Plant Systems Biology: Lessons from a Fruitful Collaboration

Systems biology is the biology of many interacting elements—genes, binding sites, pathways, and so on. Capturing the interactions correctly requires measurements under a variety of conditions and results in a model that explains the outputs in terms of those interactions. Computer scientists can help biologists design experiments, iterate on the results, and model interactions, but first these natural and artificial scientists must appreciate one another’s culture and skill sets. This can result in a close, friendly, and very productive collaboration.

PUZZLES VERSUS DATA

Computer scientists solve puzzles. They are happiest when they have found a provable algorithm to solve a problem with as little regard to the semantics of the data as possible. That is the power of the discipline: a sorting routine, a database system, or a time series analysis package each work in domains whose existence was unknown to the original programmers and algorithm designers. Although offering power, this analysis-rich but semantics-oblivious ideal to problem solving gives rise to misunderstanding at many levels.

For example, a colleague of mine came up with a very fast string comparison algorithm in the late 1980s. He presented his algorithm to biologists at the National Institutes of Health. He came back dejected and told me, “You know what they asked me. They asked me if I had a Fortran implementation!” The problem of programming was to him, well, a trivial consideration. He was surprised that the biologists weren’t overjoyed at seeing the cleverness of his technique and grateful for the speed it would give them. This was not arrogance. Had his algorithm been useful for, say, computer-aided design, the engineers would have given him his desired reaction. The biologists for their part were not interested in algorithmic beauty but practical effectiveness. Biological theories must be tested by real data after all, logical theories must be tested by real data after all, and computer scientists were used—to start with in crude form—from the very beginning of the project.

Many computer scientists would love it if they could design an algorithm, put its description on the Web or in a journal, and then have biologists discover it and implement it. The trouble is that algorithms invented this way rarely fit the needs of a practicing natural scientist. I know this from personal experience. Starting in the late 1980s, two students and I designed some of the fastest tree matching algorithms in the world. Linguists, compiler-writers, and e-commerce people liked it, but biologists never did. Our algorithms were optimized for ordered trees (where the order among sister nodes matters). This is almost never useful to biologists who care mostly about the parent-child relationships of phylogenetic relationships. Only after realizing this problem 3 years ago have we made new tools available.

Biologists care about data. They stare at it. They fret about outliers. Computer scientists, in contrast, never look at data. (Once the program is run, who cares where it comes down?) In my collaboration with Phil Benfey (Duke University, Durham, NC) and Ken Birnbaum (New York University, NY) on cis-element analysis, I would write a program, run it against some data, and send the results to Ken. When Ken put together the first draft of the paper, I pointed to part of the paper and asked “How did you find that binding site?” “You found it,” they replied, laughing. “It came from your program.” This different interest in data has its good and bad aspects. The good is that this diminishes the temptation to “fix” bad data in the program (e.g. our cis-element-finding algorithm never explicitly removed TATA boxes; they fell out because the algorithm deemed them irrelevant for statistical reasons; Birnbaum et al., 2001). The only bad is that the computer scientist gets laughed at. (In other words, there is nothing bad about it.)

The bottom line is that the two cultures of scientists should compromise in their working styles but not in their ethics. Computer scientists should not hope to work in isolation. To be relevant, they must build useful software that can evolve with a project. Biologists should recognize the value of general tools—provided those tools can specialize to the problems of their labs.

My most extensive collaboration so far has been with Gloria Coruzzi’s lab on the modeling of regulation of biochemical pathways (New York University, NY). We have developed generalizable tools to help explore a well-spaced subset of experimental spaces (“combinatorial design”) to iterate on the results of the experiments to design new ones (“pivot analysis”), and then to model the results (“Boolean circuit construction”; Shasha et al., 2001). In principle, the tools apply to any experimental discipline, but the tools were used—to start with in crude form—from the very beginning of the project.
THE SUCCESS OF THESE COLLABORATIONS SUGGESTS SEVERAL LESSONS

Meet Every Week

We meet over lunch or in a small conference room with cookies. Discussions and presentations in comfortable settings spark ideas and suggest problems to solve. For example, our particular use of combinatorial design arose from the evident impracticality of trying all possible input combinations in a large search space.

Build Prototypes

Tools find use if they are available when a problem manifests itself, not 2 years later. We write tools within a few days of the identification of a significant problem. They are used immediately. At first, the tool is so crude that only we can run it, but our biologist colleagues understand the output. Refinements (and program refactoring) occur over time. As I discovered in my research for a book, computer graphics pioneer Fred Brooks started a whole thread of computer graphics this way (Shasha and Lazere, 1995).

Teach One Another

Many computer scientists (and a remarkable number of biologists) found the terminological flood of high school biology to resemble force feeding more than learning. Fortunately, molecular biology is much simpler and more nearly algorithmic than classical biology, so there is a lot of common ground. Further, when biologists explain their view of the world, computer scientists soon realize that the biologists are using a knowledge representation. As computer scientists well know, one representation may improve on another. For example, many biologists use an arrow notation in which an element X either represses or induces element Z. This makes it impossible to express a fact such as that X induces Z only when Y is absent or Y induces Z only when X is absent. Such “exclusive or” relationships are easily expressed as Boolean circuits.

CONCLUSION. FROM IGNORANCE TO TOOLS

People work together for glory, for tenure, or for money. But creativity blossoms in an atmosphere of enjoyment and mutual respect. Biologists may not know at first exactly what they want from a computer scientist. Computer scientists may be lost in a discussion of biochemical pathways. But if each feels comfortable enough to ask questions at the risk of appearing ignorant, then fundamental patterns may emerge. From patterns come algorithms and tools. Tools, in turn, lead to more science. Glory, tenure, and money flow of their own accord. At least, that’s the theory.

Received December 23, 2002; returned for revision January 2, 2003; accepted January 2, 2003.

LITERATURE CITED


Dennis E. Shasha*
Courant Institute of Mathematical Sciences
New York University
New York, New York 10012