1

EDUCATING BIOLOGISTS IN THE 21ST CENTURY: BIOINFORMATICS SCIENTISTS VERSUS BIOINFORMATICS TECHNICIANS[1]

PAVEL PEVZNER

Department of Computer Science and Engineering, University of California, San Diego, CA, USA

For many years algorithms were taught exclusively to computer scientists, with relatively few students from other disciplines attending algorithm courses. A biology student in an algorithm class would be a surprising and unlikely (though not entirely unwelcome) guest in the 1990s. Things have changed; some biology students now take some sort of *Algorithms 101*. At the same time, curious computer science students often take *Genetics 101*.

Here comes an important question of how to teach bioinformatics in the 21st century. Will we teach bioinformatics to future biology students as a collection of cookbook-style recipes or as a computational science that first explain ideas and builds on applications afterward? This is particularly important at the time when bioinformatics courses may soon become *required* for all graduate biology students in leading universities. Not to mention that some universities have already started undergraduate bioinformatics programs, and discussions are underway about adding new computational courses to the standard undergraduate biology curriculum—a dramatic paradigm shift in biology education.

[1]Reprinted from *Bioinformatics* 20:2159–2161 (2004) with the permission of Oxford University Press.

Bioinformatics Algorithms: Techniques and Applications, Edited by Ion I. Măndoiu and Alexander Zelikovsky
Copyright © 2008 John Wiley & Sons, Inc.

Since bioinformatics is a computational science, a bioinformatics course should strive to present the principles and the ideas that drive an algorithm's design or explain the crux of a statistical approach, rather than to be a stamp collection of the algorithms and statistical techniques themselves. Many existing bioinformatics books and courses reduce bioinformatics to a compendium of computational protocols without even trying to explain the computational ideas that drove the development of bioinformatics in the past 30 years. Other books (written by computer scientists for computer scientists) try to explain bioinformatics ideas at the level that is well above the computational level of most biologists. These books often fail to connect the computational ideas and applications, thus reducing a biologist's motivation to invest time and effort into such a book. We feel that focusing on ideas has more intellectual value and represents a long-term investment: protocols change quickly, but the computational ideas don't seem to. However, the question of how to deliver these ideas to biologists remains an unsolved educational riddle.

Imagine Alice (a computer scientist), Bob (a biologist), and a chessboard with a lonely king in the lower right corner. Alice and Bob are bored one Sunday afternoon so they play the following game. In each turn, a player may either move a king one square to the left, one square up, or one square "north–west" along the diagonal. Slowly but surely, the king moves toward the upper left corner and the player who places the king to this square wins the game. Alice moves first.

It is not immediately clear what the winning strategy is. Does the first player (or the second) always have an advantage? Bob tries to analyze the game and applies a reductionist approach, and he first tries to find a strategy for the simpler game on a 2×2 board. He quickly sees that the second player (himself, in this case) wins in 2×2 game and decides to write the recipe for the "winning algorithm:"

> If Alice moves the king diagonally, I will move him diagonally and win. If Alice moves the king to the left, I will move him to the left as well. As a result, Alice's only choice will be to move the king up. Afterward, I will move the king up again and will win the game. The case when Alice moves the king up is symmetric.

Inspired by this analysis Bob makes a leap of faith: the second player (i.e., himself) wins in any $n \times n$ game. Of course, every hypothesis must be confirmed by experiment, so Bob plays a few rounds with Alice. He tries to come up with a simple recipe for the 3×3 game, but there are already a large number of different game sequences to consider. There is simply no hope of writing a recipe for the 8×8 game since the number of different strategies Alice can take is enormous.

Meanwhile, Alice does not lose hope of finding a winning strategy for the 3×3 game. Moreover, she understands that recipes written in the cookbook style that Bob uses will not help very much: recipe-style instructions are not a sufficiently expressive language for describing algorithms. Instead, she begins by drawing the following table that is filled by the symbols $\uparrow$, $\leftarrow$, $\nwarrow$, and $*$. The entry in position (i, j) (that is, the ith row and the jth column) describes the move that Alice will make in the $i \times j$ game. A $\leftarrow$ indicates that she should move the king to the left. A $\uparrow$ indicates that she should move the king up. A $\nwarrow$ indicates that she should move the king diagonally, and $*$

indicates that she should not bother playing the game because she will definitely lose against an opponent who has a clue.

	0	1	2	3	4	5	6	7	8
0		←	∗	←	∗	←	∗	←	∗
1	↑	↖	↑	↖	↑	↖	↑	↖	↑
2	∗	←	∗	←	∗	←	∗	←	∗
3	↑	↖	↑	↖	↑	↖	↑	↖	↑
4	∗	←	∗	←	∗	←	∗	←	∗
5	↑	↖	↑	↖	↑	↖	↑	↖	↑
6	∗	←	∗	←	∗	←	∗	←	∗
7	↑	↖	↑	↖	↑	↖	↑	↖	↑
8	∗	←	∗	←	∗	←	∗	←	∗

For example, if she is faced with the 3×3 game, she finds a $\nwarrow$ in the third row and third column, indicating that she should move the king diagonally. This makes Bob take the first move in a 2×2 game, which is marked with a $*$. No matter what he does, Alice wins using instructions in the table.

Impressed by the table, Bob learns how to use it to win the 8×8 game. However, Bob does not know how to construct a similar table for the 20×20 game. The problem is not that Bob is stupid (quite the opposite, a bit later he even figured out how to use the symmetry in this game, thus eliminating the need to memorize Alice's table) but that he has not studied algorithms. Even if Bob figured out the logic behind 20×20 game, a more general $20 \times 20 \times 20$ game on a three-dimensional chessboard would turn into an impossible conundrum for him since he never took *Algorithms 101*.

There are two things Bob could do to remedy this situation. First, he could take a class in algorithms to learn how to solve puzzle-like combinatorial problems. Second, he could memorize a suitably large table that Alice gives him and use that to play the game. Leading questions notwithstanding, what would you do as a biologist?

Of course, the answer we expect to hear is "Why in the world do I care about a game with a lonely king and two nerdy people? I'm interested in biology, and this game has nothing to do with me." This is not actually true: the chess game is, in fact, the ubiquitous *sequence alignment* problem in disguise. Although it is not immediately clear what DNA sequence alignment and our chess game have in common, the computational idea used to solve both problems is the same. The fact that Bob was not able to find the strategy for the game indicates that he does not understand how alignment algorithms work either. He might disagree if he uses alignment algorithms or BLAST on a daily basis, but we argue that since he failed to come up with a strategy, he will also fail when confronted with a new flavor of an alignment problem or a particularly complex bioinformatics analysis. More troubling to Bob, he may find it difficult to compete with the scads of new biologists and computer scientists who think algorithmically about biological problems.

Many biologists are comfortable using algorithms such as BLAST or GenScan without really understanding how the underlying algorithm works. This is not substantially different from a diligent robot following Alice's table, but it does have an important consequence. BLAST solves a particular problem only approximately and it has certain systematic weaknesses (we're not picking on BLAST here). Users that do not know how BLAST works might misapply the algorithm or misinterpret the results it returns (see Iyer et al. Quoderat demonstrandum? The mystery of experimental validation of apparently erroneous computational analyses of protein sequences. *Genome Biol.*, 2001, 2(12):RESEARCH0051). Biologists sometimes use bioinformatics tools simply as computational protocols in quite the same way that an uninformed mathematician might use experimental protocols without any background in biochemistry or molecular biology. In either case, important observations might be missed or incorrect conclusions drawn. Besides, intellectually interesting work can quickly become mere drudgery if one does not really understand it.

Many recent bioinformatics books cater to a protocol-centric pragmatic approach to bioinformatics. They focus on parameter settings, application-specific features, and other details without revealing the *computational ideas* behind the algorithms. This trend often follows the tradition of biology books to present material as a collection of facts and discoveries. In contrast, introductory books in algorithms and mathematics usually focus on ideas rather than on the details of computational recipes. In principle, one can imagine a calculus book teaching physicists and engineers how to take integrals *without* any attempt to explain *what is* integral. Although such a book is not that difficult to write, physicists and engineers somehow escaped this curse, probably because they understand that the recipe-based approach to science is doomed to fail. Biologists are less lucky and many biology departments now offer recipe-based bioinformatics courses without first sending their students to *Algorithms 101* and *Statistics 101*. Some of the students who take these classes get excited about bioinformatics and try to pursue a research career in bioinformatics. Many of them do not understand that, with a few exceptions, such courses prepare *bioinformatics technicians* rather than *bioinformatics scientists*.

Bioinformatics is often defined as "applications of computers in biology." In recent decades, biology has raised fascinating mathematical problems, and reducing bioinformatics to "applications of computers in biology" diminishes the rich intellectual content of bioinformatics. Bioinformatics has become a part of modern biology and often dictates new fashions, enables new approaches, and drives further biological developments. Simply using bioinformatics as a toolkit without understanding the main computational ideas is not very different than using a PCR kit without knowing how PCR works.

Bioinformatics has affected more than just biology: it has also had a profound impact on the computational sciences. Biology has rapidly become a large source for new algorithmic and statistical problems, and has arguably been the target for more algorithms than any of the other fundamental sciences. This link between computer science and biology has important educational implications that change the way we teach computational ideas to biologists, as well as how applied algorithms are taught to computer scientists.

Although modern biologists deal with algorithms on a daily basis, the language they use to describe an algorithm is very different: it is closer to the language used in a cookbook. Accordingly, some bioinformatics books are written in this familiar lingo as an effort to make biologists feel at home with different bioinformatics concepts. Some of such books often look like collections of somewhat involved pumpkin pie recipes that lack logic, clarity, and algorithmic culture. Unfortunately, attempts to present bioinformatics in the cookbook fashion are hindered by the fact that natural languages are not suitable for communicating algorithmic ideas more complex than the simplistic pumpkin pie recipe. We are afraid that biologists who are serious about bioinformatics have no choice but to learn the language of algorithms.

Needless to say, presenting computational ideas to biologists (who typically have limited computational background) is a difficult educational challenge. In fact, the difficulty of this task is one of the reasons why some biology departments have chosen the minimal resistance path of teaching the recipe-style bioinformatics. We argue that the best way to address this challenge is to introduce an additional *required* course *Algorithms and Statistics in Biology* in the undergraduate molecular biology curriculum. We envision it as a problem-driven course with all examples and problems being biology motivated. Computational curriculum of biologists is often limited to a year or less of *Calculus*. This tradition has remained unchanged in the past 30 years and was not affected by the recent computational revolution in biology. We are not picking on *Calculus* here but simply state that today algorithms and statistics play a somehow larger role in the everyday work of molecular biologists. Modern bioinformatics is a blend of algorithms and statistics (BLAST and GenScan are good examples), and it is important that this *Algorithms and Statistics in Biology* course is not reduced to *Algorithms 101* or *Statistics 101*. And, god forbid, it should not be reduced to *stamp collection of bioinformatics tools 101* as it is often done today.