Copyright

by

Kevin Marcus Grant

2016

The Report Committee for Kevin Marcus Grant
Certifies that this is the approved version of the following report:

Improving RNA Folding Prediction Algorithms with Enhanced Interactive Visualization Software

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor: _____
Mia K. Markey

Co-Supervisor: _____
Robin Gutell

Improving RNA Folding Prediction Algorithms with Enhanced Interactive Visualization Software

by

Kevin Marcus Grant, B.S.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August 2016

Acknowledgements

Dr. Robin Gutell and Dr. Mia K. Markey (University of Texas) for supervising this work and providing guidance during the project. I was also able to rely on sample data and software from the Gutell Lab.

Jamie Cannone from the Gutell Lab (University of Texas) for testing various prototypes and providing detailed comments and questions.

Students Vishal Patel and Vince Kim, for discussion on how their projects could integrate with "RNA HEAT".

The students who developed "RNA HEAT 1.0": Gurushyam Hariharan, James Zhang and Angie Li. The 2016 enhancements from this Master's Project were built on the program that they created in 2003.

Abstract

Improving RNA Folding Prediction Algorithms with Enhanced Interactive Visualization Software

Kevin Marcus Grant, MSE

The University of Texas at Austin, 2016

Supervisors: Mia K. Markey, Robin Gutell

Software improvements from this project will enable new algorithms for RNA folding prediction to be explored. Issues with capacity, extensibility, multi-tasking, usability, efficiency, accuracy and testing in the original program have been addressed, and the corresponding software architecture changes are discussed herein. Previously limited to just hundreds of helices, the software can now display and manipulate million-helix RNAs. Actions on large data sets are now feasible, such as continuous zooming. A new scripting interface adds flexibility and is especially useful for repetitive tasks and software testing. Structural analysis of RNA can be streamlined using the new mechanisms for organizing experiments, running other programs and displaying results (helices, or arbitrary text and images such as statistics). Finally, usability has been enhanced with more documentation, controls and settings.

Table of Contents

List of Tables

List of Figures

Introduction

This project addressed a series of software problems: capacity, extensibility, multi-tasking, usability, efficiency, accuracy, and testing. The target was a research program that illustrates the natural formation of helices in ribonucleic acid sequences (RNA), a process known as "folding". Researchers use the software to help develop folding-prediction algorithms, assess the accuracy of algorithms, and see how folding occurs. The problem space is vast: for instance, visualization and processing power may be needed to identify on the order of 100 correct helices out of millions of potential helices, and one may wish to see the "neighborhood" of any helix. Several different statistics can also be gathered from this data and ideally displayed alongside everything else. Any software limitations in RNA research affect the breadth of samples that can be considered and the depth of the analysis that can be done.

The types of problems mentioned above were addressed at multiple levels: software architecture, documentation, text-based scripting interfaces, graphical user interfaces, and the entire environment for running programs and organizing data.

This report gives a brief history of the original version of the RNA Helix Elimination and Acquisition Tool "RNA HEAT 1.0", and the motivations for improving that tool with "version 2.0". Details on improvements and new software designs are included, as well as quantitative results. A series of recommendations are then made to guide future development. Finally, the program source code and all remaining issues and to-do items are captured in its online repository.

Background and Motivations

This project builds on previous development of "RNA HEAT", the "RNA Helix Elimination and Acquisition Tool" [2].

The original version "RNA HEAT 1.0" includes several features that are beneficial to RNA folding research. It supports a 2D helix display, a series of helix filters (constraints) on that display, the ability to select and inspect the properties of a helix, and a way to save the display as an image. Helix data comes from the BPSEQ file format [3] developed at the Gutell Lab.

There were several reasons to make changes to RNA HEAT.

Increased capacity became a goal because the first edition of this software could not view every RNA sequence that is available in the Gutell Lab. For example, the first version of the program runs out of memory on large samples, not handling sequences with thousands of base-pairs and a million helices.

The goal of extensibility emerged when considering how difficult it is to easily reproduce previous results. This applies both to research use, and to maintenance of the software. This is commonly achieved through a scripting language, and interfaces that are bound to the key functions of the main program.

Extensibility also comes in the form of support for new file formats. Research programs can be more integrated if more RNA information is communicated between them. Thus, a goal is to be able to process more data than before (more than BPSEQ).

Multi-tasking is a natural goal; ideally, everything related to some current experiment is easy to access in one place. A visualization tool has limited research applications if it cannot integrate with other software. Without direct ties between programs, the details of a region of interest would have to be manually transcribed. Consider, a researcher scrolls to one region and finds something to explore further: he or she would need to capture all of the details about that space and then separately run a program with those details as parameters. This takes time and effort, and risks introducing errors. The task of manually collecting results from programs is equally cumbersome.

After regular use of the original "RNA HEAT" program, it was also evident that some tasks were difficult to perform primarily because of the way they were presented. There was no readily-available documentation, no support for keyboard short-cuts, and common tasks like zooming did not have intuitive controls. Time permitting, any enhancements to usability would clearly be worth the effort.

Efficiency became a goal after observing sluggish zoom, scroll and filter performance in the original tool: requests could take literally many seconds to respond. That amount of waiting discourages researchers from fully exploring an RNA. Any new version would need to have more interactive response times: immediate in most cases, with delays being acceptable for only the largest or most intricate calculations.

Another form of inefficiency is the time required to perform repetitive tasks. Why, for instance, should a researcher or his or her colleagues have to open a file and apply the same parameterized filters over and over? The software could be enhanced to automate these steps.

"Sharing" is a goal that depends on many of the other goals above: given a tool that is high-capacity, fast and extensible with multi-tasking, it is helpful to add features that aid sharing among research labs and in classes. For instance, mechanisms to overlay and animate multiple results would be useful for illustrating not just what final result was achieved but how that final result emerged.

Finally, some problems were found in the original "RNA HEAT 1.0" that created a desire for further development. Inaccurate placement of reference helices and inaccurate helix descriptions made the RNA display more difficult to understand and correlate to other data sources. All issues (large and small) were logged in the code repository [1] and they served as strong motivations for this project.

Software Modifications

CAPACITY

A very high priority change was to make it practical to manipulate more RNA samples. In "RNA HEAT 1.0", zoomed helices did not display for most of the sequences from the Gutell Lab, including the largest ones like "d.23.b.A.calcoaceticus".

The original software rendered the 2D RNA display by creating an entire "picture" of the RNA display in memory. Each action that changed the appearance of the display, such as zooming in or out, would cause new pictures to be constructed with the appropriate dimensions. Memory requirements were directly proportional to the selected zoom level and the square of the number of base-pairs in the RNA! Since the display represented all pixels and not just helix locations, sparse displays were penalized just as much as dense displays; filtering had no significant effect on memory. The Java Runtime Environment would run out of memory quickly in most experiments, meaning that large RNA files could not be zoomed in far enough to inspect any helices.

Capacity is now much better, after some non-trivial changes to the software architecture. The first change was to translate data into a base coordinate system and scale vectors instead of trying to "blow up" the raw pixels of a base image. This approach produces a very high-quality rendering, especially when zoomed, as shown in Figure 1. It has also made the code easier to read: rather than having to understand a number of seemingly-arbitrary scaling factors and offsets throughout, now the drawings use the native coordinates of RNA data. Zoom and scroll are performed respectively by scaling and translation in a transformation matrix that applies to the graphics layer; this

means that an unchanging command like “draw line from (1, 1) to (2, 2)” will automatically appear at the right size and position in the window at all times.

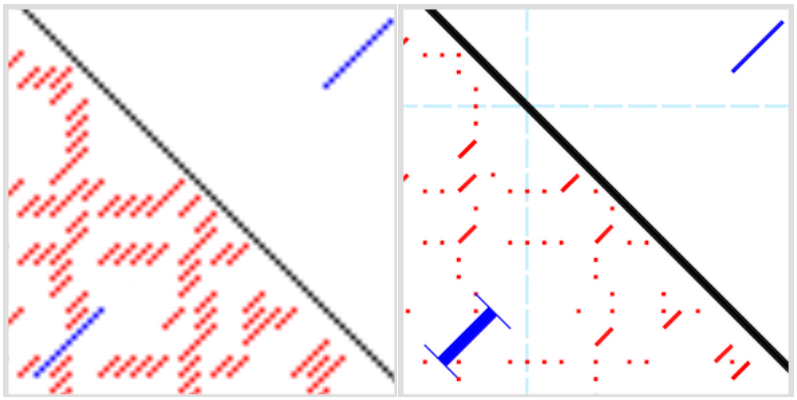

Figure 1: RNA Display: Old Zoom (v1.0) and New Zoom (This Project)

Each line represents a helix. Those above the diagonal are “actual” helices found from the input RNA sequence. Those below the diagonal are “predicted” helices, found by applying constraints such as base-pairs or range limits on the properties of helices. If a predicted helix matches an actual helix, the two lines will mirror on the diagonal. Any helix that has been clicked with the mouse is displayed in a different color. The left picture shows the rendering in the original program, where the pixels of zoomed-in helix lines are enlarged blocks. The right picture shows the program after the enhancements from this project: zoomed lines are smooth instead of pixelated, the real length of each helix is more apparent (especially for the length of 1), the selected helix is distinct from any other line, and aids such as grid lines are also rendered.

The second architecture change was to switch from a “construct image” approach to a “direct render” approach, meaning that the RNA would only be drawn when necessary and the drawing would occur directly in the target component of the graphical interface. This saved a substantial amount of memory, not only when first drawing the display but whenever the display was changed in any way. The graphical interface

component architecture makes even more optimizations possible, and these will be discussed later under Future Work.

Note that although images are no longer created during rendering, they are still needed to support the existing "save" feature. Fortunately, most of the code can be shared: both an image and a graphical interface component can rely on the same "graphics context" concept. Rendering code was modified to produce two different results from the same drawing commands: when the context was backed by an image the result was an image file, and when the context was backed by a window the result was an RNA display window.

The improvements to capacity have been measured in a very simple way: are the available RNA samples usable?

Table 1 lists a series of available RNA sequences that were tried with the original RNA HEAT program, and the number of helices that needed to be displayed after applying the default base-pair filter. With version 1.0 of the program, nearly all of the sequences were impossible to manipulate when zoomed: the runtime environment ran out of memory! And since helices cannot be clicked when zoomed-out, RNA sequences that failed to zoom were effectively broken, as there was no way to inspect their helices.

Conversely, the new version of the program displays all of these RNA sequences, zooms in to any level, and allows any helix to be inspected at any level. It also continues to function after the display has been refreshed by changes to zoom levels and constraints.

File	Base-Pairs	Helices	Old Zoom	New Zoom
b.5.b.E.coli	120	1711	OK	OK
d.5.b.A.tumefaciens	120	1691	OK	OK
d.16.b.E.coli	1542	276752	FAIL	OK
d.16.b.T.pallidum.A	1542	276752	FAIL	OK
d.23.b.A.calcoaceticus	2903	989449	FAIL	OK
d.23.b.E.coli	2904	980801	FAIL	OK

Table 1: Display of 10x Zoom (Helices Using C-G, A-U, G-U)

Each row describes a different RNA sequence, with its own input file. The first three columns show the properties of the RNA, to give a sense of scale: the number of base-pairs determines the length of each side of the two-dimensional square display in RNA HEAT, and the number of helices determines how many line segments need to be rendered. The last two columns show the success or failure of the zoom operation with the old version and the new version of the program, respectively. This shows that all RNA files can now be zoomed (and manipulated). Previously, only the smallest could be viewed and manipulated.

EXTENSIBILITY

RNA HEAT now provides a scripting interface.

Scripting has benefits for both the users and maintainers of a program. Users can make tasks repeatable and share those runs easily with peers. New ideas can be explored to some extent without having to change the main program. Software developers can use the same scripting hooks to write tests, store preferences, or perform any other task that would be inconvenient without those hooks.

The first extensions include basic commands: opening RNA files, opening helix annotation files, and running other scripts and programs.

Every constraint/filter is also a scripting command now, complete with configuration parameters. A base-pair filter has a list of two-letter identifiers to specify the pairs of interest. A helix-length filter has maximum and minimum numbers. Other constraints have a similar feel.

A series of constraints can therefore be applied automatically just by running a script. The graphical interface has been modified to take advantage of this, showing "constraint history" simply as a list of scripting commands. The user can Copy and Paste from that list, or look at the history save-file (which is also a script). This shows that the user does not necessarily have to understand the entire scripting language in order to take advantage of it; constraints can be set up interactively at first, seeing the effects immediately, and the setup can be captured in a script without ever having to write the code.

Although not used right now, some scripting interfaces have been added in anticipation of greater needs. For example, it is possible to iterate over all "actual" helices and inspect their properties, and it is possible to separately iterate over "predicted" helices (what would appear in the lower-left triangle of the 2D display). It is also possible to request any currently-selected helix. Scripts can also force results into multiple experiment directories instead of using one current experiment.

The original Preferences scheme was also converted into a script (previously it was an opaque binary format). One benefit is that users can directly modify settings in a text editor, if they prefer, without opening RNA HEAT; and, like any other use of a script, it is possible to copy settings between different people. Another big advantage is that backward-compatibility is much easier to maintain as preferences evolve. Even during this project, as new settings were added, it was trivial to read in old settings and store new ones.

Later, there will be a discussion of automated testing and the key role that the scripting interface now plays in all test cases.

Building the scripting interface was a complex task but also surprisingly straightforward.

As of 2016 there are many established ways to bind scripting languages to compiled programs. The scripting interface did not require a full parser for a language;

rather, the task was to design the interface, and ensure that the program exposed any features that scripts would need.

The main factor in choosing a scripting language was time. Ideally, scripting would be done early so that it could serve as the foundation for other tasks, and the entire project needed to be done in a few months anyway. Another factor was familiarity; since other research scripts at the Gutell Lab were already written in Python, it would be helpful for RNA HEAT scripts to have the same feel.

The JavaScript language was ultimately chosen because it could be adopted quickly and because it was easy to use. RNA HEAT is written in Java, and the Java environment has an entire JavaScript engine built into it! Even better, the syntax of JavaScript is very similar to Python (for common operations, it looks identical).

Scripts are simplified by using special interfaces for common operations. For instance, there are interfaces to log messages as information, warnings or errors. And, there is a utility for constructing file paths in a platform-independent way (anticipating that scripts may need to be shared across environments, such as Windows and Linux where paths look different). Technically, the JavaScript engine may already have these abilities but accessing them would require significant code and knowledge of the environment; for instance, it may require importing software libraries from the script in some complex way, and the code may be significantly more verbose. It was decided that straightforward scripting is a great feature, even if it creates small redundancies.

When the program runs from the command line, the user can now list any number of scripts as arguments and those scripts will be run automatically, in order. This gives great flexibility, as external processes can launch RNA HEAT for a particular purpose and not require user interaction.

In fact, although RNA HEAT is normally a graphical user interface, the command line can be used to prevent a window from opening at all. This might be very useful if a research program is trying to leverage non-graphical abilities of RNA HEAT, such as a command to open a particular RNA and run a program to dump statistics to a file, all without ever opening a window. A no-graphics mode can also provide a work-around for problems in the graphical environment, such as failing to load very large data sets. Or, it may simply be faster to not use a graphical interface, if it is easier to describe a task using a script than it would be to perform the task from a graphical display.

In order to support a non-graphical mode, the program code was divided cleanly into a "core" layer that makes no assumptions about having a window, and the remainder (which requires a window). Scripting commands are designed to detect a lack of graphical interface and behave appropriately. Some commands are not available without a window because they do things that naturally require user interaction. Most commands will do as much as they can and just skip any steps that would trigger an update to a display.

MULTI-TASKING

Research is performed using a variety of software. Each program may have a very different focus, accept different types of input and produce results in its own way. Moving data between programs is inconvenient and error-prone. Ideally, one can accomplish multiple tasks by starting from a single interface, with good integration between steps, viewing results in different forms.

Originally RNA HEAT was designed to display the helices of an RNA. The rules for which helices to display were determined entirely by RNA HEAT. What about other research programs that had special capabilities, like finding all possible helices? What about other algorithms? What if a program wanted to gather statistics and display a graph? In order to integrate with another program, the researcher would need to manually remember the details of a selected RNA and set of helices, run other programs separately, and view results separately. This should be a lot easier!

In this project, the tool was extended to allow arbitrary programs to be run, as often as desired (and even from scripts). Programs are given access to the currently-visible RNA as input. Any output messages and data files are organized for the user, and may even be displayed automatically.

The protocol for inter-program communication is quite straightforward: it is file-based. The user may set a preference for the root of all output. When RNA HEAT starts, a subdirectory is created that includes the current date and time, for uniqueness and organization of experiments. When a program runs, the latest experiment space is chosen as the current directory, and in it a copy of the user's original input RNA is placed with a

predictable name. The program may then produce any amount of output. Printed messages are captured to a log file automatically. New output files that follow a naming convention are displayed automatically by RNA HEAT when the program has finished. These special files include plain text, images, updated RNA files or helix annotations. Using these few simple rules, virtually any research program can have its results displayed by RNA HEAT! An example of multiple views is in Figure 2.

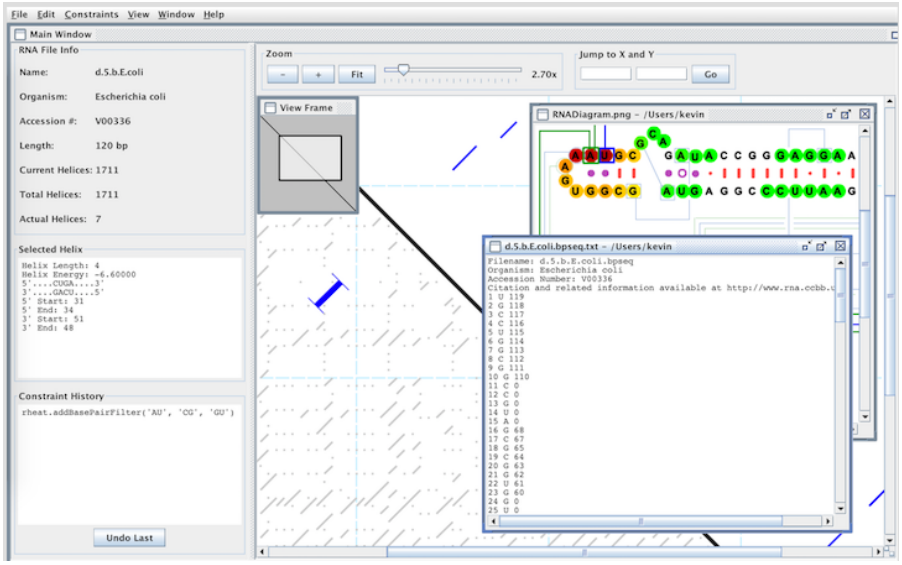

Figure 2: RNA HEAT v2.0 with RNA, Text and Image Files Open

This shows that the user may now display arbitrary text and image files in separate windows. It also shows several other RNA HEAT enhancements from this project. First, the higher-quality rendering (also shown in Figure 1). The small square "View Frame" window appears when zooming, to show the user which part of the complete RNA sequence is currently being displayed. At the top of the window, there are new zoom controls: the user can zoom continuously using the slider, or change by discrete amounts with "-" and "+"; also, a "Fit" button shows as much detail as possible. Scrolling controls have improved: the "Go" button lets the user enter one or both of the X and Y coordinates to view that location, and in the lower-right corner of the window (between the scroll bars) there are buttons for moving diagonally. Finally, Constraint History on the left side now uses scripting commands, supporting text selection and Copy for easy export to scripts.

The ability to produce standalone results is quite powerful but it became clear that partial, data-dependent results would be even more valuable. In particular, instead of having to correctly generate an entire replacement RNA, what if programs could just annotate the original helices? This is actually more flexible because the annotations can provide new information, and even give multiple labels to the same helix.

An immediate use for annotation would be to show changes over time. If multiple annotated helices occupy the same space, and were given some common labels, they could be recognized as different stages of the same final helix.

When a file in the new annotation format is read, the RNA display uses special line-widths and user-customizable colors to make each annotation obvious. Since annotations may overlap, there is a way to hide or show subsets. In addition, when clicking on a helix, all of its annotations are visible.

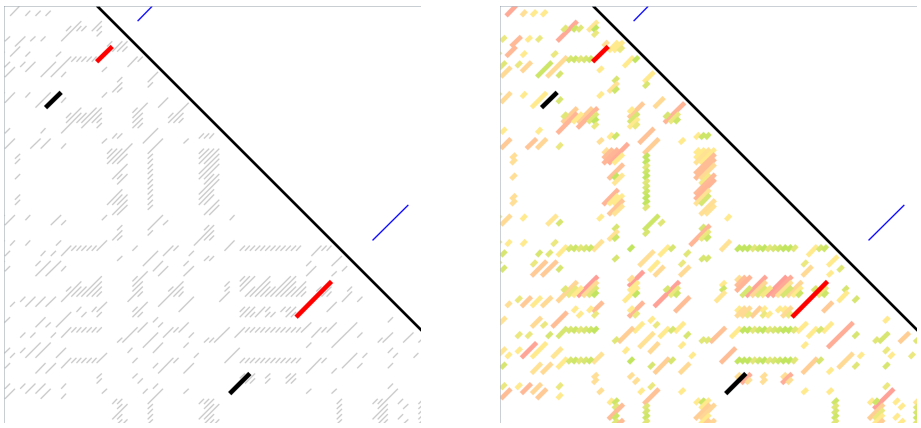

Figure 3: RNA HEAT v2.0 with Annotations and Energy Spectrum

This shows part of the display for an RNA sample. In the first case (left side) the user has applied annotations, where each annotation has its own color. Here, "primary initiator" helices are marked as red and "primary elongation" helices are black; this allows the researcher to quickly identify helices that are related. In the second case (right side) the user has added an energy constraint, creating a color spectrum that indicates the energy of each helix at a glance. For example, in this case the spectrum is between the energy levels of -10 and -0.01, with anchor colors of pink and green; therefore, helices that appear green are close to -0.01 in energy, and helices that appear orange may be in the initial quartile (perhaps -5). This allows researchers to quickly understand how energy values are distributed, without having to click on each helix to see its energy value.

Note that files in any supported format can be opened manually at any time, whether or not they were produced by running another program. This can be useful for a variety of reasons; for example, while viewing an RNA, the researcher may want to open a separate window with a text document or reference data file, or a graph with some statistics. It may also be that some data, such as helix annotations, was generated manually.

USABILITY

When software is used every day, a focus on usability can have a big impact. Some changes just make the program more pleasant to use. Others speed up research work significantly by reducing the chance of mistakes, making common operations easier to achieve, and reducing the time required for new team members to become productive.

In this project, the biggest improvements to usability depended on other changes, such as faster display rendering and scripting. Other improvements were small but easy to do in the available time, such as adding keyboard short-cuts for all commands and improving help content.

The zoom mechanism was a priority because it is frequently used, crucial for inspecting individual helices, and in RNA HEAT 1.0 it was neither efficient nor reliable. In version 1.0, only discrete zoom levels were available (hidden behind a pop-up menu) and display updates could require seconds of waiting. Worse, since the program could completely fail to zoom for some data sets, there was an element of frustration after requesting a zoom and seeing nothing happen.

RNA HEAT now provides a much more natural zoom interface: a slider that allows both continuous and discrete zooming, two keyboard-accessible buttons that zoom in or out, and a crucial zoom-to-fit feature that makes the RNA display as large as possible without requiring scrolling. The new RNA display renders so quickly that the user can use the mouse to drag the zoom slider back and forth and immediately see the

effect; therefore, one does not need to guess the appropriate zoom level to reveal a desired helix.

Another zoom-related change is that the display will try to remain centered on the same location as before. This means that if the user has scrolled to center on a helix, the helix is likely to remain in view as the display becomes larger or smaller. These types of changes have a huge impact on productivity because there is less disorientation; the user does not have to spend time trying to rediscover a region of interest after zooming.

When the display is zoomed to the point of requiring scroll bars, a miniature view of the RNA display is now shown in a separate, movable window. A gray rectangle shows the entire view with the same diagonal axis line, and a white rectangle shows which part is currently displayed. This further helps with orientation, especially in gigantic data sets, as the user is able to see a relative location.

Sometimes users know exactly where they want to be but there is no easy way to go there. To help with this, new controls have been added to enter precise base-pair numbers (from 1 to the number of pairs in the current RNA sequence). These can be specified in both X and Y directions or the value can be entered just once to move along the diagonal. Direct entry of coordinates is a very efficient way to jump around the display, and a great way to return to previously-inspected helices.

The helix display itself was originally just a series of line segments, even for length-1 helices. The lines were very narrow (hard to click with the mouse) and they also

implied helix lengths that were not accurate (in particular, length-1 helices looked like length-2 helices).

Examples of usability improvements to the display can be seen in Figures 1 to 4. Since length approximation is important for recognizing helices, the display now uses a single "dot" for length-1 helices and lines of any thickness all terminate precisely at the helix start/end locations. Furthermore, any selected helix is annotated with hairlines; these lines help researchers to compare points with nearby helices, and they add an element of distinction besides color (not all people see colors the same way).

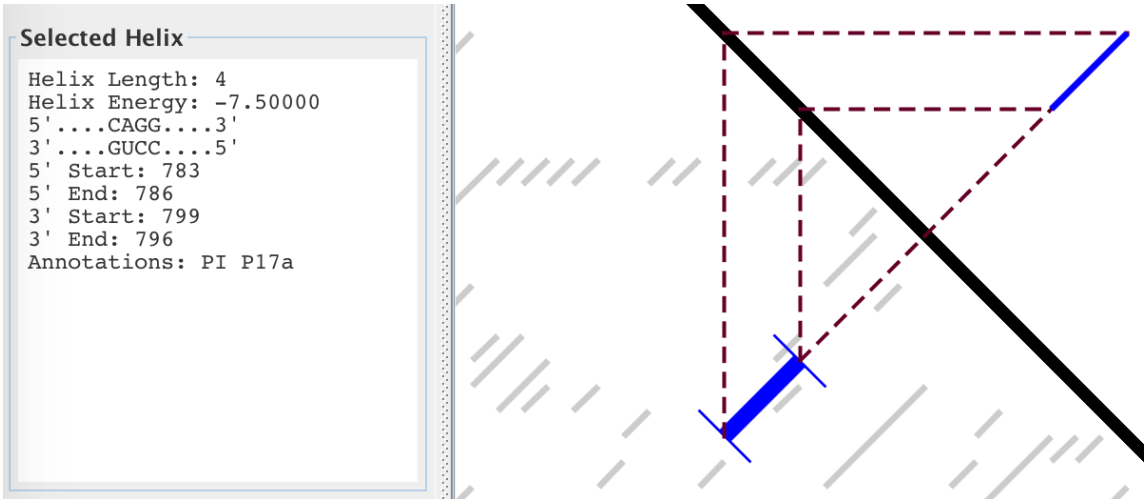

Figure 4: RNA HEAT v2.0 with Helix Matching

Now, when a helix is selected, guide lines appear that trace to the matching region on the opposite side of the diagonal. This allows the user to immediately see any differences between a "predicted" helix (such as that produced by a constraint) and the corresponding "actual" helix (read from the original RNA file). It is also obvious if the helix does not match anything at all. In version 2.0 the user can also click on even the "actual" helices above the diagonal so individual helix properties such as base-pair sequences can be compared.

What is not visible in the Figures is that helices can now be selected by clicking in a wider region around each line: not just on the line but in the blank space around the line. This makes RNA HEAT much easier to use because a person does not have to be as careful about mouse placement.

The window arrangement of the graphical interface has been changed to discourage important information from being "lost" behind other interface elements. Instead of 4 separate windows as in RNA HEAT 1.0, now the main display window is as large as possible and it contains panels for key information. Separate windows are now limited to things that are truly separate, such as any text and image files that have been opened.

Some changes were meant to accelerate common tasks. Keyboard equivalents were added for all main commands and some window controls, including the zooming operations. Dialog boxes for configuring constraints now have standard controls with standard key equivalents such as Return and Escape, which makes them easier to use as well.

Another change was simply the font, for helix information. The original font was proportional-width, meaning that letters on adjacent lines may not line up. Since helix information shows base-pair sequences as two lines of nucleotide letter sequences, the choice of font made it unnecessarily difficult to see pairings. Now the font is fixed-width and it is easy to see which nucleotides are paired between the 5' and 3' sides of a selected helix. Note an example of this in Figure 2 on the left-hand side.

Finally, a lot of improvements were simply in the form of documentation. Every menu command now has a "tool tip" that appears when hovering over the name, to describe what it does. And, the Help window (which was empty in RNA HEAT 1.0) now contains complete descriptions of all commands and all major interface elements, as well as the new scripting interface.

EFFICIENCY

Although efficiency was an explicit goal, all of the changes that provide efficiency gains are also side effects of other improvements. Efficiency is considered both for the raw performance of RNA HEAT, and the speed at which the user can accomplish every task that is required.

Raw performance in RNA HEAT is critical for two main actions: zooming the RNA display, and applying constraints.

Display updates occur very frequently. By no longer allocating memory for images or performing expensive pixel-by-pixel copies, the changes to address capacity also have a direct impact on response times.

Performance was not addressed for constraints. There was limited time, constraints are applied infrequently, and the longest delays were observed only in large RNAs. Also, given new features for integrating external programs, future constraints could be implemented in some other way.

In terms of user productivity, overall efficiency benefits from the improvements to extensibility, multi-tasking and usability. One can now perform a series of tasks very quickly and precisely by running a script. Programs and data can be easily linked together and viewed with the multi-tasking improvements. And with additions like keyboard short-cuts, common actions are quicker to perform. One of the most frequent actions, zooming, has been completely overhauled to provide more speed and flexibility, allowing for rapid identification and analysis of helices.

There is still the potential for further speed-up by looking at algorithms and data structures, as described in Future Work.

ACCURACY

There were a few accuracy issues that were not immediately apparent. Interestingly, some of them became apparent only after the RNA display was improved.

One issue was that "actual" helices did not appear at exactly the right locations. The original program had lower resolution so it was hard to notice that helices in the top-right triangle were slightly off, not lining up with their predicted counterparts. This was fixed by noticing an off-by-1 error.

Another issue was that "actual" helices did not store correct helix data. Since the original program did not allow "actual" helices from the top-right triangle to be clicked

(only “predicted” helices in the lower-left triangle could be clicked), the wrong data was not originally visible. Helices in the top-right were storing numbers based on their placement in the display so the numbers did not directly correlate to anything in the original RNA. This also made it very difficult to compare helices that appeared to be mirrored on the diagonal. The problem was corrected by moving coordinate translation out of the helix data store and into the display generator (where it was already correct). Now, when clicking an actual helix in the top-right triangle, the Helix Info numbers can exactly match a predicted helix in the lower-left triangle (assuming there is one). Similarly, the correct base-pair values are now shown for “actual” helices, since they were also found from the stored numbers.

TESTING

The new scripting interfaces developed for extensibility were also applied to add automated testing to the code.

For instance, a script can open a particular RNA data file, automatically seek certain helices and verify that helix properties match expected values. As development of the project continues, this type of script can be rerun to ensure that the results have not unexpectedly changed.

The test cases not only load data automatically but they verify results automatically. Appropriate error codes are generated by the test programs. As such, an entire suite of tests can be run and the passes and failures can be reported in a summary format.

Although RNA HEAT was originally a graphical program, this environment is not conducive to automated testing. As such, the software was redesigned to split graphical-only features from a core application that could run by itself. When test cases are launched, graphical windows do not appear; results are logged entirely with print-outs and output files.

Tests are a source of documentation for users and maintainers. They show real, working examples of scripts and they hint at how features are supposed to work.

FOUNDATION WORK

Although not the primary focus of the project, significant work was required at the beginning to allow new development on top of "RNA HEAT 1.0".

There was no mechanism to build the code (such as a project file from an Integrated Development Environment) so a Makefile was written. Also, when this 2003 program was built with the latest Java compiler, several warning messages appeared. The warnings were fixed, producing a more robust and maintainable program; for instance, by no longer using deprecated interfaces or vaguely-specified data structures.

It was important to verify the portability and behavior of the program before making changes so that any observed defects would be straightforward to explain. With the help of the Gutell Lab, new builds of the original program were tried on contemporary Mac, Windows and Linux systems and with recent installations of Java.

Conclusions

This project has broadened the capabilities of RNA visualization software in the Gutell Lab and resolved many issues. The automation and program integration features have opened opportunities to use statistics and other research tools.

All available RNA samples from the Gutell Lab can now be used, allowing researchers to support their work with more sources of data.

Defects of many kinds have been corrected, ranging from the fundamental (like problems opening files and placing helices) to the inconvenient (like using tricks to force the display to update properly).

Teams of researchers will be able to work more efficiently and share results more easily because of better extensibility, multi-tasking and usability. New features like helix annotation have also provided easy ways for results to be visualized.

While capacity and efficiency are much greater than in "RNA HEAT 1.0", there are still clear opportunities for further optimization. These changes may not be critical but they would be convenient.

Finally, a foundation has been laid for future development. The program is easier to test, and examples and documentation have been added. The scripting interface has been designed for expansion.

Future Work

There are still enhancements to "RNA HEAT" that would benefit researchers.

One straightforward change is to automate more. The scripting interface could be extended to cover more tasks, and the program-running environment could automatically display more kinds of results.

More research data formats could be supported. The required effort may vary; there may need to be a new text parser, or data may come directly from a database server. (This program is written in Java, which has established ways to query databases.)

While this project significantly improved the efficiency of visualization, very large RNAs are still noticeably sluggish. And, if the user chooses to overlay many sources of data at the same time, even smaller RNAs may trigger lots of computation.

Some of the slow behavior is observed when rendering, and some when clicking the mouse on a helix. Those tasks share a need to iterate over data in a small region of the display, suggesting that the data should be arranged visually by helix location. (The data is currently arranged sequentially by nucleotide number.) There are established methods for arranging data by location, such as the R-Tree [4]; the general idea is to be able to identify all the helices in a target region without even looking at the vast majority of helices in the data set.

If certain helix properties are commonly queried, another technique may improve performance: trading off memory to reduce run time. Borrowing an idea from databases [5], one form of this trade-off is an index, which is a look-up table. The extra memory and setup time to manage an index pays off by returning certain results in much less time.

For some tasks, rendering speed may be more important than the beauty or even the accuracy of the display while the task is performed. Right now, the same rendering approach is used at all times for all RNA data. And yet, given a very large and/or remote-sourced RNA data set, it may make sense to "cheapen" effects during certain actions. For example: while zoomed very far out, there is less value in "correctly" displaying every helix because many of them are practically invisible anyway; in that case, skipping adjacent helices may be a lot faster and not noticeably change the result. Similarly, while performing dynamic operations such as dragging the zoom bar with the mouse, rendering quality could be temporarily cut down because the user is not likely to notice anyway and the response time is more important.

Finally, the program makes very little use of threads, which provide parallel execution. These might improve usability by making the program more responsive while long-running tasks are running (such as updating helices): the main user interface would not appear to "pause" if it were implemented using threads.

A number of other minor issues and to-do items remain as future work, and they have been logged as text files in the code repository in the top-level files named "Issues.txt" and "ToDo.txt".

References

[1] Grant, K. (2016). "RNA HEAT, version 2.0" [computer software]. University of Texas at Austin: Gutell Lab. Online repository: https://github.com/utrna/foldr/tree/master/RNA_HEAT (last revised Aug. 2016).

[2] Hariharan, G.; J. Zhang; A. Li (2003). "RNA Helix Elimination and Acquisition Tool, v1.0" [computer software]. University of Texas at Austin: Gutell Lab. https://github.com/utrna/foldr/tree/master/RNA_HEAT/Old_RHEAT_v1.0_Source.zip

[3] Gutell, R. "BPSEQ File Format". University of Texas at Austin: Gutell Lab. http://www.rna.ccbb.utexas.edu/DAT/3B/Standard/Help/Menuhelp#bpseq

[4] Guttman, A. (1984). "R-Trees: A Dynamic Index Structure for Spatial Searching". Proceedings of the 1984 ACM SIGMOD international conference on Management of data – SIGMOD '84. p. 47. doi:10.1145/602259.602266. ISBN 0897911288. http://www-db.deis.unibo.it/courses/SI-LS/papers/Gut84.pdf

[5] Garcia-Molina, H.; J. Ullman; J. Widom (2008). Database Systems: The Complete Book (2nd Edition). New York: Pearson. ISBN 0131873253.

Also, personal discussions with supervisors to understand domain and original software.