

Skedge

Smarter course scheduling for our University

Dan Hassin

Supervised by
Professor Philip Guo

Department of Computer Science
University of Rochester
Rochester, New York

April 13, 2016

submitted in partial fulfillment of
the requirements for the degree
honors bachelor of science,

and as an open letter addressed to
the information technology staff
at the university of rochester

Contents

Abstract	i
1 Introduction	1
1.1 CDCS	1
1.2 Skedge	3
2 Design in reaction to CDCS	4
2.1 Modernity	4
2.2 Usability	8
2.3 Search friendliness	12
2.4 Peer connectivity	18
3 Very brief technical overview	24
3.1 System architecture	24
3.2 Data storage	24
3.3 Usage data collection	26
4 Data analytics	27
4.1 Usage	27
4.2 Effectiveness as a platform & navigations-per-add	35
4.3 Complexity of users' searches over time	39
5 Conclusion	43
5.1 Proposal to the University	43
5.2 Resources	44
5.3 Acknowledgments	44
Bibliography	46

List of Figures

1.1	CDCS and Better CDCS in their current states	2
1.2	Skedge with the search query <code>csc</code>	3
2.1	CDCS in July 2, 2010	5
2.2	CDCS and Skedge on a mobile browser	6
2.3	Section and subsection presentation in CDCS and Skedge	9
2.4	Hoverable and clickable course mention in the <i>Prerequisites</i> field of a course . . .	10
2.5	Skedge exporting options	11
2.6	Two philosophies of search	12
2.7	Mapping from CDCS’s form-based search to SkedgeQL	13
2.8	Comparison of sharing schedules on Facebook	18
2.9	Skedge’s per-schedule share page	19
2.10	Skedge Social landing page for a user who hasn’t yet joined it	20
2.11	Skedge Social’s schedule feed	21
2.12	Skedge Social integrating into a course search result	22
2.13	Skedge Social’s notifications	23
4.1	Searches over the period Oct 19 2015 to Apr 11 2016	28
4.2	Breakdown of search queries by type of search	29
4.3	Other metrics of Skedge usage over time	32
4.4	Course blocks in Skedge	34
4.5	Frequency of different navigations-per-add values	36
4.6	Frequency of navigations-per-adds for the goal of <i>adding a specific course</i>	36
4.7	Frequency of navigations-per-add/bookmark for the goal of <i>browsing for courses</i> .	38
4.8	Examples of SkedgeQL	40
4.9	User-tracked search complexity over time	41
4.10	Users split by their number of increases in search complexity	42

Abstract

In this paper I present Skedge, a web application for students to comfortably and effectively engage with the University’s course catalog. Skedge matches and surpasses the capabilities of the existing University tool for this purpose, “Course Description / Course Schedule” (CDCS) and presents its information in a more visually appealing way. As a result, Skedge boasts strong user-retention rates, long session durations, and high student adoption despite having virtually no advertisement. Through collected usage data, I demonstrate that **a)** Skedge’s differences from and additions to CDCS are usable and have real-world need, **b)** the three major use-cases associated with course scheduling—direct, exploratory, and peer-guided search—are effectively accommodated by Skedge, and **c)** Skedge’s novel search mechanism is user-friendly and self-teaches to users over time.

Chapter 1

Introduction

I am proud to finally and officially introduce *Skedge*, a website I have been developing since December 2013. Skedge began as a weekend project after far too much time was spent trying to find interesting elective classes, and it wasn't until I briefly presented it at a RocHack "Hacker Night" when I realized the potential it had to help others as well. I never expected the subject of my senior thesis to be something so close to my heart, so I am grateful for such a fun and motivating opportunity to continue work on Skedge, and, selfishly, for an outlet of recognition.

1.1 CDCS

"Course Description / Course Schedule," (or *CDCS* for short) is the University's official tool for browsing the course catalog (1). The CDCS interface consists of fields on the left side for inputting a search, and search results on the right side (see Figure 1.1a). Despite "Course Schedule" being in its name, CDCS does not offer scheduling functionality. Course times can only be viewed, and it is up to the user to keep track of courses considered for the next semester.

1.1.1 "Better CDCS"

Better CDCS (2) is a browser extension, available for Firefox, Safari, and Chrome, that offers a solution to the lack of scheduling mentioned above. It works by injecting "Add Section" and "Bookmark Section" buttons into the existing CDCS interface, and providing a tab on top of the screen that will toggle between the user's schedule and the search results (see Figure 1.1b).

1.2 Skedge

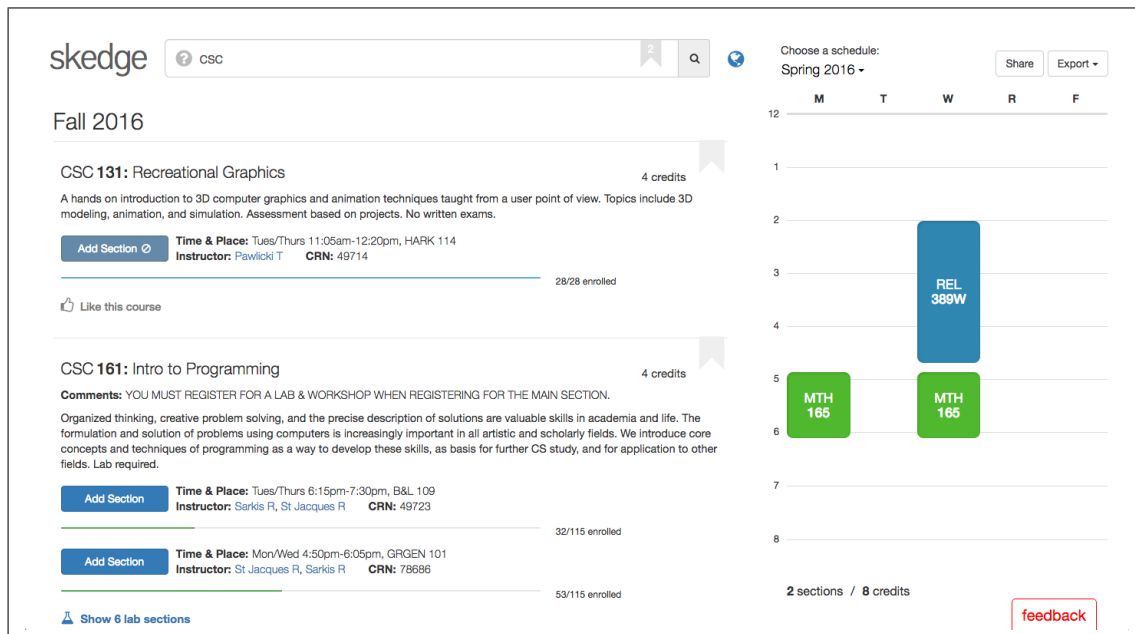


Figure 1.2: Skedge with the search query `csc` and the user’s current schedule on the right

Skedge (3) is, in short, CDCS combined with Better CDCS on a single platform (see Figure 1.2). Like CDCS, it is lightweight, service-oriented, and doesn’t require login, but unlike Better CDCS, it also doesn’t require a separate browser extension.

It matches feature-for-feature with the two tools (search options, information displayed, scheduling and bookmarks, etc.), and aims to improve upon the work of both on several fronts, which will be investigated in this paper in great detail. It is my contention that students, parents, faculty, staff, and administration can all benefit from such improvements.

This paper is organized into two parts, with a brief intermission in between. First, I will explain many of Skedge’s design decisions in response to what I call “grievances” with CDCS. We will break for a technical overview, and the second part will look at live usage data, extrapolating how Skedge is used by students and measuring its efficacy as a platform using some novel metrics.

Chapter 2

Design in reaction to CDCS

From its very inception, Skedge’s functionality and visual design were driven by the shortcomings of CDCS. Skedge is built *bottom-up*, not *top-down*—every aspect of the application was either made as a reaction to a particular grievance in CDCS or as the natural evolution of an existing Skedge feature. Skedge is thus rooted in *usability* derived from real need, not mere conjecture along the question “what could students want?”. Its success with students, shown in Chapter 4, demonstrates that this usability extends beyond my own standards and preferences and can fulfill the various discovered use-cases of students in general.

In this chapter, I invite the reader along on a tour of these grievances and their remedies.

2.1 Modernity

CDCS is an old system, relatively speaking, and its development on user-facing features has been almost entirely stagnant. It launched in 2009, seven years ago, and has hardly changed since. Figure 2.1 shows CDCS in July 2010, which, besides the addition of a few search fields, is identical to its current version. Yet, since its introduction in 2009, we have seen the rise of mobile devices into ubiquity, a boom in “hacker culture” and public APIs, and the capability for standalone web applications to be as sophisticated and dynamic as desktop-class applications without the aid of browser extensions. To this end, Skedge brings course scheduling to the modern era.

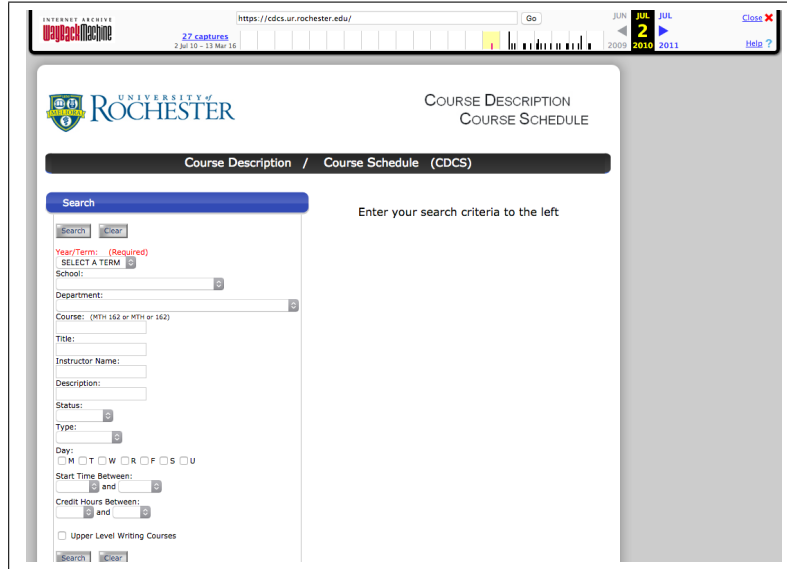


Figure 2.1: CDCS in July 2, 2010, virtually unchanged from today, courtesy of *Archive.org*(4)

2.1.1 AJAX vs. GET requests

CDCS makes an AJAX request with every submitted search, meaning that the server receives the request and returns a response all without any page navigation (i.e. the URL stays the same and no new page is loaded as the search results are displayed). Skedge, however, makes a GET request for every search submission, meaning that the user's browser loads a new page that contains the results and whose URL reflects the search query. This simple technical design decision substantially increases usability for two reasons:

1. Page navigations allow users to leverage their browser history as it was designed—after making several searches, CDCS users who use the back button on their browsers will be brought to the page loaded before the very first use of CDCS, possibly losing time spent in crafting sophisticated searches. Skedge users can go backwards and forwards through their search histories and scroll locations using native browser functionality.
2. Every search query has a unique URL (e.g. <http://skedgeur.com/?q=csc> for *csc*), so users are able to send links to a specific course or search result to others. With CDCS, the URL remains <https://cdcs.ur.rochester.edu> throughout the duration of the session.

2.1.2 Mobile

According to Mary Meeker’s 2015 Mobile Technology Trends from Kleiner Perkins Caufield & Byers(5), in 2014, 51% of adult time spent per day on the Internet was from a mobile device, versus 42% spent on a desktop computer or laptop. Time spent on the Internet with mobile devices reached three hours per day in 2014, compared to less than one hour in 2010 (when there was 12% mobile vs. 75% desktop/laptop time share).

Undeniably, supporting mobile devices and tablets in web applications is crucial for usability nowadays. Note how Skedge responds to the device form-factor in Figure 2.2b, compared with CDCS’s lack of mobile support in Figure 2.2a. CDCS on mobile requires the user to pinch and drag around both to read results and to make new searches, while Skedge adapts content to the device’s screen and fixes the search bar to the top of the screen for easy access while browsing.

Moreover, since no major mobile browser currently supports browser extensions (and if one did, the extensions themselves would most likely need to be re-architected), CDCS on a mobile device loses all scheduling functionality, unlike Skedge, which supports it.

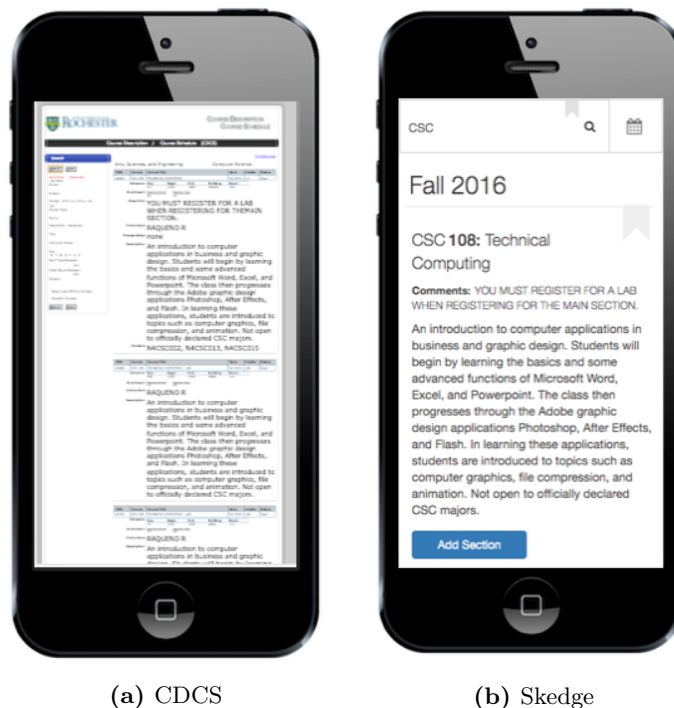


Figure 2.2: CDCS and Skedge on a mobile browser

2.1.3 Public API

With the increasing number of attendees at University of Rochester’s hackathons, it is clear that the University’s “hacker culture” is growing—more students are collaborating to build side-projects that integrate resources and services often benefitting the student community. Open-source and open-information services greatly help to foster such innovation, and having public APIs is essential toward this end (6) (7).

Skedge provides a public JSON API at the root URL <http://www.skedgeur.com/api/>, originally made at the request of a student that was interested in using course data, and the API has since been used in projects by several other student groups. The endpoints included as of now are `/api/courses?q=query` (Skedge’s query language—described in detail in section 2.3—is supported here) and `/api/departments` for a list of department names and codes.

2.1.4 Built-in scheduler

Skedge offers users a course schedule right in the page, unlike CDCS which requires the Better CDCS browser extension. Having a native schedule has several advantages:

1. Besides some CDCS users possibly not even knowing about Better CDCS, not requiring a browser extension provides for a faster and more seamless user onboarding, especially when building schedules on public computers where extensions can’t always be installed.
2. Skedge was designed with a native schedule in mind, whereas Better CDCS works within an interface that wasn’t. CDCS suffers from lower usability as a result, requiring the user to toggle between results and the schedule. Skedge, conversely, provides immediate visual feedback on how courses fit into the user’s schedule when the cursor is hovered over them.
3. Schedule data is centralized on Skedge’s servers as opposed to locally in a browser, meaning that it can synchronize across a user’s devices and can be easily publicly shared to others.
4. Extensions like Better CDCS have limited browser support. Internet Explorer and mobile browsers are unsupported, for example.
5. As separate systems trying to deliver one cohesive experience, Better CDCS and CDCS are dependent on each other. If the format of one changes, the other is likely to break.

2.2 Usability

2.2.1 Visual presentation

Skedge offers several improvements over CDCS in the quality of its data presentation:

1. Displaying information in a rigid, tabular way, CDCS does not leverage fonts and styling to adhere to typographical standards. Instead of using larger or bolder type, for instance, course titles are listed entirely in uppercase (e.g. “INTRO TO PROGRAMMING”), which has been shown to be less readable than lower-case text (8). This problem is compounded when users browse through possibly hundreds of courses. Skedge displays properly capitalized titles styled with large type that helps users to quickly group and locate them.
2. While possibly not a fault of the CDCS system itself, there are very frequently typos or missing spaces in the “comments” section of courses, which Skedge corrects.
3. CDCS displays all course times in 24-hour time, which, despite being concise and unambiguous, is not what most US students are used to. Skedge displays 12-hour time with AM/PM, and prevents ambiguities through the course-in-schedule visualization on hover.

2.2.2 Section display

Often, courses are offered at multiple timeslots, sometimes taught by different instructors and in different rooms. These are called *sections* of a course. CDCS displays each section in a discrete “section box” (all of which are nondistinct and have equal size), even if two sections pertain to the same course. (In this regard, CDCS should really be *SDSS*, “*Section* Description / *Section* Scheduler”, because it operates on the level of sections, not courses.) As a result, course descriptions (which can be lengthy), titles, prerequisites, comments, etc. are all repeated for every section of the course.

To make matters worse, many courses in the University course catalog include what I call *subsections*—secondary sections associated with a course that must be registered for separately. Namely, these are labs, lab lectures, workshops, and recitations. Once again, CDCS displays *all of these* as separate “section boxes” by default, and the course description is yet again repeated for each subsection (which, this time can be tens of times), wasting valuable page space.

Collapsing subsections within courses can result in massive improvements in filtering the data most relevant to the user. For instance, the search “csc” for Fall 2016 on CDCS results in 147 “section boxes”, while Skedge only shows 45 “*course* boxes”, with subsections collapsed within their respective course. This triage reduces the data (noise, more correctly) displayed by **~70%**, and is even higher for departments with more abundant labs and workshops, such as Physics (Skedge: 35 vs. CDCS: 226, **an ~85% reduction**), or Chemistry (Skedge: 25 vs. CDCS: 171, **an ~86% reduction**).

Skedge can reduce the amount of data to scroll through—and thus the time taken to do so—by six- or seven-fold (and possibly more, counting the attention users otherwise have to pay to distinguish courses from subsections), so this design decision has a large usability payoff.

Additionally, some Physics courses (for instance) follow the “A / B” subsection structure, where a student registered for an “A Section” (as opposed to the “B Section”) must also register for an “A Lab” and “A Workshop”. Skedge organizes subsections for these cases to help sort the two out, which get mixed up in CDCS’s linear output.

Note that in Figure 2.3a (CDCS), the first two boxes are sections for the same course, and the next two are labs for that course. Four more lab sections and *twenty* more workshop sessions for that same course follow below the truncated screenshot. Figure 2.3b (Skedge) demonstrates how this information can be conveyed more concisely.

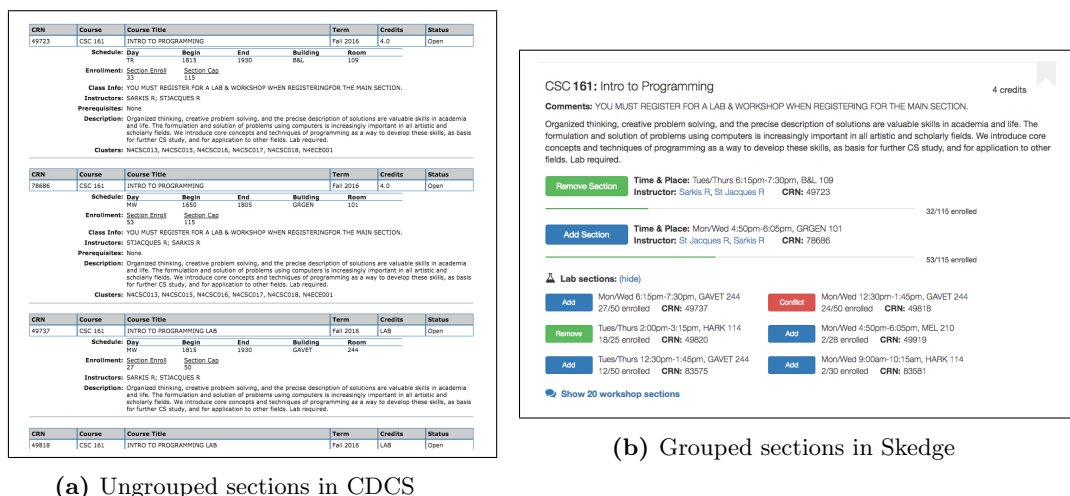


Figure 2.3: Section and subsection presentation in CDCS and Skedge

2.2.3 Course reference

Course mentions will often appear in the prerequisites, crosslists, comments, or description fields of a course (e.g. “**Prerequisites:** CSC 171 or equivalent; MTH 150 is REQUIRED”). Users frequently want to find out more information about mentioned courses (frequency shown in Chapter 4). In CDCS, because course mentions are displayed as ordinary plaintext, users have to scroll back up, make a search for *that* course, and lose their current search context as a result.

Skedge solves this by linking each course mention to its search query, in the style of Wikipedia. Moreover, it prevents context-switches by displaying a popover containing information on that course when the user hovers their cursor over the course mention (see Figure 2.4).

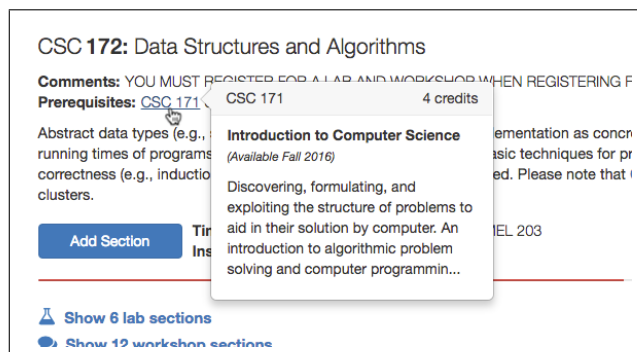


Figure 2.4: Hoverable and clickable course mention in the *Prerequisites* field of a course

2.2.4 Multiple schedule support

With an active schedule still in use, students often want to plan their next semester’s schedule. This is impossible with the Better CDCS extension, which instead adds courses from both terms to the same schedule, sometimes causing non-existent conflicts. A CDCS user must clear their entire schedule out before scheduling a new one. On Skedge, however, when a user adds a section of a term for which they do not yet have a schedule, a new one will automatically be created for that term. Users are also able to access their old schedules.

Additionally, a common feature request for Skedge is the implementation of multiple schedules *per* term, allowing users to experiment with different schedule possibilities.¹

¹Currently under development, but simple to implement given the existing schedule management structure.

2.2.5 Schedule export

Better CDCS provides two ways for users to export their schedule—as an image or to Google calendar—and the Google Calendar export feature is completely non-functional, telling users that “This export tool will only work properly for the Spring of 2015” after the user authorizes the tool to access their Google account on a separate University page (9).

Notably missing as an export option is the *iCalendar* file format (extension `.ics`), which can be imported into many calendar software products. It is the only format the Mac OS and iOS Calendar applications can accept, to offer one example of very popular products that are therefore unsupported by Better CDCS.

Skedge provides the option for users to export their own or another user’s schedule directly to *Google Calendar* (even allowing users to choose a calendar—or create a new one for Skedge—to insert their courses into), as an *image* (JPEG, PNG, or SVG), or in `.ics` format (see Figure 2.5). The `.ics` and Google Calendar exports include course codes, titles, locations, and properly repeat courses until the end of the current semester.

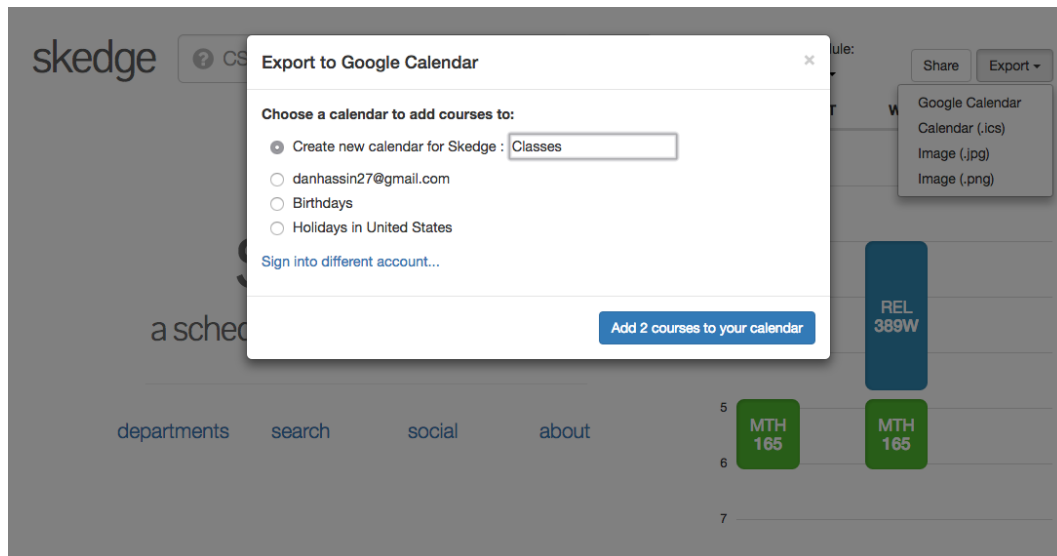


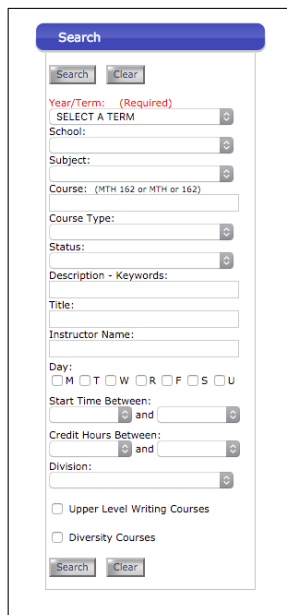
Figure 2.5: Exporting to Google Calendar (note the other export options on the top right)

2.3 Search friendliness

The most important usability concern of all for a course explorer/scheduler is being able to effectively *find courses*. In this section, I will present *SkedgeQL*, a domain-specific query language that is free-form and based on natural language. Next, I will demonstrate how Skedge leverages SkedgeQL to handle the three search criteria students have for finding courses better than CDCS does.

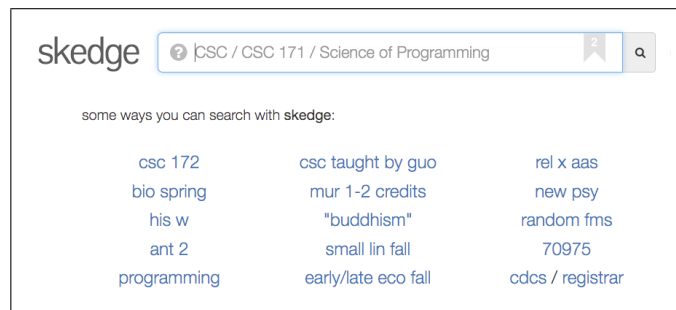
2.3.1 Natural language search

In the spirit of this chapter's theme, Skedge's search method was, again, primarily designed as a reaction to CDCS's. CDCS's search method is a 15-field form (see Figure 2.6a), of which I only found myself regularly using two (one of them being the *required* "Year/Term" field). This prompted me to closely examine the fields to a) determine redundancies between them and b) find how to make some of the valuable filter fields more usable. From this came Skedge's unified, natural language based search method, which I call the Skedge Query Language, or *SkedgeQL*. Figure 2.6b is a list of example possible searches shown to users as they type into the searchbox.



The image shows a web form titled "Search" with a blue header. Below the header are "Search" and "Clear" buttons. The form contains several fields: "Year/Term: (Required)" with a dropdown menu showing "SELECT A TERM"; "School:" with a dropdown menu; "Subject:" with a dropdown menu; "Course: (MTH 162 or MTH or 162)" with a text input; "Course Type:" with a dropdown menu; "Status:" with a dropdown menu; "Description - Keywords:" with a text input; "Title:" with a text input; "Instructor Name:" with a text input; "Day:" with radio buttons for M, T, W, R, F, S, U; "Start Time Between:" with two dropdown menus and an "and" label; "Credit Hours Between:" with two dropdown menus and an "and" label; "Division:" with a dropdown menu; and checkboxes for "Upper Level Writing Courses" and "Diversity Courses". At the bottom are "Search" and "Clear" buttons.

(a) Form-based search in CDCS



The image shows a web interface for "skedge". It features a search box with the text "csc / csc 171 / Science of Programming" and a magnifying glass icon. Below the search box, it says "some ways you can search with skedge:". Underneath, there is a grid of suggested search terms: "csc 172", "bio spring", "his w", "ant 2", "programming", "csc taught by guo", "mur 1-2 credits", "\"buddhism\"", "small lin fall", "early/late eco fall", "rel x aas", "new psy", "random fms", "70975", and "cdcs / registrar".

(b) Examples of SkedgeQL

Figure 2.6: Two philosophies of search

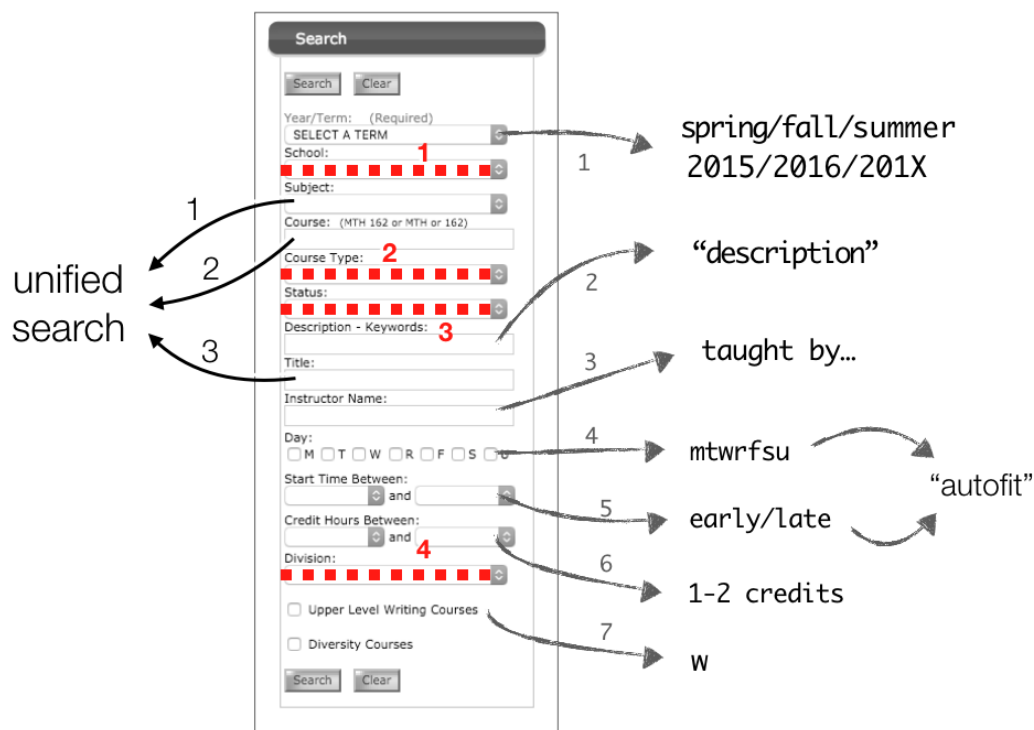


Figure 2.7: Mapping from CDCS's form-based search to SkedgeQL

Figure 2.7 is a mapping of each CDCS field to an element of SkedgeQL or its unified search. Dashed lines that end abruptly are fields that were deemed unnecessary or redundant due to a Skedge-unique feature. To briefly sum up my rationale for each field:

Removed fields

1. *School* (e.g. “College of Arts, Sciences, & Engineering”, “Eastman School of Music”, etc.) was removed in favor of always searching all schools, as is the default on CDCS. Searching by department codes (which are unique across schools) designates a school regardless.
2. *Course Type* (e.g. “Main Course”, “Lab”, “Workshop” etc.) was removed because Skedge already embeds subsections into courses.
3. *Status* (i.e. “Open”, “Closed”, or “Cancelled”) was removed because it is often useful to see closed courses if there is possibility of the instructor letting in more students, and searching for cancelled courses did not seem useful.
4. *Division* (i.e. “Humanities”, “Social Sciences”, etc.) was removed because it seems too broad to be useful, but it could be added to SkedgeQL if there is demand.

Unified Search (left side)

The *unified search* components can be entered directly into the single Skedge search field, along with SkedgeQL. It includes the CDCS fields:

1. *Subject* (department)
2. *Course* (department code and/or course number, e.g. “csc 171”)
3. *Title* (course title)

SkedgeQL (right side)

SkedgeQL is enterable using special keywords and, optionally, qualifiers for those keywords. It includes the CDCS fields:

1. *Year/term*, which is no longer a required field. It maps to the keywords {spring,fall,summer,winter} and a four-digit year to specify term and year, and defaults to the current year and term. Skedge also gets around the fundamental limitation of CDCS of only being able to search one year-term at a time, allowing users to browse *all* of a department’s courses at once, for instance.
2. *Description*, which is specified by surrounding the query in quotes (e.g. “buddhism”).
3. *Instructor name*, which is specified with the keyword **taught by** and given the instructor name as a qualifier (e.g. “taught by Guo”).
4. *Day*, which can be specified with the keywords {m,t,w,r,f,s,u} for filtering by specific days of the week (e.g. “mwf” for Monday-Wednesday-Friday classes only).
5. *Start time between* is replaced by the keywords {early,late} which *sort* (ascending and descending, respectively) the results by start time.
6. *Credit hours between* is replaced by the keyword **credits**, qualified with a number or range (e.g. “dan 1-2 credits” for Dance classes with 1-2 credits).
7. *Upper level writing* is replaced by the single-letter keyword **w** (e.g. “his w” for History upper-level writing courses).

As noted in Figure 2.7, many use-cases for the *Day* and *Start time between* fields can be obsoleted with an *autofit* feature that will be described later in the next subsection.

Advantages of SkedgeQL

The main advantage of Skedge’s search system is in its reduction of 15 fields to a single one. I hypothesize that since the vast majority of searches fit the department code / course number pattern (e.g. “`csc 171`”—this hypothesis later supported in Chapter 4), the time required to force the user to select a term and then find the correct box to enter this query is wasted.

It has been shown that once learned, natural language based query languages can be faster, more intuitive, and more usable than traditional form-based methods (10). WolframAlpha’s hugely successful online free-form search is a notable pioneer in this field (11).

Lastly, SkedgeQL is more easily extendable as new search features are invented. For CDCS, the more form fields that are added to the sidebar, the less usable its user interface becomes. For Skedge, while it may take longer for users to learn a bigger query language, there is no issue with crowding user interface real estate.

Disadvantages of SkedgeQL

One disadvantage of SkedgeQL is the possible grammatical ambiguity of some queries. For instance, the query “Fall of the Roman Empire” could be searching for courses with that as the title, or searching for courses in the *fall term* with the title “of the Roman Empire”. Unless multiple queries are run and more complex natural language processing is done, the system would have a hard time disambiguating between the two. This could also be solved with a “did you mean?” feature, which disambiguates a query into every possible SkedgeQL breakup.

Another notable disadvantage of SkedgeQL is having to know it, and it having a possibly steep learning curve for some users. However, the argument will be made in Chapter 4 that simply by using the site over time, users will self-learn SkedgeQL. One of the ways this is possible is thanks to Skedge’s search system being *multi-purposed*. Hyperlinks around the site—instructor names (which link to “`taught by <name>`”), course references, or schedule course blocks for example—all direct the user to an ordinary search results page with the search field populated, signalling to the user that such a search is possible.

2.3.2 Course selection criteria

I have identified *three* use-cases for course searching (the existence of and distinction between these cases will be demonstrated with collected usage data in Chapter 4). The types of courses user look for are **requirements**, **electives**, and **peer recommendations** (associated with direct, exploratory, and peer-guided search, respectively). SkedgeQL and other application features offer substantial improvements over CDCS for each of these cases.

Requirements

This is for are courses that are required for a student to complete their degree, and are typically searched for directly. The functionality required here is simple and is mostly satisfied by CDCS, but Skedge offers the following improvements to the process:

1. *Crosslisted courses*: For students with more than one major and/or minor, searching for courses that are crosslisted between departments can be valuable in reducing their requirement load. This search filter is unsupported by CDCS, and is supported by Skedge using the operator “x” (e.g. “csc x ece” for courses listed under both Computer Science and Electrical & Computer Engineering departments).
2. *Clusters*: Skedge already stores a users’ previously taken courses, so it can intelligently suggest either already-completed clusters or courses that would complete clusters that are missing one or two courses. For students with many degree requirements already, this could greatly reduce time spent navigating the University’s Cluster Search Engine (for which I also have a long list of grievances, but that lies outside the scope of this paper.)²
3. *CRN*: Surprisingly, search by Course Reference Number is unsupported by CDCS. It is supported by Skedge by just searching the 5-digit number.

Electives

Elective courses can either be courses that are required for a major or minor, but are chosen by the student, or they can be courses of particular interest to students who have the flexibility to

²This feature is under development and is not currently live. It was, incidentally, requested by a Skedge user.

take them. For this use-case, Skedge offers search, filter, and sorting features that substantially aid users in browsing through courses and discovering ones they might want to take. Note that none of the following features are supported by CDCS.

1. SkedgeQL includes the “**new**” keyword, which qualifies an accompanying search by only displaying the courses that were not offered the previous year for that term. For instance, searching “**new csc**” today (Fall 2016) would show Fall Computer Science courses that have been added to the catalog or changed since Fall 2015.
2. SkedgeQL could include an “**autofit**” keyword, which would take into account the user’s current schedule, and only show sections that would not conflict with any already scheduled times.³
3. SkedgeQL includes the “**random**” keyword, which displays a single random course matching the accompanying query. For instance, “**random csc**” would show a random course in Computer Science for the current term. Used in conjunction with “**autofit**”, it can a powerful way to find interesting new classes that are available for users to try.
4. SkedgeQL includes sorting by class size—useful for students trying to find smaller, discussion based classes—or by class starting time—for students with preferences of morning or afternoon/evening classes.
5. Instructor names appearing in courses automatically link to searches for more courses taught by that instructor, which is easier than having to use the CDCS “instructor” field to make a separate search.

Peer Recommendations

CDCS currently has no support at all for peer course recommendation, a highly undervalued resource for course finding. Skedge implements peer recommendations through Skedge Social, a system detailed in the next section.

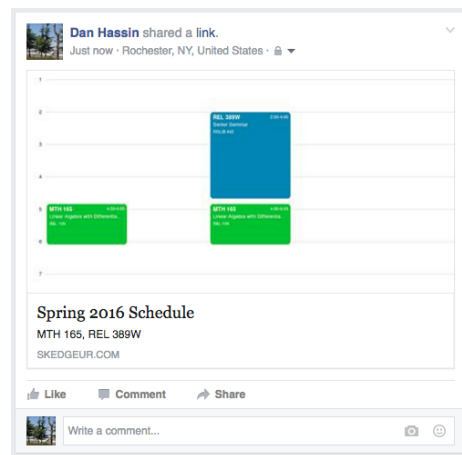
³This feature is under development and is not currently live.

2.4 Peer connectivity

The question “what are you taking this semester?” may possibly be the most common phrase uttered in campus smalltalk within the first few weeks of a semester. But college students’ social lives do not solely exist in person—lately they are dominated largely by social media, most especially Facebook (12). The “what are you taking” question typically translates to social media in the form of the schedule image exported from Better CDCS posted to Facebook (e.g. Figure 2.8a). There are several limitations to this model.



(a) Sharing a static schedule image generated by Better CDCS on Facebook



(b) Sharing a Skedge schedule on Facebook links to the live and dynamic schedule share page

Figure 2.8: Comparison of sharing schedules on Facebook

2.4.1 Limitations of the Facebook model, remedied by Skedge

Static image vs. live site

As mentioned previously, the schedules generated with Better CDCS cannot be shared publicly, and therefore students can only post a static, exported image of their schedule. This excludes the possibility of edits to the schedule being reflected to their peers in the same post. Moreover, referencing courses (i.e. looking up more information on a course a peer is taking) is not very usable under this model as it requires a manual search.

Skedge allows users to generate a public link (of the format <http://skedgeur.com/1234>, with the last four digits being a random ID for that schedule) to a view-only, larger version of their schedules (see Figure 2.9a). If a user visiting others' share pages also has their schedule saved on Skedge, the option to “overlay” the user’s schedule ontop of the share page’s schedule is available, making it easier for friends to find common free time (see Figure 2.9b).

When this link is posted on Facebook, Skedge will include a rendered image of the schedule in the post, maintaining the element of visual bait with posting a static image (see Figure 2.8b).



(a) Skedge’s schedule “share” page. Hovering over a course shows information about it, and clicking searches for it.



(b) The “overlay” feature enabled, superimposing the user’s schedule over the share page’s schedule.

Figure 2.9: Skedge’s per-schedule share page

Robustness of finding courses taken by peers

The Facebook model stimulates users from finding courses peers are taking in several ways. It

1. requires that **a)** peers share their schedules on Facebook publicly and **b)** the user incidentally checks Facebook while that post is still deemed relevant via Facebook’s algorithms.
2. is unorganized, as schedules are spread across space and time (different posts and profiles).
3. is only effective for the *current semester*. It can be very difficult for students seeking advice (or even academic help) for a specific class to find others who had taken it before.
4. is *schedule-oriented*, not *search-oriented*. Course discovery can only happen from seeing peers’ schedules, not from browsing and finding classes that their peers may be taking.

Solving these issues requires platform infrastructure external to Facebook, but Facebook’s ubiquity is invaluable. Since Skedge already saves and archives users’ schedules and Facebook has an integration API, building this platform into Skedge was the natural solution.

2.4.2 Skedge Social

The *Skedge Social* platform aims to leverage existing social media paradigms (e.g. *news feed*, *likes*, and *requests*) to more seamlessly give users answers to the questions alluded to earlier—“what are my friends taking?” and “what do my friends recommend?”. Skedge Social currently only supports Facebook, which can be integrated with a single click (see Figure 2.10).



Figure 2.10: Skedge Social landing page (<http://skedgeur.com/social>) for a user who hasn’t yet joined it. Skedge Social appears in the context of the ordinary Skedge interface, and is accessible with the globe icon to the right of the searchbar.



Figure 2.11: Skedge Social’s schedule feed (the default of four tabs) for a user who has integrated their Skedge account with Facebook. The user can toggle between a peer’s schedule and the list of their *liked* courses.

Schedule feed

Immediately after logging in, the user will see the “schedule feed,” the list of the user’s peers’ current schedules (see Figure 2.11). This takes from a familiar paradigm of the *feed* in social networks, which is routinely scrolled for updates. For each schedule, the user can visualize their own schedule side-by-side (using with the same “overlay” mechanic explained earlier), access the Facebook profile of the schedule’s owner, and see the list of courses that user likes.

This has the advantage of organizing and persisting peers’ schedules, solving the first three issues of the “Facebook post” model listed above.

Search result integration

Skedge Social also integrates into course search results, showing the user any other peers on the platform who are taking, have taken, or *liked* courses in the results list (see Figure 2.12). This has several advantages, which solve the last “Facebook post” model issue listed above:

REL 105: Asian Search for Self

4 credits

The basic teachings of Hinduism and Buddhism as to human nature and the paths to liberation. We shall investigate particularly the ways in which early Vedism, classical Hinduism, Buddhism, and Jainism conceive of the cosmos, meaningful human existence, and life's ultimate goals. Readings include original sources in translation, such as the Upanisads, the Bhagavadgita, and Buddhist scriptures in both Mahayana and Nikaya-based traditions.

Add Section

Time & Place: Tues/Thurs 11:05am-12:20pm, MEL 221
Instructor: Brooks D **CRN:** 38938

20/45 enrolled

Like this course

skedge social

Friends taking/taken this course:

Orion Kelp

Fall 2016

Flute Birdman

Fall 2015

Friends that like this course:

Gloria Rhodes

Figure 2.12: Skedge Social integrating into a course search result

- It enhances the user's social engagement and allows them to better keep up with peers.
- It promotes and facilitates communication for users either seeking or soliciting advice regarding courses (sending a Facebook message is merely two clicks away).
- It brings attention to courses which are particularly popular or liked among the user's community of peers, increasing the chances of serendipitous course discovery.

Personal schedule synchronization

In Section 2.1.4, discussing the disadvantages of Better CDCS's decentralized schedule store, I mentioned the possibility of synchronizing a user's schedules across their devices. Skedge leverages a user's Facebook integration to provide this synchronization. Upon signing into Skedge with Facebook from another device, that device's browser will automatically update with all of the user's data. Modification from any device will propagate changes to the rest.

For this purpose, implementation of a simple login system (either in-house or using NetID, for instance) could easily replace the reliance on Facebook integration. This is currently unsupported, but is a space to explore in the near future.

Privacy and share requests

Skedge offers users two privacy options:

1. “Share my schedule and likes with all my friends” (Public)⁴
2. “Share my schedule and likes only with friends I approve” (Private)

To implement privacy option #2, Skedge Social includes a basic share request/approval system. The main interface has tabs *Share requests* and *Non-sharing friends* (visible in Figure 2.11) for displaying pending share requests for approval, and for sending share requests to the user’s Facebook friends on Skedge that have the private setting enabled, respectively.

Notifications

Skedge Social provides notifications for any pending share requests described above (Figure 2.13a), as well as for any Facebook friends that have joined the platform since the user’s last visit to Skedge Social (Figure 2.13b).

Skedge users therefore do not have to seek out peer schedules—the schedules come to them. Social network notifications are an established, successful model that have been shown to be highly important for social networks’ users and levels of engagement (13). The hope is that Skedge Social’s notifications will evoke the almost Pavlovian “click-and-check” response in users, reminding them of the course network they are a part of and promoting a stronger sense of academic community.



(a) *Pending share requests* notification



(b) *New friends* notification

Figure 2.13: Notifications: none, one, or both of them can adorn the Skedge Social logo

⁴Note that even the “public” option is not truly public, as it only grants view permissions to people who are already friends with the user on Facebook.

Chapter 3

Very brief technical overview

3.1 System architecture

Skedge is a *Ruby on Rails* web application running on a webserver called *unicorn*, and routes ports 80 to it using the *nginx* webserver as a reverse-proxy. Skedge uses *PostgreSQL* for the database, and *React.js* on the front-end, a framework developed by Facebook for responsive user interfaces. This stack is hosted on a paid (by me) virtual machine instance on DigitalOcean.

3.2 Data storage

3.2.1 Courses

At the end of each day, Skedge runs an update script that updates its course data with any new course data that exists or has changed on CDCS, endearingly called “scraping.” It accesses the ordinary CDCS search endpoint, but in XML format. There is minimal text processing to get the data to its final form as it is on Skedge, such as title formatting into proper capitalization. Highly important data such as times, location, or virtually any other field remains unmodified.

A substantial amount works to connect courses to their subsections (workshops, labs, etc.), as these are not explicitly connected in CDCS and sometimes requires basic natural language processing (finding CRN references, title similarity, etc.) Any ambiguities in this process are reported to me and can easily be resolved (most arise from data entry errors in CDCS).

3.2.2 Users & schedules

User accounts are purposefully light-weight on Skedge, because it aims for as little friction and overhead as possible before users can begin to search for courses and build a schedule. Thus there are no logins (besides now with Skedge Social, which is optional), and yet Skedge keeps users' schedules persistent. This is done using a single cookie on the client side that records the user ID that client is associated with, as well as a long hexadecimal string that serves as the “user secret,” or “password”, which must match the one on the server for any changes to be made. This prevents users from manually changing cookie data to modify others' schedules.

The disadvantage of the cookie-based system is that each new browser or device is disjoint by default. Previously, I made it possible to retrieve and set the user secret from the user interface, allowing users to manually synchronize all of their devices' browser cookies to the same user ID and thus manage the same schedules from different devices, but this was replaced by the same feature offered through Skedge Social, as part of promoting the platform. A different way to solve the problem of better persistence while not sacrificing the trade-off of lightweight accounts (and one that doesn't require external services like Facebook) is being considered. A likely candidate at the moment is using University of Rochester NetID for schedule synchronization and improved persistence.

3.2.3 Skedge Social

Out of respect for privacy and a preference to *not* store sensitive data, Skedge does not keep or store any Facebook data, not even users' names or lists of friends, the two pieces of data Skedge requests access to during the integration process. Skedge only stores an anonymous ID (uniquely generated for Skedge by Facebook) per account. Only a user's name and list of friends also using Skedge are accessible via this ID, and Skedge doesn't store this in its database—both are retrieved exclusively by the client, and making separate requests that ask whether any given users (i.e. peers) are taking certain classes, etc., and for when displaying the schedule feed.

Course *likes* are also entirely locally stored and have nothing to do with Facebook—Skedge merely uses the same concept and a similar icon for its own feature. Comments on courses are not currently supported, although Facebook-integrated comments would be possible.

3.3 Usage data collection

As the saying goes, “if it’s not monitored, it’s not in production” (how else would we know if it’s really live?) (14).

Skedge collects usage data that is anonymous with regards to who a specific user is (because that information is not even known or stored), but that is not anonymous with respect to a trackable user ID. Keeping track of certain properties across users is very important for the data analysis in Sections 4.2 and 4.3. This tracking is done by the open-source library *Ahoy*, which also has a JavaScript library to log click events unobtrusively and efficiently (e.g. send them after the page has been loaded) using cookies. Note that this has nothing to do with advertisement tracking that is done by marketing agencies across the web—this tracking is entirely local to Skedge and Skedge does not advertise in any way.

Google Analytics is also enabled on Skedge, which also involves cookie-based tracking, and is used for higher-level analytics such as session duration, number of sessions, etc. (a session is defined by a period of activity expiring after 30 minutes of inactivity.)

Chapter 4

Data analytics

In this chapter I propose three hypotheses, each to be supported by analysis in the coming sections. The hypotheses are:

1. Skedge's differences from and additions to CDCS are **usable and have real need**.
2. Skedge's *navigations-per-add* demonstrate its **effectiveness** in the user search paradigms of a) **direct search**, b) **exploratory search**, and c) **peer-guided search**.
3. **SkedgeQL is user-friendly**; users learn how to make more sophisticated search queries over time **simply by using it**.

While #1 is comparative with CDCS, the other two aim to examine Skedge on its own merit.

4.1 Usage

In this section, I will address the usability of the features listed in Chapter 2 (Design). But first, here are some general statistics of Skedge to date:

4,713 users that have added or bookmarked at least one course
5,256 schedules, with an average of 5.13 and median of 5 sections per schedule
75% of sessions from returning visitors
107 sessions averaged per day
5.4 pages viewed averaged per session
6 minutes spent averaged per session

The figure of 4,713 total users may be misleading because, as explained in Chapter 3, if a user resets their browser cookies or changes browsers or devices, this registers as a distinct visitor. Unfortunately, even analytics tools that track unique visitors rely on cookies as well.

In an attempt to exclude “stale users,” we can time-constrain our analysis. In the past month, there were 1,416 visitors (unique) to the site and 1,136 visitors *with schedules*. However, this metric excludes data from seniors who did not have a need for Skedge in the past month. Going back since November 3, 2015, course registration day (~5 months ago), we find 4,647 visitors and 2,209 visitors with schedules. Thus, we can estimate that roughly **22%** of undergraduate and graduate students (9,470 total) at the College have added courses on Skedge (15).

4.1.1 Search

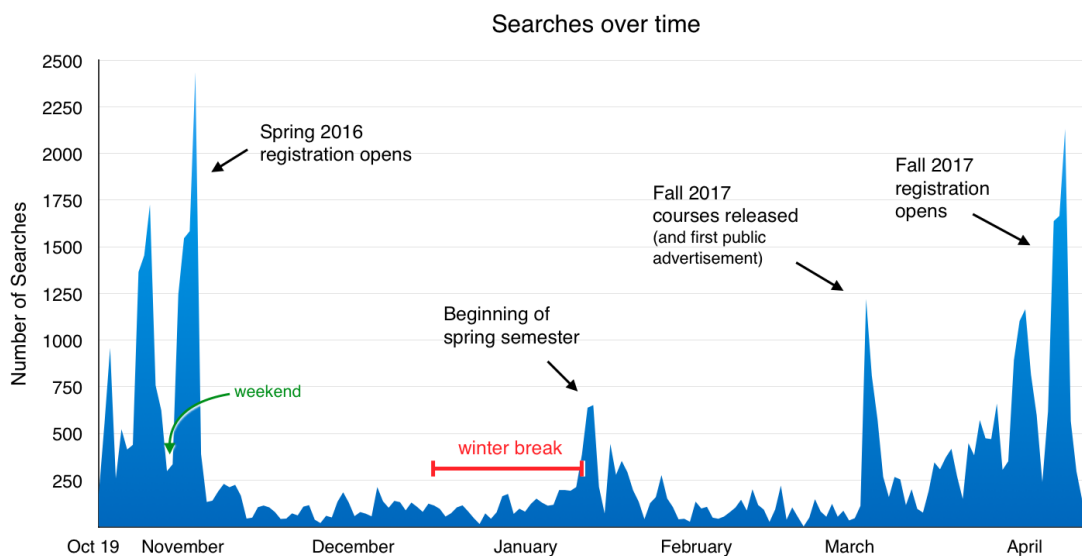


Figure 4.1: Searches over the period Oct 19 2015 to Apr 11 2016, annotated with major events

Figure 4.1 shows searches over time, and makes the periodicity of Skedge usage extremely apparent. Although there is only enough data for a bit more than one “phase,” the cycle of events—*semester begins* (students check course times, room locations, etc. while they get accustomed to the semester), *next semester’s course catalog released* (students explore the courses offered and build possible schedules), *next semester registration opens* (cross-checking and officially registering each course with the Registrar)—clearly drives the majority of site traffic.

What this means is that Skedge is not seen as an experiment or a proof of concept that is not production-ready. Rather, its traffic patterns show that it is consistently *relied upon* by students and used for its intended purpose in times of biggest course scheduling need.

It should also be noted that before March, the majority of the userbase as I knew it consisted of seniors, as the majority of people that I knew personally and shared Skedge with belonged in my class year. The latest two major events reported in the graph above thus very impressively consist of *very few seniors*, who have no courses left to schedule. March also marked the very first time I publicly advertised Skedge, although not very strongly. It was sent out to all students in the Computer Science mailing list as well as posted to two class-wide Facebook groups.

The usability of and user satisfaction with a tool is hard to measure when it holds a monopoly. Considering that Skedge is an *optional alternative* to CDCS, such consistent usage data (especially with 75% of sessions being from returning users) undeniably proves its measures of usability and user satisfaction relative to CDCS.

Search types

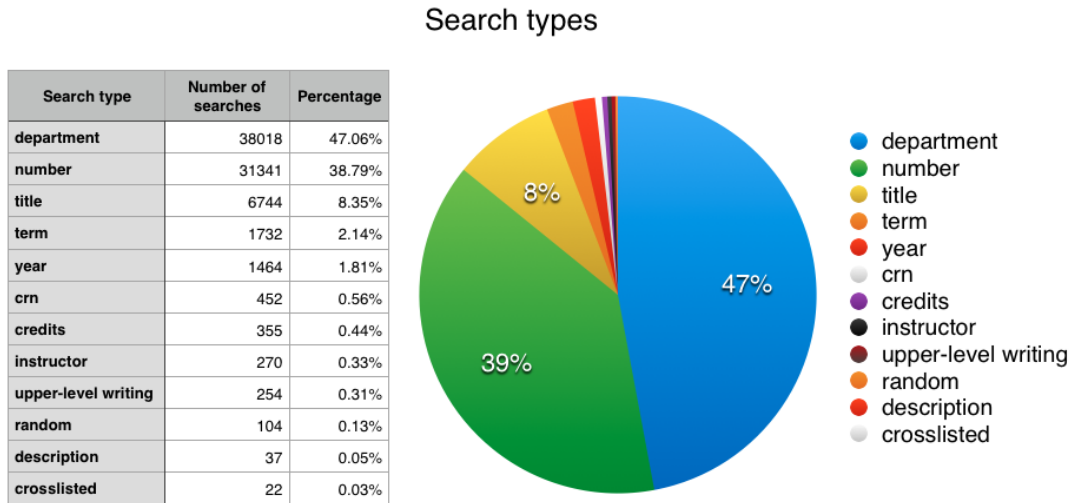


Figure 4.2: Breakdown of search queries by type of search

Figure 4.2 validates the hypothesis made in Section 2.3 regarding the search interface design, demonstrating that the *department code* and *course number* search modes dominate user

searches, accounting for over 85% of all search types on Skedge. Bringing the nondistinct field for this in CDCS into prominence is thus a design choice that accomodates empirical behavior.

Empty searches

Of the total 45,466 searches on Skedge logged since November 2nd 2015, 89.4% (40,625) of searches produce nonempty search results, whereas 10.6% (4,841) of searches come up empty.

This already is evidence of an effective search mechanism, but of those empty searches, the vast majority contain typos or are for courses that simply don't exist. The other significant portion of searches include instructor names without the requisite “**taught by**” for SkedgeQL to pick it up as an instructor search, and this issue could easily be fixed.

This type of analysis is powerful, as it can raise issues in the query language (i.e. searches I didn't implement but that users expect to work) with specific fixes. This is the same method WolframAlpha describes its team using to improve their freeform input system (16). Table 4.1 lists a few notable examples of this nature. (Table 4.2 lists a few funny queries I came across.)

4.1.2 Exports

In Section 2.2.5 (on the design of Skedge's schedule exporting), a major usability improvement claimed was the ability to export schedules to the `.ics` format, a widely used format for many calendar applications, as well as offering functional Google Calendar export. Figure 4.3a shows usage of the export functions over time, demonstrating its consistent and healthy use (traffic surges once again in line with major scheduling events.) This data confirms that there is strong demand for exporting to `.ics`—it is even more common than Google Calendar export.

4.1.3 “Link-searches”

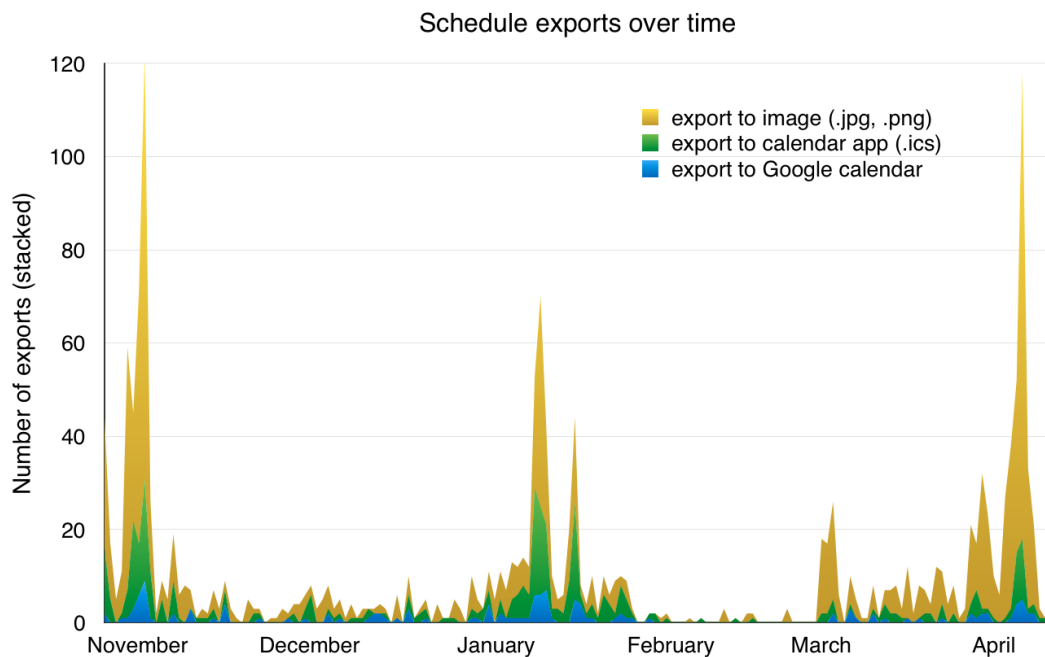
Figure 4.3b shows the amount of clicks on the Wikipedia-style embedded links throughout Skedge (instructor names link to other courses they are teaching, prerequisite and crosslist course mentions link to those courses). The number of manual instructor searches is also included for comparison, which can be seen to be vastly less usable than the analogous link-click, which doesn't exist in CDCS. Additionally, link-clicks also possibly offer a different functionality (e.g. in the course browsing paradigm).

Empty search examples	Specific fix to SkedgeQL required
<code>"Prereq 280"</code> <code>"has-Prereq: 280"</code>	Course dependencies
<code>"tuesday 3:25-4:40"</code> <code>"m/w"</code> <code>"classes tuesdays and thursdays between 11:05 and 12:20"</code> <code>"weekend summer 2016"</code>	Better day-of-the-week handling
<code>"2 credits natural science"</code>	Search by division (although there were very few)
<code>"religion and classics"</code> <code>"studio art"</code>	Search by full department name
<code>"new csc courses"</code>	The word <code>"courses"</code> stimulates the query here (somewhat common with advanced query types)
<code>"curriculum change"</code>	Alternative for <code>"new"</code> keyword
<code>"guo"</code>	Instructor without <code>"taught by"</code> (very common)
<code>"stt212 vs mth201"</code> <code>"stt212 OR mth201"</code> <code>"csc 2{4,8}2"</code> <code>"csc 242 282"</code>	Searching multiple queries at the same time
<code>"crosslisted csc lin"</code>	Probably a more reasonable syntax than the current SkedgeQL one (<code>"csc x lin"</code>)
<code>"place: hubbell"</code> <code>"hubbell friday"</code>	Search by location, although motives unclear
<code>"MTH 2*"</code> <code>"csc 2xx"</code>	Using <code>"*"</code> or <code>"x"</code> instead of an omission (e.g. searching <code>"MTH 2"</code> would work for this purpose)

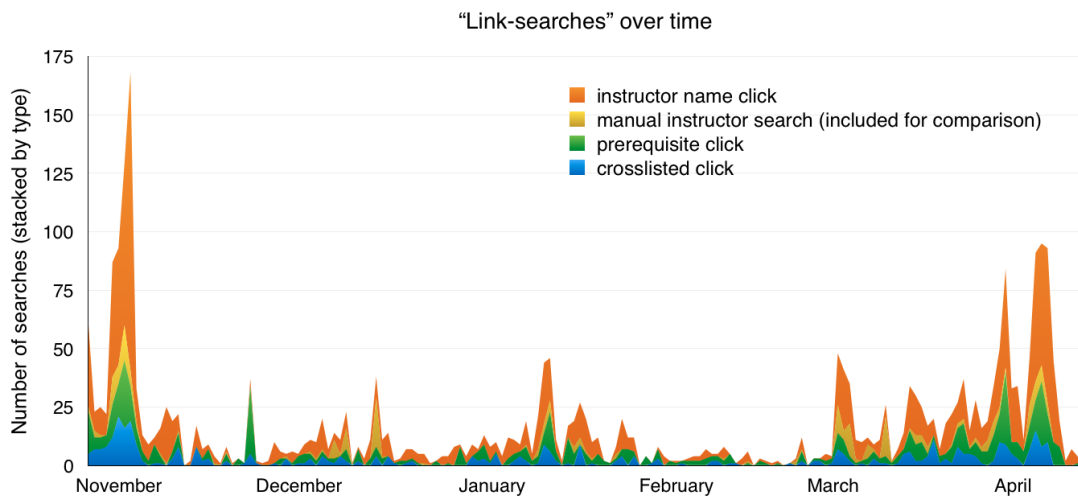
Table 4.1: Example of search queries that produced no results but that indicate demand for added search features. The corresponding features that currently lack in SkedgeQL are described on the right.

<code>"why 123"</code>	<code>"taught by seligman"</code>	<code>"cool stuff"</code>
<code>"psy cool professor"</code>	<code>"interesting courses"</code>	<code>"fun classes in csc"</code>
<code>"easy A"</code>	<code>"flute birdman"</code>	<code>"how do i make a new schedule"</code>

Table 4.2: A sample of some funny searches



(a) Different types of schedule exporting over time



(b) Different types of "link-search" clicks over time, with manual instructor search included for comparison with clicking on an instructor's name

Figure 4.3: Other metrics of Skedge usage over time

4.1.4 Mobile

Making the case for usability superior to CDCS's in the space of mobile support is easy, given that CDCS does not have a mobile-responsive website. But to illustrate the importance of ensuring good user experience on mobile devices, note that 11.54% of all Skedge sessions are from mobile devices, which is very substantial. On average, 2.74 pages viewed are viewed and 2 minutes are spent per session on mobile devices.

The nature of the content also makes Skedge commonly accessed on the go—quickly looking up room numbers while heading to class, for instance, or checking a friend's schedule on Skedge Social to meet them after a class, etc. There is still work to be done to determine the usability of Skedge's mobile site in particular. Surveys could be a good indicator, but I didn't have the chance to administer any.

4.1.5 Social

Since March 1st (40 days), based on 963 sessions on `/social`, Skedge Social has seen...

141 users on the platform (~3.5 per day)
10,242 page views (256 page views/day)
10 pages/session (users cycle through its tabs and repeatedly return to it)
a 5% bounce rate (users interact with the page before leaving)
67% of visits being returning visitors
284 overlays onto friends' schedules (2 per user)

Although the feature is too young for in-depth data analysis, these initial reports show promise, especially in the rate of adoption, its level of interaction, and the rate of returning users to Skedge Social.

4.1.6 Course blocks

As a user adds courses to their schedule, the courses appear as timeblocks on the schedule to the side of the search results. The blocks only display the course code (i.e. department code, course number, and if it is a subsection such as a lab or workshop), but hovering over them displays more useful information like the course title, location, and the exact start and end times.

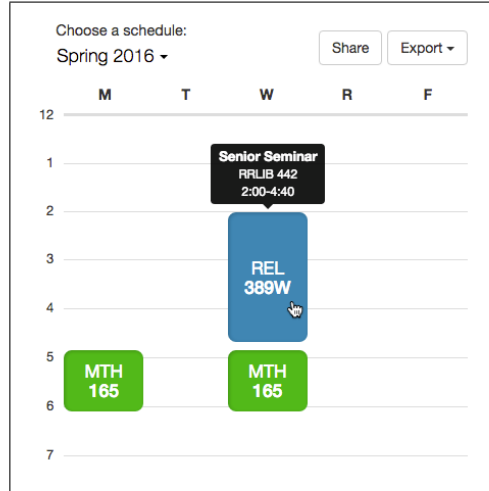


Figure 4.4: Hovering over a course block in Skedge’s side-schedule shows at-a-glance information about it; clicking on it navigates the user to that course

Interestingly, the only course information that is gained by clicking on courses is the description, instructor, and perhaps minor details like course comments, prerequisites, etc. Yet, of 12,944 sessions of users with schedules, **40%** of those sessions had at least one block-click. Those sessions with block clicks averaged **5.12 block-clicks per session**, a strikingly high number considering its limited functional use! How do we explain this?

I hypothesize that a combination of Skedge’s presentation of information, highly interactive user interface, and concern for quality of user experience contributes to a sense of enjoyment in course scheduling. Five clicks corresponds to five courses (both the median and mean for sections per schedule is 5), meaning that in every session, a user could be reviewing details and reading over descriptions for each of their courses. Enabling this sense of pride and admiration with one’s course selection is a crucial aspect of Skedge’s mission. I believe the courses one chooses to take are an integral aspect of one’s personal development and outlook on life in college. I believe ensuring that students are excited not only about their courses, but also about *exploring* what their university has to offer is key to an inspired, intellectually curious student body.

The above could only be confirmed with involved user studies, for which I have no data. At the very least, the average block-clicks per session is a strong case for Skedge’s usability over CDCS for the simple reason that Better CDCS does not even allow users to click on their schedule blocks, and this usage data shows that there is clearly a desire to do so.

4.2 Effectiveness as a platform & navigations-per-add

4.2.1 Definitions

A **navigation** is defined as

- a search, or
- a click on an instructor’s name, or
- a click on a crosslisted or prerequisite course link

The **navigations-per-add** metric is the number of navigations taken by a user (within one session) before a section was added.

To illustrate this metric, imagine that a user searches for “**csc**” (one navigation so far) and begins to scroll through the list of courses. The user sees that Philip Guo is teaching CSC 210 and clicks on his name (two navigations so far), and then adds the section for CSC 210. *A navigations-per-add of 2 is tallied for that user.* The user then clicks on the prerequisite for that class, CSC 172 (one navigation so far), and adds it. *A navigations-per-add of 1 is tallied for that user.* The user then opens the list of labs embedded in 172 and adds one of them. *A navigations-per-add of 0 is tallied for that user.* (Note that a value of 0 can also occur when a user scrolls through a list of courses and adds more than one of them without leaving the page.)

4.2.2 General trends

Figure 4.5 shows the navigations-per-add breakdown for every time a section was added in Skedge. With 40% of sections added in zero navigations and 39% added within one navigation, it is immediately apparent that Skedge delivers users the content they look for in the vast majority of cases, assuming that the intention of the user was to add a course.

Considering the hypothesis claimed in Section 2.3.2 (that course selection criteria can be broken down to *requirements*, *electives*, and *peer recommendations*), however, we ought examine if data for different use-cases are being conflated in this chart.

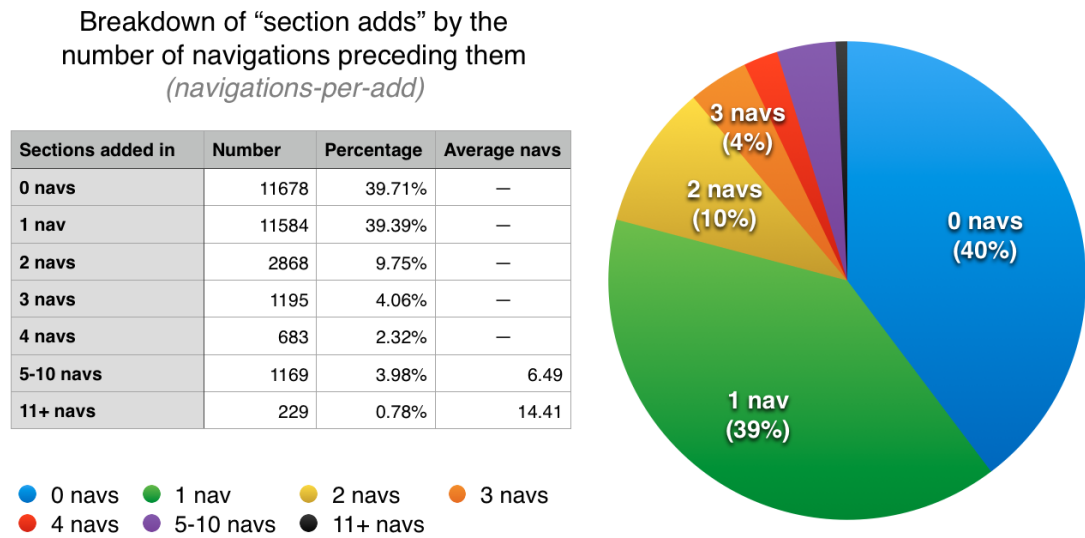


Figure 4.5: Frequency of different navigations-per-add values

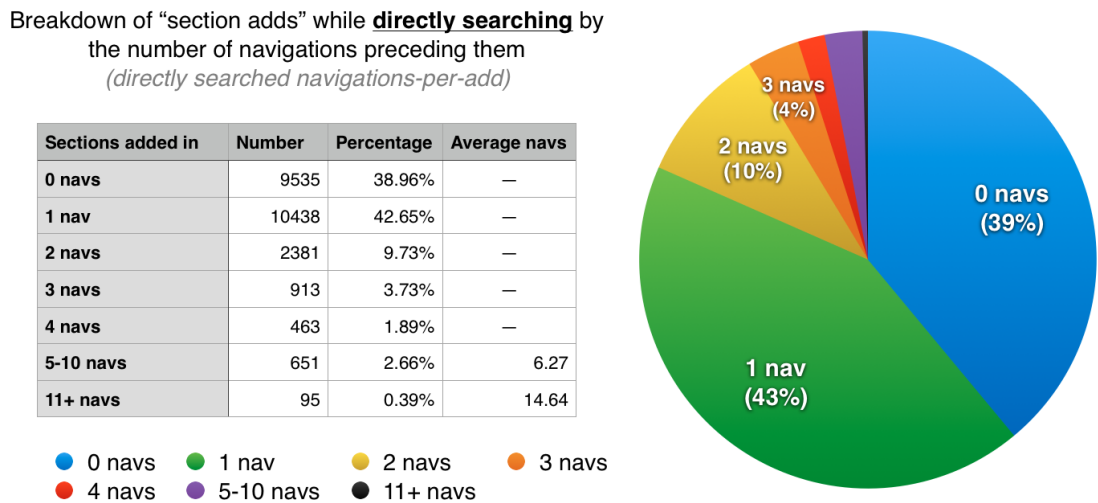


Figure 4.6: Frequency of navigations-per-adds for the goal of *adding a specific course*

4.2.3 Direct search (requirement criteria)

Direct searches are easy to distinguish, as we can take the subset of search queries that target a specific course, namely, those that **a)** contain both department and course code fields (e.g. “**csc 171**”), or **b)** contain a title field (“**intro to programming**”). Figure 4.6 shows the navigations-per-add when, under this criteria, the user’s goal was to add a specific course.

The difference between these results and those in 4.5 is subtle but logical. Users searching for a specific course should have an increased proportion of 1-navigation adds, as the graph shows, because the use-case is *search for course*, *add course*, perhaps occasionally taking an extra try to account for typos, not knowing the title or course code exactly, etc.

But given this reasoning, why would 0-navigation adds still be so common in direct searches? Further analysis on only the sections that were added from direct search and with 0 navigations shows that 64% of those sections are in fact *subsections*. As happened in the illustrative example at the beginning of this section, this shows that a very common use-case is to add labs, workshops, etc. of a course directly after adding the main section.

This hypothesis is further confirmed by doing the same analysis for $navs = 1$ and $navs = 2$, when the inverse behavior would be expected (it makes sense for subsections which were added separately from their main sections to be more rate). Only 11% of sections that were added from a direct search after 1 navigation are subsections, and that number drops to 8% for 2 navigations.

4.2.4 Browse (elective criteria)

Browsing behavior was done as a direct inverse of direct searches, detecting searches that didn’t specifically target any courses. This time, bookmarks were also counted in a user’s flow of events (i.e. they are treated the same as an add), because bookmarking is also a goal of course browsing behavior.

The results are shown in Figure 4.7. The difference between these results and the initial combined results of Figure 4.5 are more substantial than last time, and may defy expectations at first glance. While it may seem intuitive that browsed-for courses should have higher amounts of navigations, note that the graph itself is showing *effectiveness* of such browsing. In this case,

Breakdown of “section adds” or bookmarks while **browsing**
by the number of navigations preceding them
(*browsed navigations-per-add/bookmark*)

Sections ad/bk'd in	Number	Percentage	Average navs
0 navs	3464	52.15%	—
1 nav	2442	36.76%	—
2 navs	485	7.30%	—
3 navs	149	2.24%	—
4 navs	62	0.93%	—
5+ navs	41	0.62%	5.88

● 0 navs ● 1 nav ● 2 navs
● 3 navs ● 4 navs ● 5+ navs

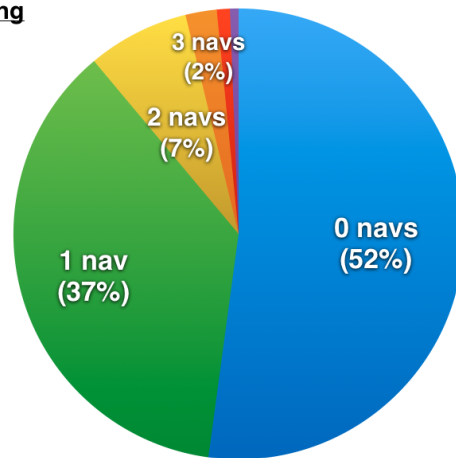


Figure 4.7: Frequency of navigations-per-add/bookmark for the goal of *browsing for courses*

it makes sense that a significantly higher proportion of adds/bookmarks occur within zero navigations, because that indicates a high-level search and subsequent adds/bookmarks from that list instead of moving on to another search.

As expected, and unlike the results of further direct-search analysis, the sections added/-bookmarked when browsing within 0 navigations were mostly *main* sections (only 37% of them were subsections). This makes sense, as users may not be concerned with subsections if the goal of the browse is merely to mark interesting courses. This criterion saw more drastic drops in subsection amounts as well, as the number of navigations increased: 5% subsections for $navs = 1$, and 4% subsections for $navs = 2$.

4.2.5 Peer-guided (recommendation criteria)

Given more time with Skedge Social being live and more forethought in tracking its usage, similar data analysis could be done on the subject of the effectiveness of Skedge in peer-guided course finding. Unfortunately, usage data to answer questions such as “how many Facebook friends were listed as liking or taking a course when it was added?” was not tracked, and cannot be retroactively computed due to Skedge’s purposeful privacy limitations (Facebook friend data can only be retrieved client-side—no data is stored on the server that could authorize such a data request). Thus, part (c) of hypothesis #2 cannot be verified.

4.3 Complexity of users' searches over time

4.3.1 Definitions

The **complexity** of a search query is the number of fields that aren't *course number* or *department* that it spans. The accepted fields are shown in the following table:

description	title	credits	crosslisted	CRN
instructor	year	term	upper-level writing	random

Table 4.3: “Complex” search fields (omits only *number* and *department*)

We can compute the number of searches it takes a user to reach a new highest search complexity record since the previous record. See Table 4.4 for an illustrative example.

A user's **first increase in search complexity** is defined as the number of searches a user made before searching with complexity greater than zero. If a user's first search has complexity 1, the user's *first increase* is 0.

A user's **second increase in search complexity** is defined as the number of searches a user made before searching with complexity greater than their *first increase*'s search complexity.

It is possible that a user has neither metric, both metrics, or only a *first increase* metric.

Query no.	Search query		Complexity	Searches till increase
#1	“CSC”	→	0	—
#2	“MTH 165”	→	0	—
#3	“csc taught by guo”	→	1	$3 - 1 = 2$ (<i>first increase</i>)
#4	“random mur 1-2 credits”	→	2	$4 - 3 = 1$ (<i>second increase</i>)

Table 4.4: Illustration of computing the number of searches before a complexity increase

4.3.2 Motivation

The purpose of this metric is to determine the usability and learning curve of SkedgeQL. The two ways Skedge currently offers to teach users about its query language are

1. by providing a table of example queries whenever the user clicks inside the search field from the Skedge homepage (see Figure 4.8), and
2. from embedded links on the site making searches with high complexity and implicitly demonstrating them to users by using the same input field they use to submit searches.¹

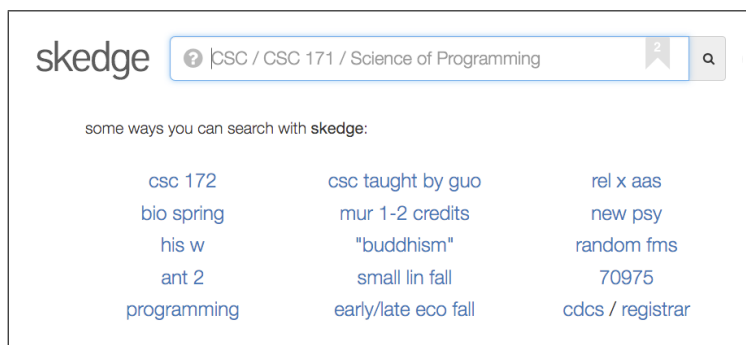


Figure 4.8: Examples of SkedgeQL

The latter method is, of course, much preferred to the former as it teaches without requiring concerted effort on the part of the user to study and learn the query language, and mostly likely remains more memorable and intuitive as a result.

A *high* percentage of users with *first increase* and *second increase* metrics, with *low* mean and median values for those increases would be desirable. This would strongly indicate the effect of teaching method #2 because the factor in teaching users about the query language would simply be time (besides the case of users becoming curious about advanced search types and looking them up explicitly, which is difficult to account for because the query example box is shown to all users fairly regularly).

¹Clicking on an instructor's name, for instance, makes the advanced search "for the user." Another example is clicking on a course box in the side schedule, which searches for that course code along with its term and year (e.g. "CSC 171 Fall 2016").

Other possibilities include linking the credits field of each course to a credits search, the CRN field of a section to a CRN search, the "W" part of the course code to an upper-level writing search, having a crosslisted course link search the crosslisting of the two departments (instead of directly link to the crosslisted course), or a "feeling lucky?" button for a search with "random".

4.3.3 Trends

Analysis of users with search complexity increases

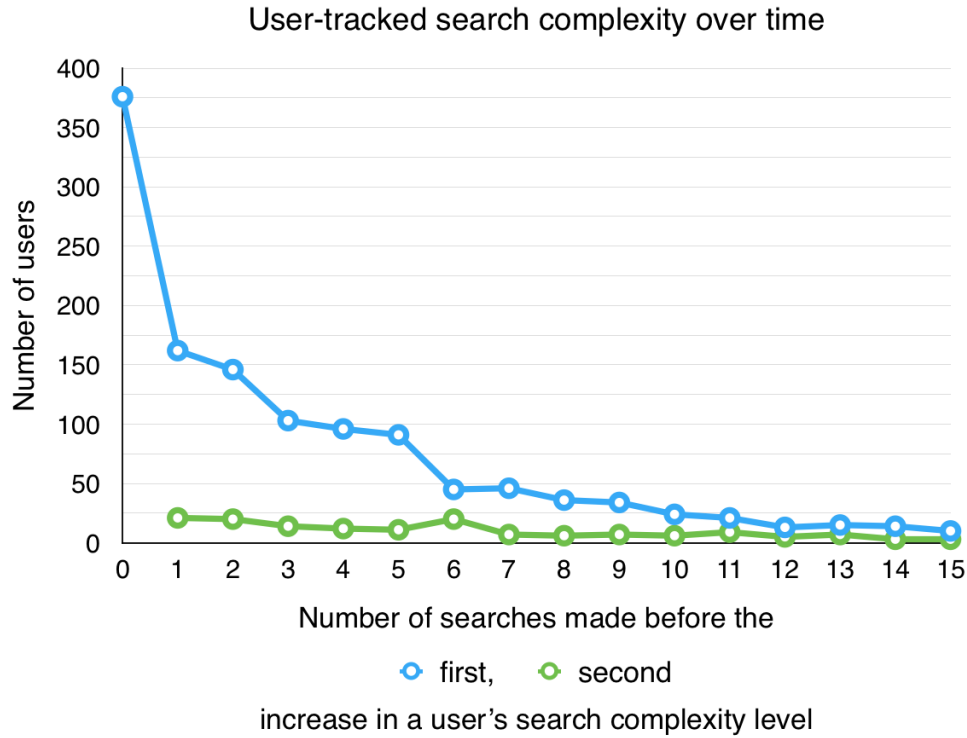
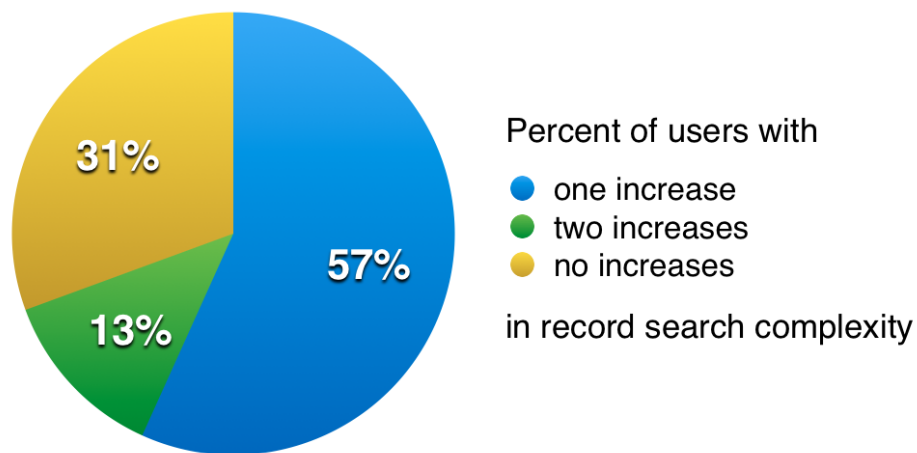


Figure 4.9: User-tracked search complexity over time

The blue line in Figure 4.9 plots the number of users that had their *first increase* in search complexity per search index (searches #16–#25 represent 47 users, and 26+ searches represents only 29 users). The green line plots the number of users for a *second increase* at the given search index (searches #16–#25 represent 35 users, and 26+ searches represents 32).

The amount of users whose first searches are of nonzero complexity may be surprising. The vast majority of these are **title** searches however, which *are* included in the “complex” field table. This is purposeful—title searches only represent 8% of all search field uses, as we saw in Figure 4.2 of the section before last. Moreover, a user’s understanding and/or intuition that the Skedge search field is free-form and can be used for different search fields simultaneously is already a significant conceptualization of SkedgeQL, albeit for a relatively basic search type.

Users with increases vs. users without increases



	First increase	Only first increase	Second increase	No increase	Total users sampled
Count	1311	1072	239	578	1889
Percent	69.40%	56.75%	of total: 12.65% of 1st inc.: 18.23%	30.60%	
Median	2		10		
Average	4.66		18.31		

Figure 4.10: Users with a schedule and more than one search split by their number of increases in search complexity

Figure 4.10 shows that 69.4% of Skedge users (*users* here defined as having at least one course bookmarked or added to a schedule, with more than two searches since November 3rd—there are 1889 such users in total) have increased their record search complexity at least once, with a median of 2 searches to do so. Of those users, 18% of them (12% of total users) have had a *second* increase in search complexity, with a median of 10 searches to do so. 30% of users have only searched with the department and course number fields.

I believe this is a strong model for evaluating the friendliness and low learning curve of SkedgeQL, and the results are certainly promising, although further analysis on beginning with nonzero search complexity is needed.

Chapter 5

Conclusion

5.1 Proposal to the University

On behalf of myself and the roughly two thousand students who regularly and voluntarily choose to use Skedge, I urge the University to consider adopting it as an official replacement to CDCS.

Given the excellence of the University's inspiring faculty and academic programs, I know that I am not alone in my disappointment with what I consider a low regard for usability in many student tools, as well as the lack of communication channels with students to improve them. Throughout the course of this paper, I have shown that the space of course scheduling holds vast potential for innovation, and that CDCS barely scratches that surface. An officially adopted Skedge—or an otherwise substantial overhaul of CDCS—would demonstrate that the University pays attention to its students and to how they use the tools it provides them.

While functional, CDCS nevertheless degrades student experience on campus and discolours perception of the administration's care for students, especially when modern, relevant, and engaging tools are within arm's reach, and most especially at a time when University of Rochester's very high tuition rates continue to rise year-to-year.

As a graduating senior, I cannot commit to continuing the upkeep of server costs and system maintenance for Skedge in the long term, and certainly not indefinitely. For the sake of the University's students, parents, faculty, and staff, I hope that my efforts with Skedge were not in vain and can lead to a substantial refresh of University tools that, most crucially, **involves a dialogue between the administration, students, and staff**. Thank you for reading.

(The views expressed in this section are entirely my own and were not reviewed by my thesis advisor or any others.)

5.2 Resources

Source code

The source code for Skedge and the scripts used to generate the analytics in Chapter 4 are available online under an open source license:

<https://github.com/RocHack/skedge>.

Live site

The site can be found at <http://skedgeur.com> or <http://skedge.org> (very recently acquired, simply redirects to the former).

5.3 Acknowledgments

I would like to give my foremost thanks to Dr. Philip Guo, who very thoughtfully reached out to *me* about using Skedge for research, and whose deep care and support for the project has since reignited and bloomed its development. I also want to give enormous thanks to Ani and Mark Gabrellian for their generous Mesrob Mashtots Innovation Grant donation which funded Skedge’s development in the summer of 2015. Finally, a huge thanks to all those students and my close friends who reported bugs, gave feedback, and in general supported, promoted, got excited about, and complained about Skedge!

Bibliography

- [1] “UR Course Descriptions / Course Schedules (CDCS).” <https://cdcs.ur.rochester.edu/>. Accessed: 2016-04-12.
- [2] “Better CDCS – Chrome web store.” <https://chrome.google.com/webstore/detail/better-cdcs/adaiboomihahdddciolkcfhalmdnlneh>. Accessed: 2016-04-12.
- [3] “skedge.” <http://www.skedgeur.com/>. Accessed: 2016-04-12.
- [4] “UR Course Descriptions / Course Schedules (CDCS).” <https://web.archive.org/web/20100702192949/https://cdcs.ur.rochester.edu/>. Accessed through WayBack Machine at Archive.org for 2010-07-02.
- [5] M. Meeker, “Internet trends 2015,” *Kleiner, Perkins, Caufield and Byers (KPCB)*, 2015.
- [6] K. Milberry and S. Anderson, “Open sourcing our way to an online commons: Contesting corporate impermeability in the new media ecology,” *Journal of Communication Inquiry*, vol. 33, no. 4, pp. 393–412, 2009.
- [7] S. Bratus, “Hacker curriculum: How hackers learn networking,” *IEEE Distributed Systems Online*, vol. 8, no. 10, 2007.
- [8] C. Wheildon, *Type and Layout: How Typography and Design Can Get your Message Across – Or Get in the Way*, p. 62. Berkeley: Strathmoor Press, 1995.
- [9] “Export classes to Google Calendar.” <http://www.rochester.edu/registrar/course-scheduler/gcal.php#>. Accessed: 2016-04-12.
- [10] K. Elbedweihy, S. N. Wrigley, F. Ciravegna, D. Reinhard, and A. Bernstein, *The Semantic Web: ESWC 2012 Satellite Events: May 27-31, 2012*, ch. Evaluating Semantic Search Systems to Identify Future Directions of Research, pp. 152–162. Berlin, Heidelberg: Springer, 2015.
- [11] “WolframAlpha: computational knowledge engine.” <http://www.wolframalpha.com/>. Accessed: 2016-04-12.
- [12] S. A. Golder, D. M. Wilkinson, and B. A. Huberman, *Communities and Technologies 2007: Proceedings of the Third Communities and Technologies Conference, Michigan State University 2007*, ch. Rhythms of Social Interaction: Messaging Within a Massive Online Network, pp. 41–66. London: Springer, 2007.

- [13] S. Bouraga, I. Jureta, and S. Faulkner, “An empirical study of notifications’ importance for online social network users,” *Social Network Analysis and Mining*, vol. 5, no. 1, pp. 1–34, 2015.
- [14] G. Gheorghiu (@griggheo), “It’s not in production until it’s monitored and graphed.” <https://twitter.com/griggheo/status/62239661568958464>, 2011.
- [15] “About the University of Rochester.” <https://www.rochester.edu/aboutus/>. Accessed: 2016-04-12.
- [16] “Wolfram|Alpha officially launched.” <https://www.wolframalpha.com/media/pressreleases/wolframalpha-launch.html>, 2009. *Four Pillars of Wolfram|Alpha: Intuitive Language Understanding*.