A first course in computing with applications to biology

Ran Libeskind-Hadas and Eliot Bush

Submitted: 6th December 2012; Received (in revised form): 16th January 2013

Abstract
We believe that undergraduate biology students must acquire a foundational background in computing including how to formulate a computational problem; develop an algorithmic solution; implement their solution in software and then test, document and use their code to explore biological phenomena. Moreover, by learning these skills in the first year, students acquire a powerful tool set that they can use and build on throughout their studies.

To address this need, we have developed a first-year undergraduate course that teaches students the foundations of computational thinking and programming in the context of problems in biology. This article describes the structure and content of the course and summarizes assessment data on both affective and learning outcomes.

Keywords: bioinformatics; education; computational biology

INTRODUCTION
We join a growing chorus of educators who believe that students in the biological sciences must be trained in the foundations of computing [1–5]. Our students should be able to formulate computational problems and understand the abilities, limitations and underlying techniques used in the software that they use to solve these problems. They must also be able to write their own programs because the growing number of computational problems in biology is rapidly outpacing the development of bioinformatics software; most biologists will confront computational problems for which there are no existing ‘out-of-the-box’ software solutions. Finally, irrespective of whether they ultimately write their own software, life scientists with a foundational computing background will be better equipped to have fruitful interactions with the bioinformaticians, software developers and other computational experts with whom they are likely to interact over the course of their careers.

In that spirit, we have developed a first-year undergraduate course at Harvey Mudd College. This ‘BioComp’ course teaches students ‘computational thinking’ and programming, motivated by compelling and current problems in biology [6]. The course is intended for undergraduates with no prior computing experience and only a high school level biology background.

This course was designed to be taught in 10 weeks. Because semesters at Harvey Mudd are 14 week long, our version of the course adds some additional content at the end that contributes to the college’s core curriculum but is not necessary for a course targeted at life science students. These extra topics include digital design and computer organization and introduction to ideas in the theory of computation.

By the end of this course, students should be proficient programmers and have the problem-solving skills required to formulate a computational problem, decompose that problem into logical sub-problems, develop simple data structures and algorithms to solve those sub-problems and test and document their resulting programs. In addition, students should be able to explain and use important
general computational techniques that are ubiquitous in biology including dynamic programming, randomization experiments, data visualization methods, among others.

In this article, we describe the organization and content of the BioComp course and some data on learning outcomes and student perceptions of the course.

BACKGROUND
Harvey Mudd College follows the 4 year liberal arts colleges model prevalent in undergraduate programs in the United States. In this model, all students take general education courses for breadth, including courses in the sciences, humanities and social sciences. Students typically declare their major field of study in their second year.

All students at Harvey Mudd are required to take a ‘CS 1’ computer science course in the first semester of their first year and a general biology course in the second semester of their first year as part of the College’s general education or ‘core’ curriculum.

In 2009, we developed an experimental year-long course that integrated CS 1 and the core biology course and offered this course to 24 students. While the course was well received by students, the integrated model was not sustainable owing to staffing constraints. Consequently, we used the valuable lessons learned from this experimental course to develop the one-semester BioComp course that covers the computer science material in CS 1 but with the motivating examples, applications and homework problems coming from biology.

Students may elect to take BioComp to satisfy the computer science requirement in lieu of CS 1.

Both CS 1 and BioComp present and exercise the same foundational computer science material including functions; data types; loops; recursion; object-oriented programming; dynamic programming/memoization; design, implementation and testing of ‘large’ programs and basic ideas in computational complexity and efficiency of algorithms.

The salient differences between CS 1 and BioComp are the motivating applications and homework problems. CS 1 presents a wide array of applications such as computer games, graphics, cryptography and data compression without privileging any application area and without attempting to weave a connecting theme among these applications. The underlying philosophy of CS 1 is that while no single application will necessarily be exciting to every student, every student will hopefully find at least some of the application areas to be compelling.

In contrast, enrollment in BioComp is by student request and thus the students who register for it are, presumably, interested in biology. BioComp presents and exercises the CS 1 computer science material via current problems in biology such as gene finding, sequence alignment, phylogenetic tree inference and folding of biological molecules.

Both BioComp and CS 1 use the Python programming language because of its relatively simple syntax (which allows us to spend more time on concepts and less time on syntactic details) and because of the many scientific packages available for Python that are likely to be useful to students in their future computing work.

STRUCTURE AND CONTENT
The BioComp course is organized as four 2 week modules, with two additional weeks at the end for students to work on capstone projects. Each module tells a biological ‘story’ with a corresponding computational problem. The programming concepts are presented ‘just-in-time’ and taught in the context of the motivating application. At the end of the first week of the module, students write short programs that develop their skills and contribute to the functionality for the somewhat larger program that they write at the end of the second week of the module.

Typically, the first lecture of the week provides biological background and foreshadows the computational challenges required to address the problem. The second lecture provides the computational tools and ideas required to solve the problem. At the end of the week, students attend a 2 h computer laboratory session where they begin working on the problem that week with support from the course staff. This accounts for approximately one-third of the problems for that week, and the students complete the rest of the work on their own time. By the end of each module, students have written their own programs that they then use to perform dry-lab computational explorations.

Our pilot offering of this course in 2009–2010 was a year-long course, which provided us with an opportunity to experiment with many different topics in biology. Based on that experience, we chose topics for the 10 week BioComp course that were representative of current uses of computing in biology, were particularly well received by the students
and allowed us to exercise all of the CS 1 computer science content.

Our 10 week course is organized into the four modules as described below.

The first module, entitled ‘Python versus Pathogens’, begins with the story of Typhoid Mary and focuses on pathogenicity in *Salmonella*, the organism that causes typhoid. The objective of this module is to use computational methods to identify and understand the function of pathogenic genes.

The first week begins with a discussion of the importance of gene finding followed by a quick review of DNA structure, genes, the central dogma and open reading frames. We introduce computer programming including data types (numbers, strings and lists), variables, ‘for’ loops and conditional statements. Students write programs to find the longest open reading frame in a DNA sequence.

The following week, we discuss pathogenicity islands and methods for identifying pathogenic genes by comparing genes in closely related organisms. Students also learn about additional looping constructs (e.g. ‘while’ loops), are exposed to top-down software design principles and write their own gene-finding program, ultimately locating putative genes in a provided pathogenicity island. Finally, they use BLAST to identify similar genes in other organisms and explore their function.

The next module is on the topic of sequence alignment, motivated by the problem of measuring the similarity between biological sequences. This topic is foreshadowed in the previous unit and we remind the students of the importance of aligning sequences in the process of measuring and identifying the similarities between genes. This unit also spans 2 weeks and culminates in students writing their own programs to align DNA and protein sequences. The primary computing idea imparted in this module is recursion. While the recursive alignment algorithm is slow, it forms the basis for faster algorithms, which are discussed in the fourth module.

The third module is on phylogenetics and is motivated by the problem of determining the evolutionary histories and relationships between species. In our experience, students are fascinated by the prospect of using their aligned genetic sequences (from the previous module) to infer phylogenetic trees. Students are exposed to basic ideas in phylogenetic inference and two different approaches to phylogenetic reconstruction: maximum parsimony and distance-based methods. Students implement a distance-based algorithm and use turtle graphics to render the phylogenetic trees.

The fourth module addresses the secondary and tertiary structure of biological molecules. In particular, we focus on the secondary structure of RNA and its importance in inferring the function of certain noncoding RNAs. Students learn about a simple model for RNA folding and then implement their own programs that find ‘optimal’ RNA foldings and render them using turtle graphics.

Students learn about the differences between exponential and polynomial time, the use of dynamic programming (memoization) to turn certain slow algorithms into fast ones and use this technique in their RNA folding programs as well as in ‘fixing’ their earlier sequence alignment programs to run fast. Finally, students are exposed to some other important ideas that arise in computational problems including intractability (i.e. NP-hardness) and the use of heuristics and meta-heuristics for dealing with computationally intractable problems. We describe some examples of such strategies that are used in computational biology software packages.

**COURSE PROJECTS**

At the end of the term, we offer students a choice of three or four projects. The learning objectives of these projects include the following:

- Students will design, implement and test a program that involves multiple interacting functions and data structures. The programs are typically ~200 lines long and students must think carefully about the design of the constituent components of their program.
- Students will learn a computational or mathematical technique that was not taught in the main part of the course in provide students with the confidence to learn a new idea and use it in their work. Examples include maximum likelihood, genetic algorithms and simulations of chemical reactions.
- Students will perform compelling and rich computational experiments with the software that they have developed.

After showcasing the projects in class, students choose one and work on it for 2 weeks. Here, we briefly describe a few examples of recent projects.
Project 1 asks students to simulate the evolution of organisms in an artificial life environment. An organism in this system consists of a set of enzymes, one for each reaction in the metabolism. The organism’s fitness is determined by providing it with a certain set of starting concentrations and observing how it is able to change those concentrations. The enzymes that make up an organism have various properties, and these vary from organism to organism. Depending on these characteristics, organisms will make more or less of various substances. Organisms are judged based on how much they make of two particular target molecules.

Project 2 involves implementing a maximum parsimony algorithm for phylogenetic tree reconstruction. While we have described the idea in the phylogenetics module, students were only required to implement a simpler distance-based algorithm. Students who choose this project are asked both to implement the maximum parsimony algorithm and then compare the solutions that it finds to those found by their distance-based algorithm using real data. In the process, students see that there are often different phylogenetic trees for the same data, and these differences are due to the underlying assumptions and computational methods.

Project 3, based on materials developed by Russell Schwartz at Carnegie Mellon University, involves the inference of gene regulatory networks based on microarray data [7]. We explain the concept of maximum likelihood and students implement an algorithm that constructs the maximum likelihood gene regulatory network from the given data. While a number of simplifying assumptions are made here to make the mathematics understandable and the programming appropriate in scope, students are exposed to important ideas that are used throughout computational biology.

Project 4 asks students to design, implement, and experiment with a genetic algorithm for artificially evolving programs to control a virtual robotic vacuum cleaner. The genetic algorithm maintains a population of robot’s, each with its own sequence of instructions for vacuuming a rectangular room. The student’s write an object-oriented Python program that iteratively constructs new generations of robots by measuring the fitness of each one (based on how well it vacuums a room in a given amount of time), mating pairs of robots with high fitness (by crossing over parts of each of their instruction sets), performing mutations, and repeating. The simulation begins with robots with randomly generated instruction sets (which do a very poor job on their vacuuming tasks) and ultimately generates populations of highly fit robots. Students exercise concepts in object-oriented programming and then experiment with the effects of varying selection and mutation parameters.

A summary of the 10 week course structure and content is given in Table 1.

<table>
<thead>
<tr>
<th>Module</th>
<th>Biology topics</th>
<th>Computing topics</th>
<th>Homework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python versus Pathogens (2 weeks)</td>
<td>DNA, central dogma, ORFs, gene finding, pathogenicity islands</td>
<td>Functions, variables, data types, for and while loops</td>
<td>Gene finding</td>
</tr>
<tr>
<td>Sequence alignment (2 weeks)</td>
<td>Measuring similarity of genomic sequences</td>
<td>Recursion</td>
<td></td>
</tr>
<tr>
<td>Phylogenetics (2 weeks)</td>
<td>Evolution and phylogenetic methods (distance-based, parsimony, likelihood)</td>
<td>Recursion on trees</td>
<td>Implementation of a distance-based phylogenetic tree algorithm</td>
</tr>
<tr>
<td>Primary, secondary and tertiary structure of biological molecules (2 weeks)</td>
<td>RNA folding and its significance</td>
<td>RNA folding algorithms, memoization (dynamic programming), computational complexity</td>
<td>Implementation of fast RNA folding algorithms</td>
</tr>
<tr>
<td>Course projects (2 weeks)</td>
<td>Topics as needed for projects</td>
<td>Topics as needed for projects</td>
<td>Choice of 3–4 projects</td>
</tr>
</tbody>
</table>

A first course in computing

COURSE DETAILS AND RESOURCES

Following the year-long experimental course in 2009–2010, the semester-long BioComp course has been taught at Harvey Mudd each year since 2010 with ~40 students per class.

The course meets twice per week for 75 min. While the course is taught in a lecture format, each
lecture has one or more 5–10 min pencil-and-paper exercises where students work in pairs on a problem posed by the instructor. For example, when discussing the gene-finding problem, we provide students with a DNA sequence and ask them to identify putative genes and explain how they found them. Before discussing phylogenetic tree algorithms, we provide students with a few short orthologous genomic sequences and ask students to construct, by hand, their best guess of the phylogenetic tree and explain their reasoning. A few students are then asked to briefly report back on their ideas and solutions, which engenders a short class discussion before the lecture resumes. These in-class exercises help keep students engaged, motivate the need for computational solutions and elicit questions and observations that might not have arisen otherwise.

In addition to the lectures, there is a 2 h computer lab at the end of each week. In the lab, students work on a computational problem with support from the instructors and undergraduate course assistants. The lab is a standard computer lab with desktop computers and empty tables where students can use their laptops if they prefer.

We have staffed the BioComp course with two faculty members—one a biologist and the other a computer scientist. In addition, several undergraduate students serve as course assistants, helping students in the labs, holding evening help sessions and grading the weekly homework submissions. In total, the course assistants allocate \( \frac{1}{30} \) person-hours per week to support the course with 40 students.

We encourage pair programming on approximately half of the assigned work each week. Our experience reaffirms that of many studies in the educational psychology literature, pair programming is a particularly effective mechanism for learning to program [8, 9]. Nonetheless, at least one problem each week is designated ‘individual only’ to ensure that every student develops the capacity to programming independently. Students can seek help from the instructors during office hours and the course tutors at the evening help sessions. Additionally, they can post questions on an electronic class bulletin board that is monitored by the entire course staff.

The student work is graded for correctness by automatic testing software, while undergraduate courses assistants provide detailed written feedback on the quality of the algorithm and software design based on guidelines provided by the instructors.

### ASSESSMENT AND OUTCOMES

BioComp has been well received by students, and our initial assessment data suggest that it is achieving the desired learning outcomes. This section summarizes the assessment data for this course.

#### Learning outcomes

Our goal is for BioComp students to be able to formulate a computational problem in biology, develop an algorithmic solution, implement their solution as software and then test, document and use their code.

For the purpose of this article, we assessed student success by scoring the student solutions on their final projects for correct functionality, quality of design, ability to communicate their computational approach in writing and ability to use their program for scientific exploration. By the time that the students embarked on the final project, they had completed all of the weekly assignments in the course and received detailed feedback on each one. Thus, by analyzing the final projects, we are able to explore the extent to which students have mastered the course concepts and assimilated the feedback on that work.

In Fall 2012, BioComp and CS 1 shared one course project: the aforementioned “Project 4” on genetic algorithms for robot control. 27 BioComp and 31 CS 1 students chose this project. Both groups of students were assessed using the same rubric by the same group of teaching assistants. The scoring rubric was 50% for correct functionality, 30% for good design, 10% for documentation, and 10% for a description of how the program was used to explore some phenomenon (in this case, the effects of changing program parameters such as mutation rate).

The BioComp students had an average score of 97.3% on this project (standard deviation of 8.1) and the CS 1 students had an average score of 94.0% (standard deviation of 8.6). These results strongly suggest that the BioComp students acquire at least the same level of proficiency in foundational computer science skills as their CS 1 counterparts.

#### Affective outcomes

Our institution’s formal end-of-course evaluations are required of all students in all courses: they provide useful insights into student perceptions of this course. On a Likert scale of 1 (low) to 7 (high), students were asked to indicate their agreement with the statements (i) *This course stimulated my interest in the material* and (ii) *I learned a lot*. The means from the Fall 2012 offering are summarized here for
BioComp, CS 1 and the mean over all computing courses at the college. In summary, these results suggest that BioComp fared better than the average computing course on these metrics and close to CS 1, which is among the most highly rated courses at the college and has benefitted from more than two decades of experience and refinement.

<table>
<thead>
<tr>
<th></th>
<th>BioComp</th>
<th>CS 1</th>
<th>All CS Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) Stimulated my interest</td>
<td>6.54</td>
<td>6.67</td>
<td>5.87 ($\mu$ = 0.80)</td>
</tr>
<tr>
<td>(ii) I learned a lot</td>
<td>6.61</td>
<td>6.66</td>
<td>6.06 ($\mu$ = 0.86)</td>
</tr>
</tbody>
</table>

**Choice of major**

Students who take BioComp are more likely to choose a bioscience major than their peer group with similar backgrounds and interests. Specifically, we compared the 63 students who took BioComp in 2009 and 2010 with the 38 students who requested to take BioComp but were unable to register due to enrollment caps in BioComp and thus took CS 1 instead. Both groups subsequently took the college’s required introductory biology course in the spring of the first year.

Approximately 27% of BioComp students ultimately chose to major in a bioscience field, whereas only 8% of the control group did so. This is notable because the BioComp students and the peer group both requested placement into BioComp, and thus their predisposition to the biosciences before taking the course should be the same. In general, 7% of all students at the college chose to major in biology during that period.

**Workload**

One potential concern is that the integration of two subjects in one course may result in more work for the students. Our findings suggest that this was not the case for this course. Students were asked on the course evaluation forms to estimate their average weekly work outside of class. In 2012, BioComp students reported working an average of 5.31 h per week outside of class, while CS 1 students reported working an average of 5.82 h. The average overall computing course was 8.17 h (SD, 4.15).

**Performance in CS 2**

As part of our assessment of computer science learning outcomes, we examined performance in the second computer science course (‘CS 2’), comparing the performance of students who completed BioComp and those who took the ‘traditional’ introductory computer science course CS 1.

Among the ~500 students matriculating to Harvey Mudd between 2009 and 2011 (all of whom took CS 1, BioComp or a more advanced computer science course to satisfy the college’s computer science requirement), 44.0% chose to take CS 2. In comparison, 49.2% of BioComp students chose to take CS 2. (The 2012 cohort is not included here because that group had not yet decided on a second computing course at the time of publication.) BioComp students had an average grade of 3.69 (on a scale of 0 to 4) in CS 2 as compared with 3.56 for all students. While the differences are not statistically significant, the results are encouraging and suggest that BioComp students obtain at least the same foundational background in computing as students in a regular first year computer science course.

**Gender diversity**

Relatively few women choose to study computer science in the United States and the same is true in many other countries. For example, a report by the US National Science Foundation showed that in 2009, 18% of computer science majors were women and that computer science had among the lowest proportion of women among all STEM (science, technology, engineering and mathematics) fields.

In contrast, 60% of biological science majors were women [10].

At Harvey Mudd, all students are required to take an introductory computer science course during their first term on campus and students can choose between BioComp and CS 1. While 47% of Harvey Mudd students are women, 65% of the students who enrolled in BioComp between 2010 and 2012 were women.

These findings lead us to believe that a course like BioComp may be particularly appealing to young women and, more generally, to students who might otherwise not take a computer science course.

**Student comments**

Students are asked for their comments about the course on the end-of-course evaluation forms. The overwhelming majority of student comments have been strongly enthusiastic. The following two comments are representative:

‘I came to Harvey Mudd knowing no CS [computer science]. This course was absolutely fantastic
in helping me begin to learn about programming. I love that we use Python—it’s easy to understand and yet can still do pretty cool stuff. I enjoyed using CS in real, applicable ways.

‘I thought the split between biology one day and programming the other worked well. This is because one day we grasped the concepts of what we were going to do and the next we learned the tools to carry out the programs.’

**USING BioComp AT OTHER INSTITUTIONS**

In this section, we discuss issues related to the adoption (or adaptation) of this course at other institutions.

**Target audience**

At Harvey Mudd, every student, regardless of his or her eventual major takes CS 1 or BioComp in the first semester. In the second semester, all students are required to take an introductory biology course.

One advantage of having all students take BioComp *before* taking the general biology course is that the biology course can include problems that require computational sophistication. Indeed, the general biology course is evolving to include an increasing amount of content and homework/lab problems that require students to write short programs to explore biological phenomena. In addition, many students tell us that they find opportunities to use their Python programming skills in a variety of other science and mathematics courses and to explore their own interests while taking BioComp.

On the other hand, most schools do not have a computer science requirement for all students and it is unlikely that a student in the biological sciences would take a course like BioComp in the first semester. Moreover, it is reasonable to expect that life science students with some biology background would benefit from taking a course like BioComp later in their studies. In fact, a small number of third and fourth year biology students from other nearby colleges have recently taken our BioComp course, and their background in biology seemed to provide them with a deep appreciation of the utility of the material taught in this course. The sample size is currently too small to compare the performance of the Harvey Mudd first year students and the more senior off-campus biology majors.

In summary, we see advantages both to teaching a computation course early as well as in teaching it to more advanced students. Based on our experience to date, it is not clear that one model is inherently better than the other.

**Class size**

In the first several offerings, we capped the enrollment at 40 students per class. However, the course materials are now sufficiently refined and stable that we anticipate teaching the course to a larger audience. The number of student course assistants will scale linearly with enrollment, but we see no other impediment to teaching the course to a larger group. Indeed, CS 1, from which BioComp has borrowed heavily, is frequently taught in sections of ≥70 students.

**Course materials**

Course materials including lecture slides and homeworks are available at https://www.cs.hmc.edu/twiki/bin/view/CS6/CS5GreenSite.

**CONCLUSIONS**

Based on our personal experiences in teaching this course and feedback from students, we believe that the BioComp model offers a compelling way to introduce computational thinking and programming to students with interests in the biosciences.

Our students take this course before they have declared their majors and many of them ultimately choose to major in fields other than biology or computer science. In the near future, the University of California at San Diego will offer BioComp to their first year biology majors. It will be interesting to determine the level of success with learning outcomes and the student affective outcomes for this student population.

We encourage other institutions to experiment with the BioComp course materials or develop their own materials and curricula for teaching computing concepts and programming to their undergraduates.

**Key points**

- Biology undergraduates should know how to program.
- Biology undergraduates should understand basic algorithmic paradigms.
- We describe a first-year undergraduate course in computing for bioscience students.
- This course provides at least the same sophistication in computational thinking and programming as an introductory computer science course, but in a context that is particularly compelling to life science students.
Acknowledgements
The authors wish to acknowledge Professors Zachary Dodds and Stephen Adolph who contributed to the design and initial offering of BioComp. In addition, Z. Dodds provided valuable assistance with the collection and synthesis of assessment data. We thank the many Harvey Mudd students who have provided us invaluable feedback that has resulted in the continuing improvements in the course and materials. Finally, we gratefully acknowledge support from HHMI and the National Science Foundation, under the CPATH grant 0939149, for the development of this course.

FUNDING
This work was funded in part by grant #52006301 from the Howard Hughes Medical Institute and grant DUE CCLI #0536173 from the U.S. National Science Foundation.

References