

Research Statement

Donald Yessick, Ph.D.

I define research as advancing my own knowledge as well as advancing the global body of computer science knowledge. Because I want to contribute to the wealth of knowledge in the discipline, I am actively engaged in several ongoing projects.

Following is a list of my major projects, with a quick assessment of each project's status. For fuller descriptions, see below.

1. Compilers: Article under way.
2. Bioinformatics — Contig Scaffolding: Long-running project with much code completed.
3. Graph Algorithms: Probably unsolvable but work is yielding valuable insights.
4. Visual Java and Student Learning: Articles drafted; classroom evaluation time (a semester or two) required.
5. Robotics: Immersed, though not on frontier.
6. Virtual Reality: Seeking applications.
7. Simulation: Seeking applications.
8. Finite Element Mesh Generation: Collaboration in progress.

In addition to the projects listed above, I am pursuing projects in complexity/HCI (article in draft stage), Web development (ongoing research) and security (limited to theoretical work; require equipment for next stage).

1. Compilers

Most research in compilers centers around code optimization. My original research in compilers described a compiler writing technique and rapid prototyping with null objects, culminating in a compiler writing tool/language called SLY. Compiler writing is a well-researched niche and in some respects my technique solved a problem that was "already solved." This closes the minds of some who examine my work, but they miss the bigger point: my technique simplified existing tools and aimed specifically at rapid prototyping of new languages using source-to-source translation. As such it built on top of existing tools. The tool, SLY, was not the primary research but an example of implementation.

Nonetheless SLY served as a focus for my compiler development. SLY is built over existing compiler writing tools and used Java as a development language and target language. Interesting at the time was that SLY was written in SLY and self-compiled, meaning the entire tool was developed using the tool itself. While this boot-strapping is parallel to compiler development of general purpose languages (GPLs), SLY reached a point at which the tool "contradicted" itself. After producing five or more targets, SLY was making the process more rather than less complex. While I do not believe this diminishes the original merits of the tool (rapid prototyping should, in general, require only one to three targets, which were greatly simplified using the tool) the next extension to the tool needs to resolve the complexity issue by separating the concerns of each generated target. I have developed a variety of designs to solve this problem but each has fallen short.

I want add modules to SLY, allowing a separation of concerns regarding target code. I have eschewed traditional means of handling this complexity because they put me back in the original space. Traditional methods use the visitor pattern to delegate code generation away from the abstract syntax tree and I hope to avoid the added level of delegation, which adds a layer of complexity and obfuscation. However, I have yet to find a way around this using traditional languages such as Java or C++. I have investigated implementing the code in C++, which would give access to multiple inheritance, but moving to C++ surrenders useful language features that exist only in Java. The tool's extensive use of Java features such as the object base class, efficient string concatenation via toString(), garbage collection, and reflection make a C++ implementation difficult. Likewise the extended abstraction model I seek is not possible in Java without multiple inheritance. The traditional idea of faking multiple inheritance using interfaces cannot solve the inheritance needs. Without true multiple inheritance I am not able to recombine multiple abstractions into a single unit.

I need a new model of inheritance, as yet unsupported by any language I am aware of, to reach the desired abstraction level. I call the feature absorption and I will describe it briefly here.

Absorption is a variant on cloning, aliasing, and inheritance. Most object-oriented languages feature inheritance and support copy construction or cloning as a method of creating an object as a duplicate of another object. To my knowledge, no language features a method by which a declaration can change the type of an existing object in a

type-safe way other than casting. While downcasting is relatively routine, upcasting often creates runtime errors. Casting from a smaller type to a larger type fails in all but the most trivial of cases. Consider the following code example assuming we have class structure such that class B extends class A:

```
A objectA = new A();  
B objectB = new B(objectA);
```

Under current models of object-oriented programming, the objectB is not an alias to objectA but will house a copy of the objectA. Hence a modification to objectA may not be reflected in objectB depending on the shallowness of the modification or cloning.

Under the absorption model, objectA would be allowed to become absorbed by objectB and conceptually they become aliases to the same object, albeit otherwise obeying normal inheritance rules.

Absorption is the abstraction model I believe is required to extend SLY to allow modules that could plug into each other seamlessly.

Attentive readers will argue that an alias could be assigned such that the behavior I refer to is reachable via an object reference such as "objectB.objectA.x" where x is the field of interest. I wish to avoid this because it will require that modules be ordered and will lead to deeper confusion with deeper references. I seek to avoid a construct along the lines of "objectD.objectC.objectB.objectA.x." and replace it with "objectD.x." Again the attentive reader may object that we can add a reference to objectD that alias objectA.x but consider "objectA.x = new Object();" or "objectD.x = new Object();" and the attributes fall out of synch. Under absorption the references remain tied. The problem arises when we build upon an object of type A then require passing it to another construct whose type will not be known beforehand, but will require absorption of the A object constructed in a previous step. All current methods for dealing with this problem add layers of complexity, callbacks, listeners, and errors.

The alert reader may have also considered this alternative using copy constructor semantics and a casting of the clone:

```
A objectA = new A();  
B objectB = new B(objectA);
```

objectA = (A) objectB;

This is very close to the desired effect, because destruction of the original objectA allows it to reference a true alias as desired to a cast form of objectB, but this construction is not a complete solution because we have lost the original object. While the difference is subtle, any additional aliases to the original objectA (suppose by objectC) are broken.

Also under compilers is a C-based stack machine language I developed for the compilers class. I have used it twice but in neither instance did I have any students choosing to complete the code generation phase of the compiler. The students have opted to accept a 75 percent project score or they have taken an alternate option of developing a language of their own design. Without student feedback I have not been able to fully evaluate this work.

2. Bioinformatics — Contig Scaffolding

Another project that I have been pursuing since 2004 is in bioinformatics: whole genome shotgun sequencing. The essence of my interest in this area stems from a paper by Gene Myers describing the "contig-scaffolding problem" and his heuristic solution. I do not take issue with his heuristic solution but he presented the original problem solved as presented as NP-hard and this strikes me as incorrect.

In 2004 I modeled the problem and through numeric simulations satisfied myself that my assertion is correct. Yet my numeric simulation is not yet proof of that assertion. Numerically simulating the model pointed out potential deficiencies in my proposed approach. First, my solution did not handle gaps well and second, numeric simulation did not model all of the complexities found in actual genome data. At the time, I had access to public databases containing whole genomes to further test my solution. In 2007 I repeated my simulations using human genome data. Again, my hypothesis was not disproved but I could not fully confirm my hypothesis either. The published data was not available in raw form — areas of high repeat had already been removed — so I was unable to confirm the algorithm in the face of repeats. The second problem was that while I could break the genome data into simulated contigs and scaffolds I could not certify or compare this to the actual problem presented in the literature, which lacked details required to model the simulation exactly. I found some indication of better data sets being available through the literature but in fact all named FTP sites were nonexistent or secured. A review of new literature led me in fall 2008 to a raw dataset of a small genome.

Over the December 2008 break I sought to finally find the proof I needed. The data secured contained a whole genome as well as all of the data used to produce the genome. The contig scaffolding problem can be described as having some number of DNA "islands," referred to as contigs. The contig islands are formed by identifying overlapping reads and chaining these together.

Because the data will also contain nonoverlapping regions and gaps, the contig scaffolding problem aims to obtain an order and orientation for the contigs. The orientation issue stems from the fact that a particular DNA island may come from either side of the DNA double helix. During shotgun sequencing DNA is broken into random pieces by vibrations, then sequenced by machines producing double-ended reads called mate pairs. The mate pairs have a known average length and are read at each end with a span of unsequenced DNA connecting the reads. The shotgun sequencing problem is to assemble these double-ended mated pair reads into a genome — not unlike solving a jigsaw puzzle except that this puzzle has a mirror image to add complexity. When overlaps have been discovered and placed the problem of contig scaffolding can begin. The goal is to find mate pairs with each end in a distinct contig. This gives you an estimated distance between the two contigs and can suggest the order as well.

Myers maps this problem to finding a Euler path, which is known to be NP-hard then proceeds to describe a heuristic method that solves the problem using a happiness metric to find the optimal placement of the contigs. I assert that Myers's algorithm performs well because the underlying graph problem is not NP-hard. Myers maps the graph problem to a Euler path. I wish to start by mapping to a different problem. Instead of a Euler path I want to map the contig scaffolding problem to all-pairs shortest-path problem which is not NP-hard. If the contig scaffolding graph is a complete graph, the contig ordering can be found by identifying the diameter of the graph. The diameter of a graph is the longest-shortest path. A graph's diameter can be found by computing all-pairs shortest-path and finding the maximal value. This requires cubic time. An improvement can be made if the first and last ending contigs can be identified and the graph is known to be complete. Under these conditions Dijkstra's algorithm will identify the contig order in quadratic time. The graph is likely to be sparse; thus, Dijkstra's algorithm can improve the result to near linear if the ends are known. Even if the ends are not known, the algorithm can still be applied until the end points are found. Repeat regions will appear as loops. Dijkstra's algorithm will still discover the end points and then any loop can be unwound using the same method and inserted into the known chain. The analogy I use as proof that this

method will succeed is to compare the contig scaffolding problem to the traveling salesman problem with the additional knowledge that all of the cities are on a single highway. Under this scenario the traveling salesman problem is not NP-complete as the graph diameter indicates the ideal route.

As mentioned I sought to complete this over the December break, but the dataset supplied contained the raw reads rather than the contig scaffold graph. Thus I had to assemble the contigs via overlaps before I could produce the graph. Finding and identifying overlapping reads are complicated by reads errors. DNA alignment is often performed using the longest common substring (LCS) algorithm and sequence alignment uses special alignment scores to account for common mutations. The LCS algorithm is essentially quadratic time and the overlapping process requires comparing all subsequences to all other subsequences.

I developed an algorithm that allowed me to find with high precision pairs of strings with a likely overlap. Using the algorithm I was able to build an initial set of contigs. The resulting contig scaffold graph however was not complete (thus I was not able to find a large diameter) so I need to relax my overlapping criteria to allow and build more or larger contigs. Development halted when in late December my laptop melted its motherboard while processing data for one of my projects. I was able to recover my datasets but the combination of robotics, classes, and computer repair have for the moment halted further development.

I have not studied the literature regarding the overlap detection and building of the initial contigs. My understanding is that a pairwise sequence alignment is done against all other pairs. The method I developed is very fast, does not require pairwise alignment to all other sequences and may lead to an improvement in this area as well as the contig scaffolding problem. This will require a literature review of the current methodology.

3. Graph Algorithms

For several years I have pursued interest in three related graph algorithms: vertex cover, independent set, and maximum clique. These algorithms are NP-hard/NP-complete and have linear mappings to each other such that solving any one of these solves the set. Most research into these problems finds a simple variation, or a fast heuristic approach, but heuristic approaches cannot promise optimal results and rarely can provide any degree of promise compared to an optimal solution. A notable exception is the vertex cover algorithm for which there exists a heuristic that promises to be no worse than a factor of

two from the optimal solution yet even this metric is meaningless when one considers that twice optimal may include every node.

My attention to this problem has concentrated on finding efficient brute force solutions. Many of these solutions find heuristically promising solutions first which allows pruning. My research in this area stemmed from a desire to have a means of measuring the value of a cubic heuristic algorithm for the vertex cover problem. Also of interest is discovering the types of graphs that are difficult to solve. It turns out that brute force algorithms can run very fast for graphs of a certain type (low or high density depending on the problem). I would like to measure whether a difficult graph problem for one problem maps to an easier graph of a companion problem. Perhaps a hard vertex cover is more easily solved as maximum clique.

Recently I've been working on a maximum clique algorithm that describes cliques using a regular expression. The algorithm is able to start with a subgraph and build larger and larger regular expressions by adding nodes to the graph. The algorithm has a linear characteristic regarding the graph nodes but the regular expression grows exponentially when the graph is randomized. Ordering the graph nodes by finding maximal cliques or independent sets delays the exponential explosion, allowing a much larger subgraph to be discovered. I have explored several methods of reordering on the fly when the exponential growth reaches threshold levels and the algorithm is able to solve moderately sized graphs very quickly compared to an enumerating brute force method. While this algorithm so far gives very promising results, the problems here are known NP-hard and I have no expectation of proving $P=NP$.

4. Visual Java and Student Learning

Teaching is why I went into this business. I have always enjoyed working with students one on one. On the whole, I find students to be less open to the idea of working one on one with faculty. I see students struggle from the beginning with programming and the complexity inherent in learning the basic concepts of programming. In 2007 I developed a program to help students recognize errors and debug data structures. I wrote a paper and submitted it. While this work was not accepted as submitted, its reviewers encouraged further work and provided excellent criticism and feedback, pointing to two solvable problems with my paper: (1) insufficient coverage of previous work in the area and (2) insufficient evaluative evidence. I am currently expanding the literature review portion of the paper. I agree with the reviewers that the paper needs more evaluative evidence; at the time of its writing I had only the result of using the product in a class of

six students during a summer course. I have not been the instructor for the class in question since receiving feedback to be able to produce additional evaluation but hope to be able to repeat the experiment this semester.

5. Robotics

My first week at Coastal during January 2005, the grandmother of a student from the Academy for the Arts, Science and Technology called to ask whether I could help her son with a programming project. This led to my first exposure to the FIRST Robotics Competition and to robotics in general. Prior to coming to Coastal I had little exposure to robotics and I am now considered the department expert, yet I know I still have much to learn. I have found the field of robotics to be both challenging and rewarding. I have led students through many programming challenges in robotics now and have had the opportunity to pursue some research ideas in the area as well.

The field of robotics has exploded the realm of domain specific languages (DSL) and is leading the way into new directions for embedded systems programming and programming tools in general. Graphic programming tools in robotics can be found in Lego robots, MSRS, PIC, Vex kits, LabVIEW and other platforms. I have implemented several algorithms popular in robotics, including simulations for wavefront and sphere motion planning. The simulations allow obstacle avoidance in a potentially dynamic environment and the user can select new target destinations at will. I have written a program that demonstrates through a step-by-step animation how a wavefront planner explores a map to find the shortest path using a queue data structure. This wavefront animator can find all paths or single best paths from custom maps.

I am currently working with the robotics team to develop a robot control system. Complexities of the challenges this year are two-fold. The robots are working on a surface with a small friction coefficient, not unlike ice. The control system must allow teleoperated control with a high degree of success and control. We are working on methods to maximize traction and reduce wheel slippage, analogous to antilock brakes and traction systems found in modern vehicles. The robot must also be capable of autonomous pursuit of other robots. The robot's autonomous actions will be controlled using data transmitted by a wireless camera. The robot must find and acquire a target robot, which will be moving, maneuver behind the target robot, and deposit playing pieces in a scoring bin attached to the target robot.

6. Virtual Reality

Virtual reality is a hot topic at the moment, and Coastal's computer science department has several programs and courses aimed at second life and other virtual environments. These programs allow users to build worlds using custom and expensive tools. I have been pursuing virtual environments from another angle: building a virtual environment from scratch. I do not aim to be as complete as the commercial platforms (no lighting, no shading) but my plan is to build a framework that can be presented to students at the 150 level. As a first application of this framework, last semester students in my CSCI 150 lab modeled a spinning cube flying into the distance. As computer scientists it is important that we understand the framework underneath the virtual world as well as the use of commercial tools. Students may go to work on either side.

7. Simulation

Closely related to virtual worlds are simulation environments. Simulation and modeling are constants in robotics and many other fields of engineering and design. While there exist many commercial CAD tools and simulation software such as MSRS, I am developing code for my own simulation environments. Building from bare bones may be reinventing the wheel but it also offers a unique insight into the creation of these tools. There is a need to understand not only the use of these tools but also their development.

Yet another motivation for developing these tools/environments comes from robotics, where one may need to maintain a data model of the robot's position and configuration. Complex robots require accurate data modeling of arms, appendages, and position within the terrain. A system of any complexity requires configuration tracking and this requires virtually the same techniques as those used for modeling and simulation (leaving out only shading and lighting). Data models for configuration allow motion and path planning of a robot.

I have completed a first generation of both a simulation and virtual reality system and would be happy to give a demonstration of the code.

8. Finite Element Mesh Generation

In this project I am collaborating with Nouredine Hannoun, a colleague in Algiers with whom I have corresponded for several years. In 2005–2006 I developed a prototype quadtree mesh generator in C++, and during 2006–2007 I advised Nouredine on the development of a mesh generator by his students using Java. We are currently considering a new mesh generator in Java for finite element analysis. During this work we will compare Java performance to Fortran both for ease of development and for

efficiency of the algorithms. Specifically we are building a generator capable of adaptive mesh refinement because many current mesh generation techniques exceed the computational abilities of most systems. Nouredine has suggested that we develop an adaptive mesh generator using a technique called circle packing, and I wish to apply concepts picked up in developing the robotic and virtual world simulations to map into a 3D mesh when we are ready.