Introducing Computer Science in an Integrated Science Course

Barry Lawson
blawson@richmond.edu

Doug Szajda
dszajda@richmond.edu

Lewis Barnett
lbarnett@richmond.edu

Department of Mathematics and Computer Science
University of Richmond
Richmond, VA 23173-0001 USA

ABSTRACT

This paper describes our implementation and experience of incorporating computer science concepts into a team-taught, first-year interdisciplinary course for prospective science majors at the University of Richmond. The course integrates essential concepts from each of five STEM disciplines: biology, chemistry, computer science, mathematics, and physics. Including computer science in this course faces three primary challenges: few of the students have any CS background; the time devoted to CS instruction is reduced compared to a traditional introductory CS course; and the spirit of the course requires the CS material to be highly integrated with the other disciplines. Here we discuss our experience from three-plus years of offering the course and its impact on the major/minor pool of students in our own discipline.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: *computer science education, curriculum*

Keywords

CS1; Integrated Science

1. INTRODUCTION

At the University of Richmond, faculty from the disciplines of biology, chemistry, computer science, mathematics, and physics have designed an innovative year-long course with the intent of more fully integrating essential concepts from each of these science, technology, engineering, and mathematics (STEM) disciplines. The course, named Integrated Quantitative Science (IQS), spans two academic semesters, and each semester the course is team-taught by five faculty, one from each of the disciplines above.

The overall goals of this course include: (a) increasing understanding by students of essential disciplinary and interdisciplinary concepts; (b) increasing understanding by faculty of interdisciplinary concepts and connections; (c) increasing the ability of faculty to develop future courses that draw on and integrate material from multiple scientific disciplines; (d) increasing the number of undergraduates who participate in cross-disciplinary academic and research opportunities at the university and beyond; and (e) allowing faculty to bring new material, from a variety of disciplines, into existing courses within their own disciplines.

The course is open to first-year students only. Students apply for admission to the course, and preference is given to those students who meaningfully express a strong yet broad interest in the sciences and in scientific exploration. Student load for the course is equivalent to two courses per semester for each of the first two semesters. At the end of the two semesters, students will have received sufficient background to allow them to move into a major of their choice in any of the STEM disciplines listed above, and they receive general-education credit for the natural science and symbolic reasoning fields of study at our university.

To facilitate the necessary integration, we selected a unifying theme for each semester that allowed all five disciplines to be involved and that leveraged the expertise of the participating faculty. For the first semester, the current theme is antibiotic resistance; for the second semester, the current theme is cell signaling. As much as possible, each semester is structured so that no single discipline receives a large contiguous block of lecture time — the challenge is to balance a sense of integration with a sense of topic continuity.

In this paper, we focus on implementation details and our experience with this course from the computer science perspective, referring the interested reader to [6] for a more broad treatment of the course, constraints, and outcomes. When initially designing the IQS course, all parties involved felt strongly that computer science should be included. In addition to recognizing CS as an important field of study in its own right, there is clear recognition of the impact CS has in research areas of each of the other disciplines. Although none of the science majors at our university currently requires CS, many of our science colleagues understand the importance of strong CS skills for their own research and for their students. We see this course as an opportunity to present early the discipline of computer science to many science students who might not otherwise take a CS course (or might only take one very late in their course of study).

However, including CS does present challenges:

- Computer science is but one of five disciplines to receive coverage in this course. Our experience indicates that it is more difficult to present computer science effectively when, in the same course, the students' responsibilities and time are divided. Indeed, the total amount of time and the frequency at which the students focus on CS topics is significantly reduced compared to a traditional CS1 course.

- Few of the students who have taken the course have had any background in computer science. The same is not true for the other four disciplines. Therefore, while our science and mathematics colleagues can try to move quickly into more advanced material, we in CS must start at square zero[1].

- In the integrated spirit of the course, we feel it is very important to include applications and projects which are meaningful within the theme of each semester and, as much as possible, are highly integrated with other of the disciplines. The primary challenge in this regard is to choose topics that are sufficiently meaningful yet accessible by students new to CS.

2. IQS: CS INTEGRATION

Our IQS course is split across two motivating topics in consecutive semesters, and, especially from the CS viewpoint, there is an unfortunate gap that occurs with the winter break. Because most of our students are new to computer science—in terms of learning a programming language and thinking algorithmically—we believe the students benefit from frequent, smaller assignments, especially initially. The winter break presents a period of four weeks during which the students are unlikely to be actively working.

Accordingly, our approach is to introduce most of the traditional CS1 concepts within the framework of the first semester of IQS: basic computer organization, variables, assignment and expressions, control structures, loops, methods and parameter passing, fundamental object-oriented programming, arrays, and file I/O. These concepts are presented using projects and assignments that fit within the antibiotic-resistance theme of the first semester. Our language of choice is Java, to maintain consistency with our standard CS1 and CS2 courses.

We then use the second semester of IQS to reinforce those fundamental concepts through applications and assignments that relate to the second-semester theme of cell signaling. Students understand from the beginning that, because we are including five disciplines into the time equivalent of four courses, at the conclusion of the course they may need to pick up some small amount of additional material on their own before moving into subsequent courses within a particular discipline. So in the second semester, we also try to include any remaining CS1 concepts (more advanced OOP, recursion, searching, and sorting), but a full coverage of these remaining topics has not always appeared in the three offerings of this course.

2.1 First Semester Topics and Projects

The first semester of IQS currently focuses on the theme of antibiotic resistance. We use projects on agent-based simulation, Monte Carlo methods, and DNA sequence comparison to motivate the presentation of particular CS1 topics.

[1]As computer scientists, we feel compelled to start here rather than at square one.

Agent-based Simulation.

The first full CS project is an agent-based simulation model of antibiotic resistance. In this model, students implement methods which use strings of binary characters to model the binding sites of antigens, antibodies, and antibiotics [15]. Students also write a simple class to represent an agent in the model, its corresponding characteristics (e.g., spatial location, strings for binding sites), and its necessary behaviors (e.g., immune response, antigen mutation).

The CS1 concepts necessary for implementation of this project include variables, strings, conditional execution, for-loops, writing and using methods, and elementary OOP (using objects and writing a new class). We use roughly the first half of the first semester to introduce these topics in small, manageable chunks. For example, in the CS lessons during the first two weeks of class, we introduce variable declarations and assignments followed by strings and string methods through simple written homework and programming exercises. In the next two weeks of class, we introduce conditional execution and for-loops, again through manageable assignments similar to those in a typical CS1 course but with IQS-specific contexts. We then progress to writing a simple agent class, which the students later extend for use in the simulation model. Implementation of this first project is not due until slightly over halfway through the semester.

Admittedly, the entire agent-based model is not one that first-year students can reasonably complete on their own. We provide most of the framework for the interactive model, including code for a front-end GUI and the simulation engine. The students must implement an agent class which can then plug into the provided simulation framework.

Beyond writing the code for the agent class, we require the students to use their model once implementation is complete. We believe this is very important as the students see early the idea of CS as an area of study unto itself, not simply as a field of potential tools for the other sciences. Using the GUI, students are able to perform experiments by changing various parameter values (e.g., probability of infection, antibiotic string length) and evaluating the result on the number of infected, uninfected, and living agents over time. A screen shot of the GUI is provided in Figure 1.

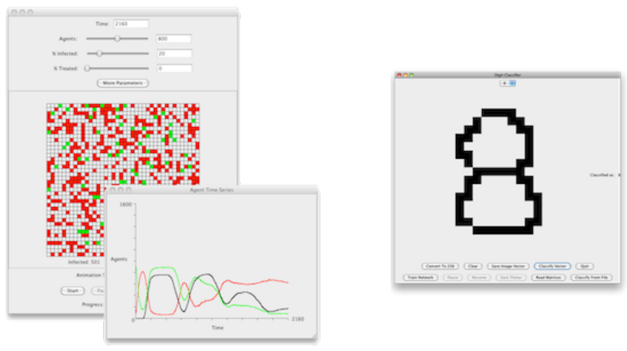

Figure 1: Agent-based simulation model GUI (L) and neural network character classifier GUI (R)

Monte Carlo Methods.

The next CS project involves a Monte Carlo approach for determining low energy structures of antibiotic molecules, integrating CS with chemistry and mathematics. Motivation for this project lies in determining stable (low energy) conformations of molecules, since they are more likely to occur in nature than higher energy conformations. Steric crowding (when atoms are closer to one another) generally leads to a higher energy state due to repulsion among the atoms. Hence, the energy of a molecular conformation is most affected by modifying dihedral angles within the molecule.

One approach for searching the energy state space is to generate an *ensemble* of different conformations of a given molecule by varying random dihedral angles, and then minimizing that ensemble to identify only the low energy conformations. Generating the ensemble is a natural application of the Monte Carlo technique. Given a molecular conformation, select at random a number of dihedral angles to vary, select at random the actual angles to vary, and then select at random the values for the angles. Repeat this process many times, adding each new conformation to the ensemble.

Again, we provide some framework for the students: classes for representing a conformation and an ensemble, and tools for reading and writing appropriate file formats. Students should be capable of writing the classes for a conformation and an ensemble, but we provide them because of time constraints. We use a straightforward file format (Z-matrix) for representing a conformation, providing an additional context for the chemistry professor to cover bond lengths, bond angles, and dihedral angles, and allowing the CS professor to introduce the algorithmic process of file I/O.

Integration with mathematics occurs via discussion of minimization techniques (e.g., gradient descent) during lecture. Students understand the mathematics behind minimization of the ensemble, but do not implement the minimization step within the CS project, again because of time constraints. Rather, we have the students use research-grade molecular modeling software to visualize the conformations generated by their own code, and then to conduct a full Monte-Carlo minimization process (generate and minimize an ensemble, and visualize the results).

There are no new CS1 concepts required for this project (requiring primarily looping, random number generation, and use of objects), providing a means of reinforcing ideas in a different context. In our allotted lesson times during this period, we cover data representation, low-level program execution, and the overall concept of Monte Carlo methods.

DNA Sequence Comparison.

The final project in the first semester integrates biology and computer science. Throughout the semester, students conduct a sequence of wet-lab experiments to sequence the 16S ribosomal DNA gene of bacterial symbionts isolated from marine sponges, specifically looking to identify novel antibiotic-resistant species of bacteria in the sponge cultures. Given the resulting DNA sequences, students use NCBI's BLAST utility [9] to identify good matches with DNA in other organisms. Working in groups of five, students write code that allows repeated searching of the BLAST results, which then permits the students to analyze those results for hypothesizing about antibiotic resistant bacteria in nature and potential therapeutic uses.

This project requires students to understand and implement basic file I/O, arrays, and searching. The students download BLAST results in XML format and then use a simple approach for reading necessary information (sequence accession number and species description) from the files. Students store all data using a multi-dimensional array of strings, and searches are performed using sequential search.

Additionally, in class we provide an introduction to algorithm analysis, including big-O notation and scalability. We present the concept of binary search (with the notion that data must be sorted) as an alternative to sequential search, but because there is not time to meaningfully cover sorting algorithms, students are not required to implement binary search. We also cover the Needleman-Wunsch algorithm [10] for performing global alignment of two DNA sequences, which provides an introduction to the dynamic programming technique. Students do not implement Needleman-Wunsch in code, but working with the algorithm by hand provides them an understanding of a technique similar to those used in the BLAST utility.

Final Poster Presentation.

A final requirement in the first semester is for each group to present a poster about their work. The poster, primarily driven by the marine sponge experiments and data analyses, must incorporate and integrate all five disciplines. Students are graded (a) as a group on poster content and integration, and (b) individually during a one-on-one oral presentation of the poster to a faculty member from the course. We also open the presentations to the university at large.

2.2 Second Semester Topics and Projects

The first semester of IQS includes at least some front-loading of required topics from all disciplines. The front-loading is particularly pronounced for CS, for reasons described earlier. This results in greater flexibility regarding selection of topics for the second semester of the course. For the computer science components, we try to choose topics that a computer science researcher might address, to counterbalance the focus on skills acquisition in the first semester, and to dispel any notion that computer science is "just programming".

We have offered a variety of second-semester topics and projects over the first three years of IQS, some of which are mentioned in Section 4. For the remainder of this section we discuss the topic, and its organization, that will be covered in the second semester of the current academic year.

Artificial Neural Networks.

During the coming spring semester, the CS component of IQS will focus on the theory and implementation of artificial neural networks, the first time that only a single research area will be covered in the second semester of IQS. In each of the previous three years, three or four different topics were covered. However, our experience is that even gifted first-year students respond better to advanced topics when those topics are introduced in small chunks. Moreover, discussing a single topic allows for a more detailed exploration, and one that potentially covers more advanced CS concepts. The use of a single overarching topic for the CS material also has the advantage of avoiding the startup costs of introducing several different topics. And because the performance of a neural network is well quantified, students see and understand the need for additional mechanisms that improve observed results. Note that, although we used this project as one of three during the previous offering of IQS, a full-semester version of the project will allow for more theoretical detail, more time for experimentation, and increased student understanding.

The students implement a neural network intended to classify the ten base-10 digits. A digit is represented as an image, which is then cropped, centered, scaled, and vectorized. The network contains perceptrons that use a logistic activation function, and the cost function is regularized. Optimal weight values for the network are computed using gradient descent and the back propagation algorithm. Correctness of back propagation implementations is verified using gradient checking, which is subsequently disabled.

Students are provided with a base GUI (see Figure 1), and are required to write methods to handle the corresponding events (e.g., outputting drawn digits into appropriate file format). Students are also supplied with default values for required parameters (such as number of hidden layers, regularization parameter, gradient descent step size, delta value for gradient checking, and stopping threshold), but are required (and encouraged) to experiment with the values of these parameters. A single training set with roughly 1000 training vectors is created by pooling a small number of drawn digits (in file format as generated by the GUI) from each student.

Few outside packages are used for the project. In particular, students code the perceptron class, the back propagation and gradient checking algorithms, the gradient descent algorithm, methods for reading matrices from and writing to files, and methods for vectorizing and preprocessing images. Basic matrix arithmetic operations are performed using the Jama Matrix package [3].

Not all of the neural network processing and checking is performed or discussed from the start. Rather, students experiment with methods for improving the performance of the initial version of the network. For example, they do very little preprocessing of the data at the outset, using raw pixel data, and attempt to implement the neural network and back propagation without performing any checking, other than final network output. When the neural network performs poorly (as it will at this stage), we discuss gradient checking, which the students then implement. This has the bonus of demonstrating that numerical approximation of partial derivatives for performing gradient descent is, in this context, generally infeasible. When back propagation implementations are verified, and network performance is still poor, we discuss image preprocessing steps, which students then implement. The final product is a neural network that classifies with a success rate of approximately 75-80%.

The theory of neural networks includes fundamental CS issues and integrates well with topics from mathematics and biology. We discuss machine learning, and in particular that machine learning methods often require the minimization of functions of several variables. This prompts a review of functions of several variables, partial derivatives, and gradient descent (covered in the first semester), as well as brief coverage of some improved minimization algorithms. The neural network concept is motivated by brief discussion of biological neurons, a topic which is subsequently covered in greater detail by the biologist teaching IQS. We also discuss the choice of a logistic activation function, as well as why, and how, this function classifies behavior that cannot be modeled in the absence of a non-linear transformation. Bias and variance are covered when discussing the concept of regularization. Finally, because the implementation is vectorized, the basics of matrix arithmetic are also discussed.

Among the desirable properties of this particular project is that the concepts can be handled by first-year students, and that when the project is completed, students have a tool that allows for further exploration, including testing additional preprocessing algorithms and comparing neural network configurations and parameters. Most important, however, is that students have been exposed to some of the types of "non-programming" questions considered by research computer scientists.

3. OUTCOMES

From the CS perspective, benefits to offering a course like IQS include the following:

- Early exposure to CS is beneficial for students. Our university has no CS requirement, and generally natural science students either do not take a CS course or wait until late in their course of study. IQS is one way to present CS early to alleviate these issues. Anecdotally, a few of our science colleagues have indicated IQS students in their research labs demonstrating coding skills and algorithmic thinking not present in their more traditional research students.

- A significant number of IQS students have gone on to take a subsequent course in CS, boosting our enrollments. Moreover, we have garnered additional CS majors and minors from among those students.

- Faculty development is an important component of IQS. Working in a highly interdisciplinary setting provides CS faculty with more applications from a variety of disciplines which we can bring into our own courses. For example, one of us has incorporated the topic of Monte Carlo conformational search into an upper-level Simulation elective.

- Because the course development process promotes conversation among STEM faculty, new research opportunities for faculty can be discovered. Anecdotally, one of us is currently working on an interdisciplinary research project that resulted from the initial IQS development.

To quantify the second bullet above, we have collected data about students from the first four years of IQS. Of the 78 students who have taken IQS, 19 have gone on to take at least one course at the CS2 level or above — nearly 25%. This is a larger percentage than we had initially expected, but certainly is welcomed. In the same time period, we have had 230 students in our traditional CS1 courses. Of those students, 86 have gone on to take at least one course at the CS2 level or above — roughly 37%. We are thrilled with a 25% retention from IQS for a group of students that, in general, we would not see (or see early) in our courses.

We polled those 19 students about their perceived influence of CS within IQS: 12 of them indicated that early exposure to CS in IQS was the primary reason they took additional CS courses; only 3 of that 12 indicated they may have taken a CS course anyway but later in their course of study. Of the 7 who, prior to IQS, already felt they wanted to pursue CS, all but one indicated that the CS experience in IQS firmly established that desire (the remaining student decided to pursue a non-STEM major). The absence of IQS would not have precluded these students from taking a traditional CS1 course, but these data support the notion that early exposure to CS is important.

Moreover, of the 19 IQS students who have taken at least one course at CS2 or above, 16 are eligible to be declared majors or minors. Of those 16 students, currently 6 are declared as CS majors and 1 as a CS minor. Based on discussions with last year's IQS students (now eligible to declare), we expect those to increase to 9 majors and 2 minors — a 14% yield of all IQS students. Finally, 6 of the 14 have conducted summer research in CS at the university. Admittedly, we cannot claim that IQS is the *sole* reason these students have taken an additional CS course or declared CS as a major, but we can claim IQS had a strong influence in that regard.

From our most recent report to the primary funding agency for development of the course, the Howard Hughes Medical Institute, we also have data from the students who applied to IQS during the inaugural year but were not accepted (54 students). From that control group, only 13% (7 of 54) took a CS1 course and only 2% (1 of 54) took a CS2 course. In contrast, 26% of IQS students (5 of 19) from the same year as the control group took a CS2 course; 24% of all IQS students (19 of 78) have taken a CS2 course (plus several from the current year expressing an interest to do so in a future semester). Again, these data support the need for early exposure to CS.

Development and delivery of this course is not without issues, however. The impact of IQS-required faculty resources on staffing within CS is a primary issue. In the absence of IQS, we in CS would likely have never seen most of these students in our traditional CS1 courses. However, we are essentially staffing two additional sections of a CS course per year, roughly 10% of our total course offerings, to serve 20 IQS students — a significant impact. The initial development year, in which two faculty obtained reassigned time specifically for planning and development, exacerbated this issue. In the absence of IQS, we could have offered additional upper-level electives or general-audience service courses. (While enrollment in our traditional CS1 courses grew during this period, the demand was not sufficient to require additional sections.) These faculty resource impacts, especially on small disciplines such as CS and physics, are a major issue that must be addressed, both for the sustainability of IQS at our university and for its ability to be adopted by other institutions. Our university has received a grant renewal from HHMI to address ways to make IQS sustainable and exportable, specifically in terms of the faculty resources required.

A significant issue with the current state of the course is the requirement that certain topics from each discipline be presented. Because this is a first-year course that counts for the first course in each of the five disciplines, buy-in from the five departments depended on including minimum content from each discipline (typically a subset of the standard-course minimum content). As a result, course development was constrained by the opposing goals of trying to present challenging interdisciplinary problems and the need to cover a specific set of disciplinary topics.

Another primary challenge of this course is to identify meaningful programming projects that are well integrated with the course yet pitched at the appropriate level for the students. It is not reasonable, for example, to expect students to be able to code their own worms and viruses, despite the many interesting ways in which the topic integrates with biology and mathematics. In the first semester, in which we cover the fundamental CS concepts, it is appropriate to have several different, smaller projects. In the second semester, in which we have more flexibility, a semester-long theme for projects seems more appropriate. For example, neural networks and genetic algorithms share the important aspects of integrating well with several disciplines, covering sufficient material for a full-semester theme, and allowing for interesting experimentation, while still being feasible for first-year students to implement. In either semester, providing to the students some initial code framework for particular projects mitigates the latter issue of feasible implementation.

4. OBSERVATIONS

The course content and its delivery have evolved each year we have offered IQS, and there are several lessons we have learned in the process.

- An objects-first approach to CS, which we tried in the first year, is difficult to execute successfully. This is due in no small part to the divided attention of the students and to the somewhat abbreviated nature of overall topic coverage. Moving to a procedural approach early, with plenty of simple exercises, works better in our experience with this course.
- Our expectations of the students in the course were too high initially. Because the students are accepted into IQS through an application process, we expected them to be able to handle an initial introduction to CS through carefully-crafted offline tutorials. This was not the case, and we had to request more lecture time. In hindsight, the mixture of student abilities we encounter in our CS1 courses is very similar to those in the IQS course. Moreover, we need to keep in mind that, based on the structure of this two-semester course, students in the second semester are "equivalent" to students in the second half of a one-semester CS1 course.
- It is critical in this setting for students to encounter CS on a frequent basis, particularly with regular, accessible coding exercises. Large blocks of time between CS assignments/lectures is not conducive to success.
- In the second semester of IQS, there are several integrated topics we used initially that in hindsight are not especially well-suited to the current version of this course. For example, topics in image processing related to the automation of two-dimensional gel electrophoresis present an interesting interdisciplinary research problem. However, the general problem itself is too advanced, and, as currently structured, involves too many individual stages to be feasible in a single semester first-year level course (e.g., requiring coverage of frequency domain processes and complex number theory). Other topics, such as implementing the Needleman-Wunsch sequence alignment algorithm or a Brownian motion simulator, suffer from the opposite problem: insufficient material to comprise a full-semester theme or to retain student interest over the course of a semester.

5. CONTEXT AND RELATED WORK

Similar efforts fall roughly into three categories. The first comprises courses intended to introduce science majors to computing skills required by their majors [1, 5, 8, 11, 14]. The second has been characterized as *contextualized CS*. Related courses in this category draw on the sciences or mathematics for organizing themes in introductory courses [2, 7]. The third category includes science courses that attempt to combine two or more disciplines in an integrated fashion.

The University of Richmond's IQS course falls squarely into the third category. There are several relevant examples from other institutions. The Accelerated Integrated Science Sequence at the Claremont Colleges [13] integrates biology, chemistry, and physics. Students are introduced to computer modeling with Matlab, but no computer science credit is awarded. The closest match for the Richmond course is Princeton's *Integrated Science* course [12], which served as a motivating example for the design of IQS. The course has been described in several articles in the *Daily Princetonian*, e.g., [16]. Both courses have a strong problem-solving orientation and present material from biology, chemistry, computer science, mathematics and physics. They differ primarily on distribution of emphasis among the disciplines, with the Princeton IS course providing "a mathematically and computationally sophisticated introduction to physics and chemistry, drawing on examples from biological systems" [12] (Freshman Year course description), while Richmond's IQS course provides students with preparation equivalent to the initial course in all five disciplines. The Richmond and Princeton courses are the only ones in this category that prepare students to move directly into CS2. The first year curriculum of Princeton's Integrated Science course takes as its organizing framework the set of quantitative skills and techniques needed for tackling big problems, particularly in the area of quantitative biology. Each of the techniques is developed through applications in the participating disciplines [4]. Richmond's course, on the other hand, is organized about a research theme for each semester, with the quantitative tools introduced as needed to develop the theme. The strategies are different, though the ultimate goals are similar.

6. CONCLUSIONS

Along with faculty from four other STEM disciplines, we have developed for prospective science students a first-year interdisciplinary course which has CS as a significant component. Challenges include students with generally no CS background, reduced presentation time compared to a traditional CS1 course, and the need to balance complex integrated problems with accessibility for students. Benefits include non-traditional students who proceed to additional courses and/or major in CS, the opportunity to present some advanced topics early, and faculty development. We are currently working to reduce staffing impacts and to make the course exportable. We will gladly provide materials and additional details to interested parties.

7. ACKNOWLEDGMENTS

The authors and their faculty colleagues gratefully acknowledge the support of the Howard Hughes Medical Institute, PKAL/KECK FIDL, the National Science Foundation, and the University of Richmond.

8. REFERENCES

[1] J. C. Adams and R. J. Pruim. Computing for STEM majors: enhancing non CS majors' computing skills. In *Proc. SIGCSE 2012*, pages 457–462, Raleigh, NC, March 2012.

[2] V. Barr. Create two, three, many courses: an experiment in contextualized introductory computer science. *J. Comput. Sci. Coll.*, 27(6):19–25, June 2012.

[3] R. F. Boisvert, J. Hicklin, B. Miller, C. Moler, R. Pozo, K. Remington, and P. Webb. JAMA: A Java Matrix Package. Retrieved Sep 5, 2012 from `http://math.nist.gov/javanumerics/jama/`.

[4] D. Botstein. An Integrated Science Curriculum at Princeton. *iBioMagazine*, June 2011. Retrieved Sep 3, 2012 from `http://ibiomagazine.org/issues/june-2011-issue/david-botstein.html`.

[5] Z. Dodds, R. Libeskind-Hadas, C. Alvarado, and G. Kuenning. Evaluating a breadth-first CS 1 for scientists. In *Proc. SIGCSE 2008*, pages 266 – 270, Portland, OR, March 12 – 15 2008.

[6] L. N. Gentile, L. Caudill, M. Fetea, A. Hill, K. Hoke, B. Lawson, O. Lipan, M. Kerckhove, C. Parish, K. Stenger, and D. Szajda. Challenging disciplinary boundaries in the first year: A new introductory integrated science course for STEM majors. *J. College Sci. Teaching*, 41(5):24 – 30, May 2012.

[7] M. Guzdial. A media computation course for non-majors. In *Proc. of ITiCSE '03*, pages 104–108, Thessaloniki, Greece, 2003.

[8] S. Hambrusch, C. Hoffmann, J. T. Korb, M. Haugan, and A. L. Hosking. A multidisciplinary approach toward computational thinking for science majors. In *Proc. SIGCSE 2009*, pages 183 – 187, Chattanooga, TN, March 2009.

[9] National Center for Biotechnology Information. BLAST: Basic Local Alignment Search Tool. Retrieved Sep 5, 2012 from `http://blast.ncbi.nlm.nih.gov/`.

[10] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Bio.*, 48:443–453, 1970.

[11] Princeton University Computer Science. Computer Science 126, April 1 2012. Retrieved Aug 17, 2012 from `http://www.cs.princeton.edu/~cos126`.

[12] Princeton University Lewis-Sigler Institute for Integrative Genomics. Integrated Science, March 1 2012. Retrieved Aug 17, 2012 from `http://www.princeton.edu/integratedscience/`.

[13] K. L. Purvis-Roberts, G. Edwalds-Gilbert, A. S. Landsberg, N. Copp, L. Ulsh, and D. E. Drew. Accelerated Integrated Science Sequence. *J. Chem. Ed.*, 86(11):1295 – 1299, Nov. 2009.

[14] R. Sedgwick and K. Wayne. *Introduction to Programming in Java: An Interdisciplinary Approach*. Addison-Wesley, 2008.

[15] P. Seiden and F. Celada. A model for simulating cognate recognition and response in the immune system. *J. Theor. Biol.*, 158:329–340, 1992.

[16] T. Thein. Integrated science pays off for graduates. *Daily Princetonian*, October 12 2009.