INFORMS Transactions on Education

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Course on Advanced Software Tools for Operations Research and Analytics

Iain Dunning, Vishal Gupta, Angela King, Jerry Kung, Miles Lubin, John Silberholz

To cite this article:
Iain Dunning, Vishal Gupta, Angela King, Jerry Kung, Miles Lubin, John Silberholz (2015) A Course on Advanced Software Tools for Operations Research and Analytics. INFORMS Transactions on Education 15(2):169-179. http://dx.doi.org/10.1287/ited.2014.0131

Vol. 15, No. 2, January 2015, pp. 169–179
ISSN 1532-0545 (online)
INFORMS Transactions on Education

http://dx.doi.org/10.1287/ited.2014.0131

A Course on Advanced Software Tools for Operations Research and Analytics

Iain Dunning, Vishal Gupta, Angela King, Jerry Kung, Miles Lubin, John Silberholz
Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139,
{idunning@mit.edu, vgupta1@mit.edu, aking10@mit.edu, jkung@mit.edu, mlubin@mit.edu, josilber@mit.edu}

It is increasingly important for researchers and practitioners to be familiar with methods and software tools for analyzing large data sets, formulating and solving large-scale mathematical optimization models, and sharing solutions using interactive media. Unfortunately, advanced software tools are seldom included in curricula of graduate-level operations research (OR) and analytics programs. We describe a course consisting of eight three-hour modules intended to introduce master's and Ph.D. students to advanced software tools for OR and analytics: machine learning in R, data wrangling, visualization, big data, algebraic modeling with JuMP, high-performance and distributed computing, Internet and databases, and advanced mixed integer linear programming (MILP) techniques. For each module, we outline content, provide course materials, summarize student feedback, and share lessons learned from two iterations of the course. Student feedback was very positive, and all students reported that the course equipped them with software skills useful for their own research. We believe our course materials could serve as a template for the development of effective OR and analytics software tools courses and discuss how they could be adapted to other educational settings.

Keywords: active learning; teaching analytics; teaching optimization; teaching statistics; teaching with technology; visualization
History: Received: May 2014; accepted: August 2014.

1. Introduction

Advanced software tools are a critical part of modern operations research (OR) and analytics practice. Often, "data wrangling" and visualization with a statistical package like R (R Core Team 2014) or Python's pandas package (McKinney 2012) are some of the first steps taken when handling the large, complex data sets that are encountered in real-world applications. State-of-the-art optimization solvers like CPLEX (IBM 2013) or Gurobi (Gurobi Optimization 2014) are often needed to efficiently solve mathematical programs. Parallel and distributed computation using a cluster of computers is sometimes the only way to feasibly complete a large-scale analysis. Finally, conveying insights to make an impact with a nontechnical collaborator frequently requires representing solutions with interactive media or distributing them over the Internet. The end-to-end workflow of modern OR and analytics practice requires fluency with a spectrum of software tools.

Although some OR programs have begun integrating computational elements into their curricula (Alpers and Trotter 2009), few formally introduce students to a broad range of software tools. For example, we reviewed the course descriptions for eight top OR programs[1] for coursework pertaining to solving large-scale linear optimization problems. Large-scale linear optimization is one of the cornerstones of OR practice and a clear opportunity for teaching advanced software tools. Thirteen courses mention techniques for solving large-scale linear optimization problems in their course description. However, of the nine courses with publicly available syllabi, only five (56%) covered using software tools for optimization, and of the four courses with publicly available homework assignments, only one (25%) required students to implement computational techniques for large-scale linear optimization.

On the other hand, some of these tools are covered by courses offered by other departments out-

[1] Industrial and Systems Engineering at the Georgia Institute of Technology; Operations Research and Industrial Engineering at Cornell University; Industrial Engineering and Operations Research at Columbia University; Industrial Engineering at the University of California, Berkeley; Decisions, Operations and Technology Management at the University of California, Los Angeles; Management Science and Engineering at Stanford University; Operations Research and Financial Engineering at Princeton University; and Industrial and Operations Engineering at the University of Michigan.

side of OR programs. For instance, many computer science programs offer courses on visualization,[2] and numerical computation programs offer courses on distributed and parallel computation.[3] Individually, however, these courses fail to fully cover the spectrum of tools required in OR and analytics practice. Moreover, these computer science and numerical computation courses typically focus on theoretical issues and implementation challenges as seen through the lens of those fields, whereas often OR and analytics practitioners are seeking a more applied "How do I use this tool?" perspective. Finally, these sorts of courses are not universally available for students; for instance, our university, the Massachusetts Institute of Technology (MIT), does not offer regular semester-long data science or visualization courses.

To address the need for courses covering advanced software tools for these topics, we developed *15.S60*: *Software Tools for Operations Research*, an MIT course devoted entirely to these tools and the end-to-end workflow of OR and analytics practice. The course, a series of three-hour modules designed and taught by graduate students, launched during the 2013 winter term and ran a second time during the 2014 winter term. The course is targeted at doctoral and master's students, though two advanced undergraduate students have completed the course. Participants are expected to have already taken coursework in machine learning and optimization, and all students are required to have taken a graduate-level course in optimization in order to register. The course is designed as an introduction to advanced software tools for OR and analytics but not as an introduction to programming; participants are required to have familiarity with some programming language.

In this paper, we describe the curriculum and lessons learned from two iterations of this course. In §2, we describe our course design philosophy, citing relevant educational literature that informed our decisions. In §3, we detail the individual course modules and summarize student feedback about these modules. In §4, we describe lessons learned from the second iteration of the course. Finally, in §5 we review overall course feedback and discuss how our course materials could be adapted for use in another program. The supplemental files (course_content.zip) for this paper (available as supplemental material at http://dx.doi.org/10.1287/ited.2014.0131) include a full set of course materials from the second iteration of the course, including lecture slides, assignments and solutions, and heavily commented example code.

[2] Examples include the Georgia Institute of Technology's CS 7450, Harvard University's CS 171, the University of California, Berkeley's CS 294-10, Stanford University's CS 448B, and Indiana University's Massive Open Online Course (MOOC) *Information Visualization*.

[3] Examples include Cornell University's CS 5460, the Georgia Institute of Technology's CSE 6220, Harvard University's CS 264, Princeton University's COS 598A, and University of Illinois at Urbana–Champaign's ECE 408.

2. Design Philosophy

Before delving into the details of the content of each module, we summarize our overall design philosophy, drawing attention to issues many educators may face when creating a course on state-of-the-art software tools. Our ultimate focus was on creating a pragmatic course to empower students to use software in their own research and projects. This design philosophy, in turn, helped shape the structure and content of the course.

2.1. Active Learning and a Workshop Environment

Perhaps the most critical element of our design philosophy was to create a workshop environment that would promote active learning and enable students to be highly engaged with their own learning processes. Active learning has been defined as "instructional activities involving students in doing things and thinking about what they are doing" (Bonwell and Eison 1991). This well-studied pedagogical method has been shown to enhance deeper, more meaningful learning (Smith et al. 2005). OR educators have reported success in using active learning in topics such as service operations management (Behara and Davis 2010) and linear programming (Devia and Weber 2012, Kydd 2012).

To facilitate an active learning environment throughout our course, students were required to bring laptops to each module. Class time was a mix of lecturing to introduce the new tool, group coding exercises during which the instructor would live-code on a projected screen, and short exercises for which students would break off into small teams while the instructor circulated to give one-on-one feedback. By working in class, students could collaborate with partners, providing an opportunity for students with weaker programming skills to learn from classmates with stronger skills. Moreover, technical and syntactical issues were easily addressed by the instructor in real time, allowing students to focus on the higher-level learning objective of the exercise. Nearly all in-class exercises were accompanied by a more challenging "bonus exercise," which provided the most advanced students in the course with an opportunity to further hone their skills. Student feedback substantiated our opinion that the workshop format for the course was more effective than a traditional lecture format would have been.

2.2. Balancing Modularity and Integration

By nature, a software tools course covering a range of tools and concepts will take on a modular design—each module will cover a specific tool or technique. Specifically, we structured our course as a series of eight three-hour modules detailed in §3. This modularity provides a number of advantages:

Simplified course updates. As technology evolves, state-of-the-art tools necessarily change, and to keep up to date, course content must be updated. Indeed, between the first and second iterations of our course, one module was dropped, two were added, two were substantially changed, and four remained similar. Modular design simplifies the process of updating some content while leaving other content unchanged.

Facilitating repeat enrollment. Changing the modules taught each year encourages students who previously took the course to reenroll or audit select modules again in later iterations. Of the students who attended the first iteration, approximately 20% attended at least one module in the second iteration.

Simplified development with multiple instructors. Software tools courses are well suited for multiple instructors. With seven instructors for eight modules, we were able to ensure instructors were resident experts in the material they taught, often having extensive industrial experience with the tools they were covering. Modular course design limits the dependency between material, streamlining and simplifying the course development process with multiple instructors.

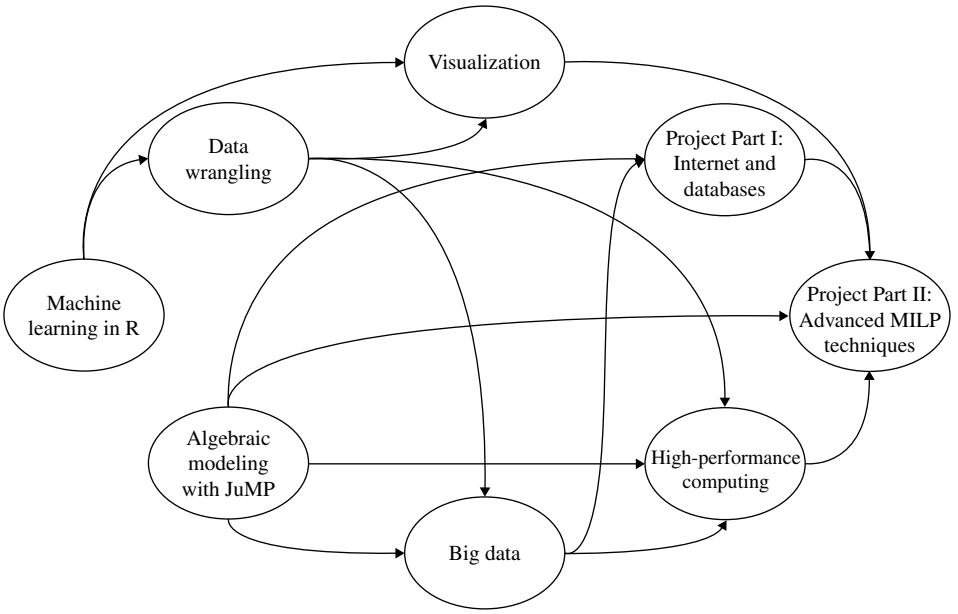
Despite the advantages of modular design, there is evidence that integrated curricula can improve educational outcomes (Vars 1991, Bransford et al. 2000). Consequently, we employed four techniques to partially link the modules together, while retaining the benefits of modularity. Figure 1 summarizes how modules were interconnected; an arrow from module A to module B indicates that module B relies on material from module A.

Recall through in-class exercises. In most modules, we incorporated programming exercises (described in §3) that relied on a previous module but that were simple enough that students who had not attended the previous module could seek assistance and still benefit from the exercise. We felt, and noticed in course feedback, that these small exercises helped students link together the modules and increase retention through knowledge recall and repetition.

Reusing programming languages. In the second iteration of our course, we limited instruction to the R (R Core Team 2014) and Julia (Bezanson et al. 2012) programming languages. Though this decision introduced dependencies to the modules where we introduced these two languages and limited the software tools we could teach, students reported the continuity in programming language to be beneficial. During our first iteration of the course, we taught using five programming languages over seven modules, and students complained that this led to cognitive overload.

A single, consistent data set. We used the Hubway Data Visualization data set (Hubway 2012), an open-source data set released by Hubway (Boston's bike-sharing program) as part of a visualization challenge in 2012, in all modules. It is a clean, moderate-sized

Figure 1 Connecting Modules throughout the Course



Note. Course content was reinforced through small exercises that relied on material from previous modules. An arrow from module A to module B indicates that module B relies on material from module A.

(550,000 trips) data set that includes geospatial, time-series, and demographic information. The continuity that arose from using one data set throughout the course, including in the optimization-focused modules, highlighted the various capabilities of the tools taught and how they might be used in tandem.

Capstone project. Finally, the course culminated with a two-part capstone project. This capstone project (detailed in §§3.7–3.8) drew on tools from each module and illustrated how they can be used in concert to formulate, solve, and deliver a high-quality solution to an OR problem. Our goal with the project was to contextualize these tools in the problem-solving process for students.

2.3. Essential Role of Feedback

A third aspect of our design philosophy was leveraging a cycle of continuous feedback from students—before the course began, during the course, and after the course.

For both iterations of the course, we performed a pre-course survey to identify new techniques students would most like to learn and determine the list of modules to be taught. Additionally, in the second iteration, we reviewed the previous year's feedback on which existing modules were most useful to students. These surveys were instrumental in choosing topics that were relevant to our student body and presenting those topics at an appropriate level of difficulty.

During the course, we solicited feedback on each individual module. To reduce the burden on students, we used Google Forms to distribute an anonymous, online course evaluation at the end of each session via hyperlink; this is a popular platform for collecting feedback for each lecture in a course (Gehringer and Cross 2010). Google provides basic, real-time analysis on these surveys. We used this feedback both to identify misconceptions from previous lectures that could be addressed with a short discussion at the beginning of the next module and to provide comments to instructors, which they could use to improve their own teaching style and techniques.

Finally, at the conclusion of the course we solicited feedback on the overall course structure and modules, including questions on how difficult each module was, which modules were most useful to students, and what other topics they wished were covered. We will use this feedback next year to help redesign the modules included in the course. See §3 for excerpts of student feedback at the module level and §5 for feedback at the course level.

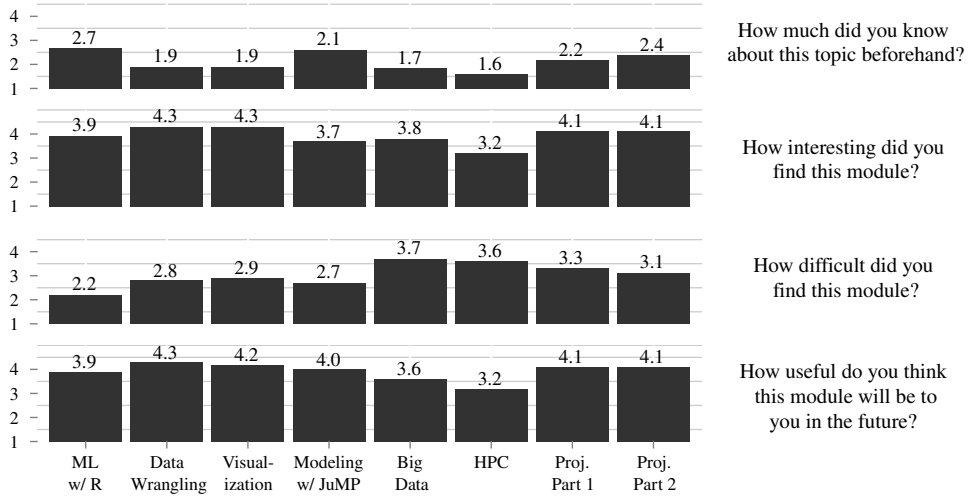
3. Course Modules

Our key course objective was to provide students with expertise in the wide range of advanced software tools for OR and analytics. In this section, we summarize the content of each module taught in the second year of the course and provide excerpts from the student evaluations for that module. Figure 2 summarizes student feedback for each module. The supplemental files (course_content.zip) provide more detailed information about the content of the modules, including slides and heavily commented code.

3.1. Machine Learning in R

Machine learning algorithms are used to detect patterns in and make predictions from data. These methods form a core part of how analytics practitioners use empirical data to build models. The goal of this module was to teach students how to run many common machine learning algorithms by using freely

Figure 2 **Feedback Collected at the Conclusion of Each Module**



Note. All responses were on a 1-to-5 scale (5 being the highest), and the numbers presented here are the average across respondents.

available packages for the statistical computing language R (folder IntroR within course_content.zip).

We first taught students how to read a comma-separated value file into R and how to use built-in functions to quickly extract summary statistics from a data set. We then taught them how to execute common algorithms such as linear and logistic regression, classification and regression trees, random forests, clustering, and support vector machines. We emphasized the importance of out-of-sample model evaluation and validation as well as how to calculate the coefficient of determination, interpret variable significance reports, generate confidence intervals, display confusion matrices for classification problems, and examine properties of clusters.

Most students had previously taken a machine learning or statistics course and therefore appreciated the module's focus on software instead of detailed descriptions of methods, with one commenting, "I like the fact that the class went fast and covered many common analytics tools without dwelling on explaining them." This was generally regarded as the easiest module, though all students found it interesting and a large majority thought it would be useful in their future.

3.2. Data Wrangling

Data Wrangling, the second module in R, focused on teaching students how to manipulate and reshape data sets to facilitate analysis (folder DataWrangling within course_content.zip). This critical and often frustrating step is usually one of the first parts of the modeling process. We chose to wait until the second module in the course to address these topics, however, as we wanted to first expose students to the variety of machine learning packages in R to contextualize *why* cleaning and reshaping data is important.

In this module we showed students how to identify and deal with outliers, handle missing data, and manage date or time information within a data set. Students built upon their knowledge of built-in R functions from the Machine Learning in R module by learning how to use them inside the `tapply` function, an R function that works like the pivot table operation in Excel. Students learned how to reorganize data contained in multiple files by merging data sets together on unique identifiers and learned about data reshaping through the split-apply-combine paradigm (Wickham 2011). The split-apply-combine framework takes a large problem and *splits* it into manageable pieces, *applies* some sort of operation on each piece independently, and *combines* the pieces back together (for an example exercise from the course, see Figure 3). Many students mentioned this framework of data analysis as a particular highlight, which suggests it is an essential component for future data wrangling classes.

This module was regarded by students on average as the most enjoyable, and was selected by no students as their easiest or hardest class (Table 1). All students thought it would be at least somewhat useful in their futures, and most felt that one class was long enough to cover the topic in sufficient depth. Some students expressed interest in understanding how these techniques scaled for larger data sets and how they related to databases, but we feel these topics were covered sufficiently in the later modules in the course.

3.3. Visualization

Visual representations of data and models can facilitate deep understanding at a glance. The goal of this module was to teach students how to take advantage of the power of visualization throughout the modeling process: to conduct initial exploration of a data set, to interpret a model, and to communicate with an audience (folder VisualizationR within course_content.zip).

Students learned how to create visualizations in R by using the `ggplot2` R package (Wickham 2009). We taught them to conduct data exploration by making scatterplots, plotting geospatial data on a map (see Figure 4 for an example exercise), making histograms, and making heat maps. Students practiced

Figure 3 In-Class Exercise from the Data Wrangling Module Covering Split-Apply-Combine

In-Class Exercise

Given the data frame `trips` containing all trips taken on Hubway bicycles:

1. *Split* the trips based on the bicycle used
2. *Apply* a function to each subset to extract the bicycle id, mean and standard deviation of trip durations, and total number of trips
3. *Combine* the results into a data frame with one row for each bicycle

Solution in R

```
spl <- split(trips, trips$bike.nr)
process.bike <- function(x) {
    bike.nr <- x$bike.nr[1]
    mean.duration <- mean(x$duration)
    sd.duration <- sd(x$duration)
    num.trips <- nrow(x)
    return(data.frame(bike.nr, mean.duration,
                      sd.duration, num.trips))
}
processed <- lapply(spl, process.bike)
bicycle.info <- do.call(rbind, processed)
```

Note. Pages 19–20 of script.pdf and exercise3_full.r, both within the DataWrangling folder of course_content.zip.

Table 1 Summary of Final Course Feedback with 28 Responses

	ML w/ R	Data Wrangling	Visualization	Modeling w/ JuMP	Big Data	HPC	Proj. Part 1	Proj. Part 2
Most enjoyable module?	0	29	25	11	7	0	14	11
Least enjoyable module?	0	0	0	14	11	61	4	4
Easiest module?	57	0	18	14	4	0	0	4
Hardest module?	0	0	0	11	21	29	14	18
Which module would you . . .								
extend?	4	11	32	14	11	0	14	7
remove?	7	4	0	7	11	54	7	7

Note. All numbers are percentages and the values may not sum to 100% as students could select "no module."

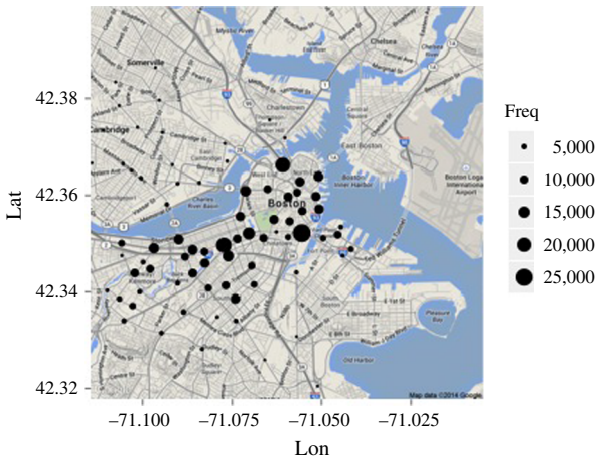
interpreting models visually by drawing regression lines and confidence bands, plotting convex hulls of clusters, and coloring a map based on predicted values. The use of visualization to convey uncertainty in model results was stressed. We presented principles of effective visualization, and we used examples of both good and bad visualizations to illustrate these concepts.

This content was new material for most students, and this was one of the most popular modules in the course (Table 1). Many students singled out the visualizations that employed maps as the highlight of the class, with one student stating that they, "…didn't realize it was so easy to make such a nice-looking visualization," and another simply saying, "maps are cool." The comparison of good and bad visualizations was polarizing—one student described it as "definitely appreciated," and another described it as "too sweeping and general." The breadth of possible topics that could be covered in this area was evident with one-third of the students selecting this class as the class they would most like to see extended—substantially more than any other class. The visualization of graphs, perhaps with `Graphviz` (Ellson et al. 2002), and tools to create interactive plots like `D3` (Bostock et al. 2011) and `Shiny` (RStudio and Inc. 2014) were consistently mentioned in the long-form text feedback.

3.4. Algebraic Modeling with JuMP

Optimization is a core skill in the OR and analytics toolboxes, and many options are available to model optimization problems, such as spreadsheets or raw matrix-based approaches such as MATLAB's `linprog` function. As these approaches do not scale well and are not appropriate for implementing advanced algorithmic approaches, we elected to use JuMP (Lubin and Dunning 2014), an algebraic modeling language (AML). JuMP is a package for Julia and was chosen as it is the only freely available AML that is both solver independent and supports advanced techniques such as callbacks for mixed-integer programming solvers. Although a number of both commercial and open-source AMLs would be suitable for the material in this module, in subsequent modules we took significant advantage of JuMP's advanced solver features and its being embedded in a general-purpose high-level language.

Figure 4 In-Class Exercise from the Visualization Module



Implementation in R

```
freq.start.table <- table(trips$start_station)
freq.start <- as.data.frame(freq.start.table)
names(freq.start) <- c("id", "freq")
freq.stations <- merge(stations, freq.start,
                       by = "id")
boston <- get_map(location = "boston",
                  zoom = 13)
ggmap(boston) + geom_point(data = freq.stations,
    aes(x = lon, y = lat, size = freq))
```

Note. Students were asked to plot Hubway stations on a map of Boston, with station size relative to the frequency of trips departing from that station. Lines 217–226 of script.R within the VisualizationR folder of course_content.zip.

The module was an introduction to both Julia as a programming language and JuMP as an AML for linear and mixed-integer programming (folder Modelling within course_content.zip). The class covered basic syntax, aimed toward students already familiar with languages such as MATLAB or Python. We asked students to implement a basic facility location model as an in-class exercise. As a more advanced exercise, we then asked students to enumerate the four best solutions to the facility location problem by iteratively solving the model and adding a constraint to cut off the optimal solution each time. Focusing on the simple-to-state but sufficiently complex facility location problem was motivational for many students, with one student summarizing it as, "challenging, but . . . it tied the whole class together."

Most students found this module relatively easy but expected the content of the module to be useful in their futures. Many students found the first third of the module that focused on teaching the Julia language better suited to a pre-class homework assignment. The concept of an AML was fairly novel for some students, as an alternative to, ". . . [creating] a large matrix and [keeping] track of the indices."

3.5. Big Data

Recent years have witnessed an increased prevalence of huge data sets in data analytics (Lohr 2012). This module was designed to highlight the differences in exploration and modeling for these large-scale problems (folder BigData within course_content.zip).

We reviewed fundamentals of data storage and proposed a taxonomy to distinguish between "small data" (fits in memory of a personal computer), "medium data" (cannot fit into memory completely, but can be filtered or partially loaded), and what is truly "big data" (may not even fit on a single computer). This taxonomy was generally described as empowering, with one student saying that they had "never known how to answer the 'how big is big' question [before]." We discussed techniques appropriate for the medium-data case with an emphasis on partial file loading and R packages for operating on data sets larger than memory capacity such as `biglm` (Lumley 2013). We then moved on to streaming and sketching algorithms used in big-data applications, such as reservoir sampling algorithms (see Figure 5). This small section was disproportionately popular in the feedback, with one student stating they had, "never thought about sampling in that way." Finally we introduced the concepts of the MapReduce (Dean and Ghemawat 2008) framework, a methodology for addressing very large distributed data sets that is similar to the split-apply-combine framework.

This module was described as one of the hardest modules of the course, and the majority of the students knew little about the topic beforehand. A relatively high number of students were not confident it would be useful to them in the future. The presentation of MapReduce as a methodology was well received but many found the implementation exercises too challenging. In contrast, the earlier exercises on performing operations on data sets iteratively were generally described positively.

3.6. High-Performance and Distributed Computing

It is valuable for students to be able to transform their theoretical understanding of algorithms into efficient, high-performance code. Additionally, because of the wider availability of academic computing clusters and computing as a commodity in the "cloud," we believe it is important for students to be aware of both the capabilities and limitations of such systems for solving large-scale OR and analytics problems. This module focused on both aspects of high-performance and distributed computing (folder DistribComputing within course_content.zip).

The first section of this module proceeded through a basic implementation of the classical gradient descent method, introducing students to important considerations in modern computing architectures, which are more often bound by slow memory access than by speed of computation. The BLAS (Dongarra

Figure 5 Example of a "Big Data Algorithm" from the Big Data Module That Will Scale Well for Very Large Lists or Operate on a Stream of Data

Challenge

Write code that samples one item from a list of unknown size, where each item is equally likely to be selected, and the list can only be traversed once.

Solution

Take the first item as the incumbent. For item $i = 2, \ldots,$ replace the incumbent with probability $1/i$. If we check for $n = 3$ we see that item 1 is selected with probability $(1 - \frac{1}{2})(1 - \frac{1}{3}) = \frac{1}{3}$ and item 2 is selected with probability $(\frac{1}{2})(1 - \frac{1}{3}) = \frac{1}{3}$.

Implementation in Julia

```
function sample(datastream)
    count_so_far = 0
    selected_item = nothing
    for item in datastream
        count_so_far += 1
        if rand() <= 1/count_so_far
            selected_item = item
        end
    end
    return selected_item
end
```

Note. Slides 25–28 of bigdataslides.pdf and exercise3_sol.jl, both within the BigData folder of course_content.zip.

et al. 1988) and LAPACK (Anderson et al. 1999) libraries for linear algebra operations were covered in this section, as they remain important for high-performance codes. The second section focused on distributed and parallel computing. We presented a case study of solving a large-scale stochastic programming problem in parallel by using Benders' decomposition (Benders 1962), with scenarios drawn from historical data in the Hubway data set.

This was expected to be one of the more demanding modules, and it was reported as the hardest module in the course. The step-by-step guide to improving the performance of the algorithm was popular, but many students found the distributed computing concepts difficult to grasp within the time frame and felt the compromises required to teach it made it feel "too hands off" and "too high level."

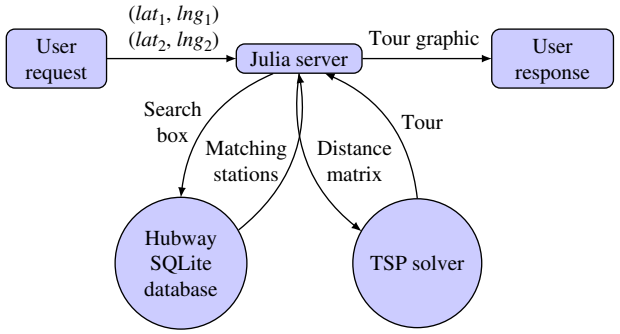
3.7. Project Part I—Internet & Databases

The capstone project linked together many of the data and optimization techniques learned throughout the entire course. The goal was to create an Internet service that solves a traveling salesman problem (TSP) on demand by accessing data that are requested from a database using the architecture depicted in Figure 6. It was developed over two modules: the first addressed the Internet and database aspects of the project, and the second covered the advanced techniques required to solve the TSP (folder ProjectPart1 within course_content.zip). We chose to address Internet services as they are a modern way to share data, models, and results. They eliminate the need for model deployment on individual computers, which is often a barrier to the implementation of OR and analytics solutions in practice.

The first topic covered was an overview of how the Internet is structured and how clients and servers interact. Students created a simple service that would return different messages based on the input and would solve simple arithmetic problems. The second topic was an introduction to databases, the main data storage tool in the industry. This introduction included both relational and "NoSQL"-style databases. Students learned the basics of using SQL to interact with a relational database, in this case `SQLite`. In-class exercises included joining the Hubway trip and station data sets together and filtering based on text patterns. Finally, we combined the two topics taught in this module to build an Internet service that would take all Hubway stations within user-specified latitude and longitude ranges and return the pairwise distances between them.

On average, students found the module interesting, and many students were surprised at how easy it was to create an Internet service. A substantial portion of the class was familiar with the structure of the Internet, especially those with computer science backgrounds. Many students had prior exposure to SQL or were familiar with the concept, but feedback indicated that in general they enjoyed the high-level review of the topic regardless of previous exposure.

Figure 6 **Design of the TSP Web Service That Students Implemented During the Capstone Project Modules**



3.8. Project Part II—Advanced MILP Techniques

The second project module focused on complementing the theoretical knowledge of combinatorial optimization learned in other courses (folder ProjectPart2 within course_content.zip). Although most students were familiar with the concept of branch and bound as a way to solve integer programs, we illustrated in detail the computational structure of modern mixed-integer linear programming (MILP) solvers and how users can interact with them by using callbacks.

We first presented a simple example to introduce using callbacks to enforce an implicit, or *lazy*, constraint in a model by checking if it is violated within a user-defined callback (see Figure 7). As a more complex example, we asked students to constrain a decision vector within a Euclidean ball. As the Euclidean ball can be represented as an infinite number of linear constraints, tangent approximations can be added dynamically (lazily) to enforce the constraint. We presented a standard formulation for the TSP, which has an exponential number of subtour elimination constraints. Students implemented this exponential family of constraints as lazy constraints. Finally, the TSP solver was integrated into the existing Internet service created in the previous module to complete the project.

The TSP is a classical problem in OR. Indeed, Lee and Raffensperger (2006) discussed teaching the methods to solve the TSP using AMPL scripts. In contrast to their work, where TSPs are modeled and solved in isolation, we used the TSP primarily as an example of a nontrivial optimization problem that can be integrated with databases and Web servers in a realistic, modern architectural design.

Figure 7 Example Used to Introduce Lazy Constraints to Students in JuMP

Simple Lazy IP:

$$\begin{aligned} \max \quad & x_1 + 2x_2 \\ \text{subject to:} \quad & x_1, x_2 \in \{0, 1\} \\ \text{lazy:} \quad & x_1 + x_2 \leq 1 \end{aligned}$$

Implementation in JuMP:

```
m = Model(solver=GurobiSolver())
@defVar(m, x[1:2], Bin)
@setObjective(m,Max, x[1] + 2x[2])
function lazy(cb)
    xVal = getValue(x)
    if xVal[1] + xVal[2] > 1 + 1e-4
        @addLazyConstraint(cb, x[1] + x[2] <= 1)
    end
end
setLazyCallback(m, lazy)
solve(m)
```

Notes. Slides 30–31 of projectPartII.pdf in the ProjectPart2 folder of course_content.zip. Users define a function that checks if the constraint $x_1 + x_2 \leq 1$ is violated by the current solution (up to a tolerance), and if so, adds this constraint to the model at runtime.

The second part of the project was regarded as one of the harder classes, but students derived satisfaction from bringing all the components of the course together, with one describing it as "a good capstone class." Students found the concept of MILP callbacks mostly understandable, and the escalation of complexity allowed them to appreciate the mechanics. The module also served as a refresher for the content of the Algebraic Modeling with JuMP module, as well as general programming practice. All students strongly agreed that the final project accomplished the goal of showing how to combine concepts and tools into a complete application, and 86% of the students were motivated by the topic of the project.

4. Lessons Learned

In addition to the specific feedback about modules detailed in §3, we learned several lessons during the second iteration of our course that will affect future iterations.

Workshop Environment. We employed active learning instead of a more traditional lecture format as we felt it to be particularly appropriate to the teaching of software tools. One potential problem with this workshop style is the increased demands on the instructor because of the higher degree of interaction between the instructor and students. We found that this was not an issue for our class sizes (approximately 40 students), especially as there was usually an additional instructor present to address technical issues and to help students who were lagging behind the rest of the class. We believe that the workshop format could scale to a larger number students if additional teaching assistants were used.

Distributing Course Content. The course was created as a series of loosely coupled modules, which enabled the independent and simultaneous development of all content. A consequence of this was that the course materials were frequently updated both before and even after the course began. To enable the instructors and students to stay in sync, we stored all course materials online using the distributed version control system (VCS) GitHub (GitHub 2014). The benefit of using a VCS over the Web interface of our university's course management software was that both instructors and students could easily synchronize their computer's copy of the materials with the master online copy. However, we found that many students were unfamiliar with the use of a VCS, requiring us to supplement our course materials with an extra tutorial on the topic. Collected feedback revealed that more than half of the students found this tutorial helpful, and we plan to make it available at the beginning of the course in the future.

Adjustments to Modules Offered. In future iterations of the course, we expect to remove the module on High Performance and Distributed Computing, as student feedback suggested that course participants did not have sufficient programming background to benefit from the material. Though we plan to solicit suggestions from students before launching the next iteration of the course to identify potential modules to add, student feedback suggests that a second module covering additional visualization topics would be well received.

5. Discussion and Conclusion

Both iterations of the course were well attended—in the second iteration 57 students were registered (38 for credit, 19 as auditors), and on average 38 students attended each module. Though most registered students were graduate students at the MIT Operations Research Center (73.7%), others included graduate students at various MIT engineering departments (7.0%), MIT undergraduates (3.5%), and visiting students and postdoctoral fellows (15.8%). A key goal of this course was to equip students with the skills they need to conduct research; all 28 students who responded to the end-of-class survey (partial summary in Table 1) reported that it did so. Given the popularity of the course and its success in equipping students with a range of advanced software tools for

OR and analytics, we believe it was effective and could be used as a template for software tools courses in other OR and analytics programs.

We believe that others will be able to replicate and modify our course content. This paper's supplemental materials contain the full set of course materials from the second iteration of our course, including slides and heavily commented code. The modular design of the course, detailed in §2.2, will enable others to easily retain some modules but adjust or replace others to meet their needs. The modularity also makes it simple to retain some software tools but to change others, an important point of flexibility as the choice of software tool is often based on an instructor's expertise. For instance, between the first and second iterations of our course we retained R as the statistical software package but updated all other modules to use Julia and JuMP. Instructors might choose to replace R with another statistical software package like SAS, Stata, or Python's pandas package (McKinney 2012), or they might choose to replace Julia and JuMP with Python-based interfaces for Gurobi (Gurobi Optimization 2014) or CPLEX (IBM 2013), for example.

The modular design of the course also creates flexibility in the course delivery. We delivered the course with one student instructor per module, but a software tools course could also be delivered by one or two instructors with broad expertise in statistical software tools and optimization tools. One promising mode of delivery would be as a massive open online course (MOOC). By leveraging prerecorded course content, a MOOC enables course delivery with a large number of instructors without needing them to all be available to teach at the same time. A downside of using a MOOC to deliver the course is that the active learning component of the course would be lost in this format.

Because our course was offered during MIT's Independent Activities Period, a four-week January term that features seminar-style courses, assessment was limited to a pre-class homework assignment for each module. In these assignments, we required students to install the software and add-on packages needed for the module. Students needed to validate their installation by running a small snippet of code and submitting the output for credit. For instance, the pre-homework assignment for the visualization module required students to install four R packages and to plot and submit a scatterplot, country map, and street map (README.md in the VisualizationR folder of course_content.zip). These assignments force students to address installation early, avoiding time-consuming troubleshooting during class time. A natural opportunity to expand assessment in the course would be assigning individual coding tasks following each module that require students to apply the tools from that module to a new data set or optimization problem. The course material is also well suited for a culminating project, in which individuals or groups of students complete projects involving machine learning and optimization.

Our second iteration of the course had 24 hours of class time, which is the ideal length for a university with a quarter system and roughly two-thirds of a semester-long course. However, a number of software tools could be added to the course to extend it to a full semester length. For example, when asked in the final course feedback to select one module to extend, approximately one-third of students selected visualization, indicating demand for more content on this topic. Other promising areas to add modules would be in simulation, tools for nonlinear optimization such as CVX (Grant and Boyd 2014), and specialized tools for MapReduce such as Hadoop (White 2009). Alternately, the course could be expanded to make it accessible to students who have less familiarity with machine learning or optimization than our participants. More emphasis could be placed on detailed descriptions of machine learning algorithms or on the mathematics of advanced optimization techniques such as column generation and lazy constraint generation. For participants with less programming experience, modules covering an introduction to programming could be added.

The need for education in OR and analytics software tools will only increase. Recently, dozens of business and engineering schools have created one-year Master of Science in Business Analytics or Data Science programs. These programs focus on training students to be effective OR and analytics practitioners, for which fluency in advanced software tools is essential. Such programs could especially benefit from a condensed OR and analytics software tools course because of their short duration and applied focus. We believe the materials accompanying this paper could serve as a template for the development of effective software tools courses in these and other programs.

Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/ited.2014.0131.

Acknowledgments

We wish to thank Ross Anderson, André Calmon, Virot Chiraphadhanakul, Velibor Mišić, and Allison O'Hair for their contributions to this course as student instructors. We also thank Dimitris Bertsimas for his role as faculty sponsor of the course. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship [Grant No. 1122374] (J. Kung and J. Silberholz). Any opinion, findings, and conclusions or recommendations expressed in this material are ours and do not necessarily reflect the views of the National Science Foundation.

M. Lubin was supported by the DOE Computational Science Graduate Fellowship [Grant No. DE-FG02-97ER25308].

References

Alpers A, Trotter LE (2009) Teaching computational discrete optimization at the undergraduate level. *INFORMS Trans. Ed.* 9(2):63–69.

Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, et al. (1999) *LAPACK Users' Guide*, 3rd ed. (SIAM, Philadelphia).

Behara RS, Davis MM (2010) Active learning projects in service operations management. *INFORMS Trans. Ed.* 11(1):20–28.

Benders JF (1962) Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4:238–252.

Bezanson J, Karpinski S, Shah VB, Edelman A (2012) Julia: A fast dynamic language for technical computing. CoRR, abs/1209.5145.

Bonwell CC, Eison JA (1991) Active learning: Creating excitement in the classroom. ASHEERIC Higher Education Report No. 1, George Washington University, Washington, DC.

Bostock M, Ogievetsky V, Heer J (2011) D3 data-driven documents. *IEEE Trans. Visualization Comput. Graphics* 17(12):2301–2309.

Bransford JD, Brown AL, Cocking RR, eds. (2000) Learning and transfer. *How People Learn: Brain, Mind, Experience, and School* (National Academy Press, Washington, DC), 51–78.

Dean J, Ghemawat S (2008) MapReduce: Simplified data processing on large clusters. *Comm. ACM* 51(1):107–113.

Devia N, Weber R (2012) Active learning exercise: Newspaper page layout. *INFORMS Trans. Ed.* 12(3):153–156.

Dongarra JJ, Croz JD, Hammarling S, Hanson RJ (1988) An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Software* 14(1):1–17.

Ellson J, Gansner E, Koutsofios L, North SC, Woodhull G (2002) Graphviz: Open source graph drawing tools. *Graph Drawing* (Springer, Berlin, Heidelberg), 483–484.

Gehringer EF, Cross WT (2010) A suite of Google services for daily course evaluation. *40th Frontiers Ed. Conf. (FIE)* (IEEE, Piscataway, NJ), F4C-1–F4C-2.

GitHub (2014) GitHub. http://github.com.

Grant MC, Boyd SP (2014) The CVX Users' Guide Release 2.0. http://cvxr.com/cvx/doc/CVX.pdf.

Gurobi Optimization, Inc. (2014) Gurobi optimizer reference manual. http://www.gurobi.com.

Hubway (2012) Hubway data visualization challenge. http://hubwaydatachallenge.org.

IBM (2013) Version 12.6: User's Manual for CPLEX, http://pic.dhe.ibm.com/infocenter/cosinfoc/v12r6/.

Kydd CT (2012) The effectiveness of using a Web-based applet to teach concepts of linear programming: An experiment in active learning. *INFORMS Trans. Ed.* 12(2):78–88.

Lee J, Raffensperger JF (2006) Using AMPL for teaching the TSP. *INFORMS Trans. Ed.* 7(1):37–69.

Lohr S (2012) The age of big data. *New York Times* (February 11). http://www.nytimes.com/2012/02/12/sunday-review/big-datas-impact-in-the-world.html.

Lubin M, Dunning I (2014) Computing in operations research using Julia. *INFORMS J. Comput.* Forthcoming.

Lumley T (2013) Biglm: Bounded memory linear and generalized linear models, R package version 0.9-1. http://CRAN.R-project.org/package=biglm.

McKinney W (2012) *Python for Data Analysis* (O'Reilly Media, Sebastopol, CA).

R Core Team (2014) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org/.

RStudio, Inc (2014) Shiny: Web Application Framework for R, R package version 0.9.1. http://CRAN.R-project.org/package=shiny.

Smith KA, Sheppard SD, Johnson DW, Johnson RT (2005) Pedagogies of engagement: Classroom-based practices. *J. Engrg. Ed.* 94(1):87–101.

Vars GF (1991) Integrated curriculum in historical perspective. *Educational Leadership* 49(2):14–15.

White T (2009) *Hadoop: The Definitive Guide* (O'Reilly Media, Sebastopol, CA).

Wickham H (2009) *ggplot2: Elegant Graphics for Data Analysis* (Springer, New York).

Wickham H (2011) The split-apply-combine strategy for data analysis. *J. Statist. Software* 40(1):1–29.