that they don't have to worry about whether a possible combination of courses can be taken: the application determines that for them. Similarly, the application allocates units to each course in a way that optimizes for requirements, while meeting constraints.

The main idea is that we incrementally build a list of possible schedule permutations. Every time a course is added, for each of its offerings (for instance: Autumn and Winter) we generate a new schedule and add it to our list.

However, this approach can lead to an exponential increase in the number of possible schedules, especially if the constraints are lax. For example, the 5 foundation courses are offered almost every term. If we allow courses in 6 terms, any day of the week, we quickly have more than 7500 permutations, out of the gate.

As the user picks more courses, the schedules fill up, so more conflicts appear, diminishing the number of possible schedules. Tighter constraints (fewer terms, fewer days) also reduce the number of schedules. Nonetheless, with few constraints, it is possible to hit 1,000,000 possible schedules.

Moreover, this list is traversed quite often:

- When a course is picked, for each schedule and each offering of the course, we generate a new schedule.
- Upon any change, we check all non-picked courses to see if they can still be added. This involves trying to fit each course into the current list of schedules. If a course cannot be picked, we must test it against the whole list of schedules.
- Upon any change, we update all the requirements. For course requirements – such as "Pick 3 of the following list" – progress is easy to assess: counting picked courses is enough. It is trickier for unit requirements ("Pick 27 units from this list"): since we want the maximum unit count, we need to calculate unit allocations for each schedule in the list.

This is all tractable when there are on the order of 1000 schedules. At 500,000, it is not. The solution we went for was to do sampling:

- Can a course be picked? We check 1000 randomly sampled schedules.
- How many units are fulfilled? We take the maximum over 1000 random schedules.
- Picking a course: if there are already more than 1000 valid schedules, randomly select 1000 and discard the others. Here, the sampling is done without replacement.

The sampling approach works well because we are only interested in proposing a small set of valid schedules to the user. If there are too many such options, we can just cut back. The application is most useful when the constraints are tight: if the user wants to graduate in 3 quarters, or never have more than 3 days of courses per week. In those cases, sampling is never necessary.

We can do a quick calculation to show that sampling is a valid approach, meaning that we won't lose all the valid schedules by sampling. Let's consider the case where two courses, A and B, conflict in a time slot. Let's imagine that course A has another term in which it is offered and in which there is no conflict. So if I have already picked A, half of my schedules will conflict with B, while the other half won't. If I have 1 million schedules and I sample 1000, the chance that I won't pick any that can fit B is 0.5 to the power of 1000, a very small number.

RESULTS

We performed some user testing, such as doing heuristic evaluations with 3 volunteers, which led to some improvements to the interface. While we do not have timing statistics on how long it took the users to build schedules, they all acknowledged the application was extremely helpful. To illustrate the possible results, we show below some use cases.

Graduating in 3 quarters

While Computer Science students rarely go above 10 units per quarter because of the work load, it is possible to complete the 45 required units for the Master's degree in just three terms. Nonetheless, it makes scheduling a lot more difficult because time conflicts cannot be solved by taking one course one year and the other course the next year.

We tried completing the Artificial Intelligence track. We set the maximum number of units per term to 18, which is another tuition threshold. We allowed all days of the week.

We proceed by filling in the requirements one at a time. Courses that can no longer be picked are marked, so it is easy to only pick valid courses.

Within 30 seconds, we have picked a combination of courses that satisfy all the requirements. Figure 5 shows three possible schedules.

Graduating in 5 quarters, 3 days a week

In this example, we decide to meet the requirements for the Computer and Network security track. Our scenario assumes the user needs to free up two days a week for some other activity, such as working at a start-up. Moreover, we want to graduate in fewer than 6 terms.

Therefore we select five terms (Autumn 2012-2013 through Winter 2013-2014) and set the maximum days per week to 3.

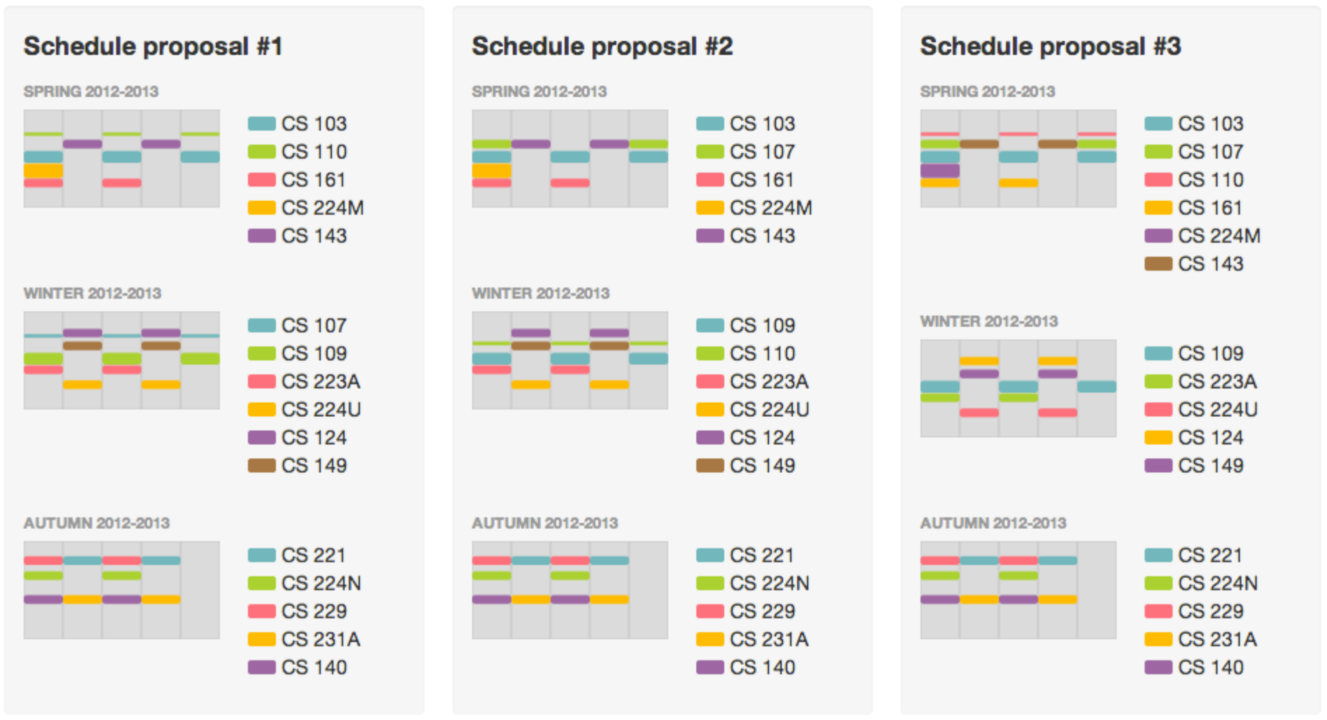


Figure 5: 3 proposed schedules for completing the Artificial Intelligence track in 3 quarters, with a maximum of 18 units per quarter

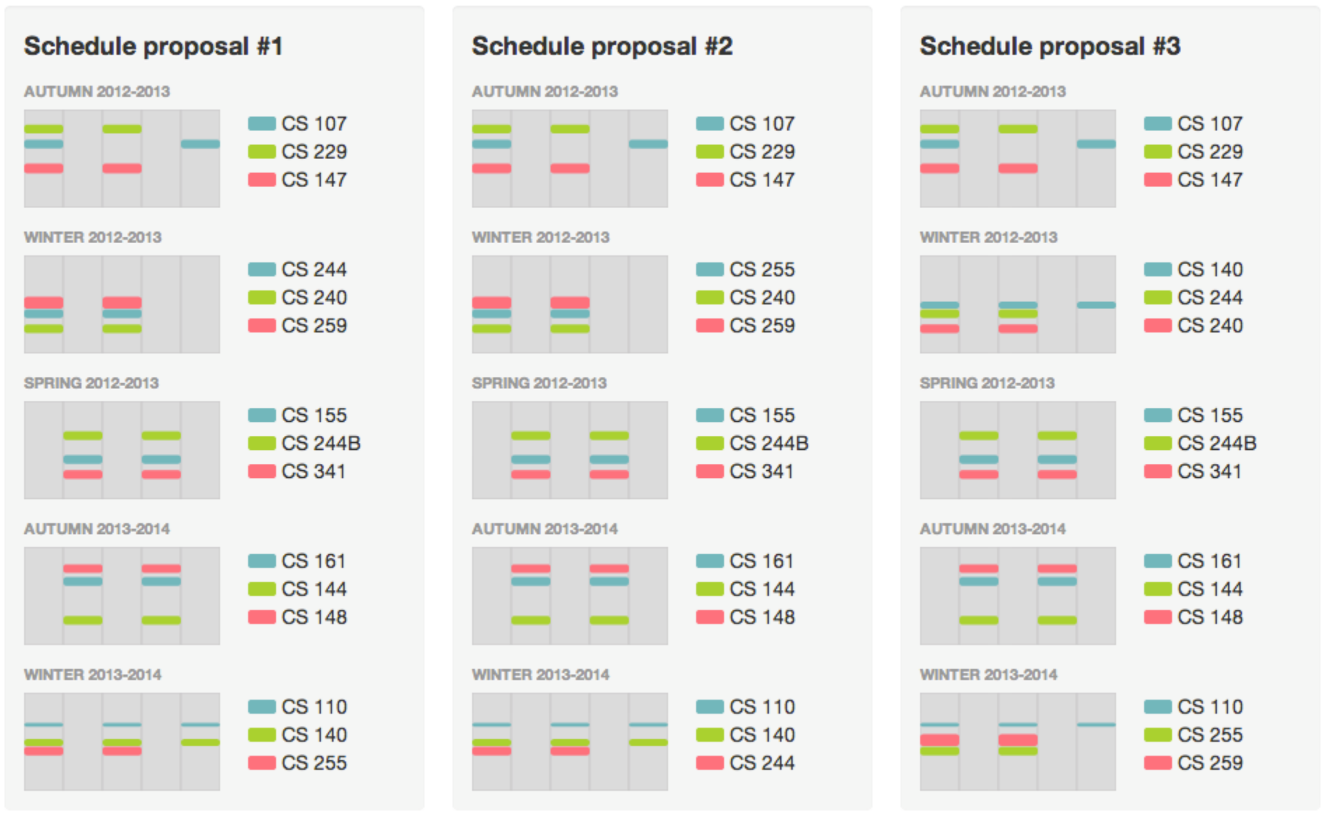


Figure 6: 3 proposed schedules for completing the Computer and Network Security track in 5 quarters, with only 3 days of courses a week and no more than 10 units per quarter.

Again, we try filling in the requirements. By clicking on the filter that only shows courses that can be picked, it becomes clear that we cannot fulfilled all the requirements. However, if we decide to waive two foundations courses – CS 103 and CS 109 – space frees up. Figure 6 shows three possible schedules. Notice how the alignment of the terms brings out the fact that only a few days a week are used.

DISCUSSION

We have built a single application that makes a lot of information easily available and allows advanced interactions to build schedules. We can evaluate Class Picker in the context of the three goals set out in the introduction.

a) Information on specializations and requirements

Class Picker allows the user to pick any of the 10 specialization tracks offered by the department. For each track, the user can inspect each requirement.

b) Information on courses

Class Picker includes all the information from the Stanford Bulletin, with improved navigation. Indeed, there is no limit to the results per page. Moreover, all courses are linked to the requirements they fulfill, which is essential information. Class Picker also shows all the courses that count towards requirements, not just those from the Computer Science department. It is nearly impossible to compile that list in the other tools presented in Related Work.

c) Experimentation and feedback

This is likely the most valuable aspect of Class Picker. Users, once they find the information they want, can also select courses in the same application. Immediate feedback is given for requirement progress and course conflicts. The actual fitting of courses into a schedule is managed behind the scenes by the engine, generating all possible permutations of schedules. This reduces the amount of work for the users and means they do not have to build their schedules on paper or in another window.

FUTURE WORK

There are still several features we need to add to Class Picker before it fully encompassed all the possibilities available when designing a program sheet.

Firstly, we have not surfaced the option to pick a dual specialization. This option is offered by the department, where only 21 units need to be taken in the primary Depth specialization, and 5 courses in the secondary Depth. There is no Breadth requirement in this case. Adding support for dual-depth should not be too difficult, although it brings up new cases where both depths overlap.

Secondly, we would want to add preference to earlier terms when it comes to scheduling courses. This is true because the scheduling information for the second year is simply extrapolated from the current year. There are no guarantees that it is going to be the same. Hence it is preferable to schedule courses in the first year, especially if they are not offered often. Moreover, students generally want to graduate as early as possible. Class Picker sometimes generates schedules with entirely empty terms because they are valid, even though students would probably not opt for these.

Thirdly, we would like to extend the constraints feature. In particular, our user-testing has shown that users would like to limit the number of units for each term separately. A common use case is to do 3 terms with 10 units, and one with 15 units.

Finally, some courses, such as seminars and research, can be repeated for credit, something Class Picker does not support. Students may also want to take courses scheduled at the same time if one of them is video-recorded on SCPD [8].

Beyond the addition of features, we would like to allow the user a little more freedom to play with the results and perhaps deviate from some requirements, since that is permitted by the department in some cases.

CONCLUSION

Class Picker is an application that solves a very real problem for Stanford Computer Science students. The data visualization aspect of the problem arises from the mass of information available from many different sources. Class Picker uses varying levels of details and interactivity to bring all this information into a single application. The second aspect of the problem is cognitive: they are a lot of parts to keep track of when picking courses (requirements, constraints, conflicts). Class Picker's engine takes a lot of this load off the user.

ACKNOWLEDGMENTS

We would like to thank the CS448B staff, its instructor Jeff Heer and the teaching assistants Amy, Megan and Eli, for their feedback and efforts.

Thanks also go to Jim Sproch, who exposed an XML API for accessing course information from the Stanford Bulletin.

Finally, we are grateful to the classmates and friends who volunteered their time for user-testing Class Picker.

REFERENCES

1. Stanford Bulletin Explore Degrees http://exploredegrees.stanford.edu
2. Stanford Computer Science department http://cs.stanford.edu/courses
3. Stanford Bulletin Explore Courses http://explorecourses.stanford.edu
4. Courserank http://www.courserank.com
5. Axess http://axess.stanford.edu
6. Jeffrey Heer, DOITrees revisited: scalable, space-constrained visualization of hierarchical data. *Advanced Visual Interfaces* (2004), 421-424 http://vis.stanford.edu/files/2004-DOITree-AVI.pdf
7. ColorBrewer http://colorbrewer2.org
8. Stanford Center for Professional Development (SCPD) http://scpd.stanford.edu

SOFTWARE CREDITS

1. jQuery.js http://jquery.com
2. Backbone.js http://backbonejs.org
3. Underscore.js http://underscorejs.org
4. jQuery TOOLS http://www.jquerytools.org
5. Bootstrap http://twitter.github.com/bootstrap
6. D3 http://d3js.org