VISUALIZING BIOINFORMATICS ALGORITHMS

Final Project Report

Adrian Kwok
adriank@sfu.ca
SFU ID #200136359

CMPT 711
Professor Wiese
April 26th, 2011

1. Research Question

Bioinformatics is an interdisciplinary field, drawing expertise from many different areas of computer science, molecular biology, biochemistry, and other sciences. However, a fundamental understanding of the algorithms and workings behind the techniques and tools used in this field is crucial for success. Because of this, CMPT 711 is a required course for students at Simon Fraser University intending to pursue a degree in Bioinformatics; it is a course with a strong emphasis on the theoretical and algorithmic side of computing science. Unfortunately, due to its interdisciplinary nature, some of these students may not have had any prior formal training in computing science, making much of the course material difficult to understand.

Given the complexity of the algorithms covered in this course, is there an effective and efficient alternative method to present this information so that most students can build upon and solidify the concepts they learn during lectures? More specifically, is it possible to visualize these algorithmic concepts in such a way that students who aren't naturally adept at learning from lecture slides and textbooks[2] can do so more easily?

2. Objectives

As stated in the previous section, the main problem I addressed in my project was to make the more difficult algorithms discussed in CMPT 711 easier to understand for students without a strong background in computing science. I accomplished this by developing learning tools which allow students to interactively run through bioinformatics-related algorithms step by step while simultaneously showing what the algorithms (and its related data structures) are doing with strong visual cues. The most challenging aspect to this project was in reinterpreting the algorithms in such a way that it is easily comprehensible to students while still maintaining the integrity of the algorithms itself.

In my project proposal, I initially proposed three primary objectives for the project:

1.) The ***relevance*** of the tool for current and future students of CMPT 711.
2.) The ***correctness*** of the tool in portraying how a specific algorithm works.
3.) The ***effectiveness*** of the tool in helping students understand and analyze the runtime of difficult algorithms.

However, after careful deliberation, further refinements on these objectives were made. Due to the importance yet trivialness of the first objective, it was delegated as a preliminary objective – while ample thought was given in choosing the algorithms that the learning tools are built upon, the number of "difficult" algorithms introduced in each topic covered were few, and thus these decisions were relatively simple. Furthermore, the second initial objective – the correctness of an algorithm – while important, should be an implied characteristic of the

effectiveness of the learning tools developed; an incorrect interpretation of an algorithm would clearly result in an ineffective learning tool, and thus the objective itself is redundant.

With these thoughts in mind, the following four refined objectives were identified prior to the development of each learning tool and were the primary considerations during each tool's initial design process – most design decisions were based on fulfilling these objectives.

Preliminary Objective: **The learning tools developed should cover one difficult algorithm from three of the topics covered in the course.**

As mentioned previously, the learning tools should be based on difficult algorithms chosen from the topics taught in CMPT 711. In this semester's course offering, five of the following topics were covered: Restriction Mapping, Motif Finding, Genome Rearrangements, Pair-wise Sequence Alignments, and Multiple Sequence Alignments.

A difficult algorithm should be defined one that is the most complex in its respective topic, or one that is somewhat complex yet fundamental to the understanding of more complicated subsequent topics. The associated objective in the project proposal assumed that eight topics would be covered by the end of this course, and as such the initial optimistic estimate was that seven learning tools would be developed for these eight topics. Unfortunately, due to time constraints and unexpected difficulties, this objective has been refined to focus on only three learning tools for these five topics.

Objective 1: **The learning tools should be effective in portraying how an algorithm works.**

This objective is a difficult metric to quantify: effectiveness, in this context, can be generally defined as whether a student using the tool will be able to learn what an algorithm is doing at each step. More specifically, the tool can be deemed to be effective if the tool portrays the material in a more useful or convincing manner than via the lectures and the textbook alone. Do note, however, that this objective does not imply that the tools are meant to be substitutes for conventional teaching methods – they are merely meant to supplement the course material, not replace it. In this vein, the learning tools should effectively show what an algorithm is doing at each step, but not why it is doing it – the onus of algorithmic analysis is placed on the course instructor and is beyond the scope of this project.

To create an effective learning tool for an algorithm, it is imperative that careful analysis of the algorithm be performed such that its most crucial aspects – for example, its underlying data structures, its decisions made at each step, and its computational complexity – are portrayed efficiently with its corresponding ideal visual representations. Moreover, the tool must also be aware of information overload: many of the algorithms discussed in CMPT 711 actually have

many intricate details happening at each algorithm iteration, and the more information that is portrayed on the screen at any given time leads to an overall ineffectiveness of the whole interface.

Since the learning tools are intended to supplement the lectures and text, it is important that the tools follow the way the algorithms are portrayed there. A severe disconnect between a learning tool's interpretation of the algorithm and how an instructor chooses to teach the material will undoubtedly lead to confusion; an effective tool is one that causes the least amount of ambiguity as to how an algorithm actually works, without contradicting the way the material is initially taught to the student.

Objective 2: **The learning tools should work for a variety of different inputs and allow a student to traverse from algorithm iteration to iteration.**

The justification for this objective is that this encourages students to explore an algorithm's inner workings – if a student was restricted to only a few sample inputs, they would be unable to try to see on their own when an algorithm would perform admirably and when it would perform poorly (this is the next objective, Objective 3) – for example, many algorithms perform exponentially worse when the input size increases linearly: it is imperative to allow for such inputs so that students can understand these failures.

Furthermore, by allowing a student to efficiently traverse from step by step on their own, they will be able understand how an algorithm works at their own pace; the ability to backtrack whenever necessary, or quickly revert back to an algorithm's initial or completed state encapsulates a variety of different uses for the tool aside from its original intention – for example, a student may be able to use the tool to check whether their handwritten implementation of an algorithm is correct.

Objective 3: **The learning tools should show the strengths of weaknesses of each algorithm.**

This objective focuses on the actual complexity of an algorithm; ideally, the strengths and weaknesses of each algorithm should be *implicitly* shown via the learning tool – that is, for some inputs, it should be abundantly clear when an algorithm performs poorly. The basis for this objective lies in a shortcoming with many of the examples shown for these algorithms in both the textbook and the lectures; for example, in the Branch and Bound algorithm for the Partial Digest Problem, trivial, almost best-case inputs are used to illustrate how the algorithm works. Without understanding why or when an algorithm may perform poorly, a student may arrive at an incorrect notion regarding the effectiveness of an algorithm; although formal runtime analyses such as the use of Big-O notation attempt to illustrate this, actual practical examples drive the point home for students who are not familiar or particularly comfortable with theory.

3. METHOD

As mentioned previously, the three primary objectives were referred to throughout the design process of each learning tool. To fulfill the first preliminary objective, three learning tools were developed for the three topics – Motif Finding, Restriction Mapping, and Pairwise Sequence Alignment – corresponding to the *Branch and Bound Algorithm for the Median String Problem*, the *Branch and Bound Algorithm for the Partial Digest Problem*, and the *Dynamic Programming Solution to the Global Pair-wise Alignment Problem*. The justification for choosing these algorithms will be discussed in their own respective sections.

Of particular note, a learning tool was not developed for the Genome Rearrangements topic as the algorithms covered in those lectures were relatively trivial – the most difficult algorithm was *ImprovedBreakpointReversalSort*, which requires simply defining breakpoints and increasing strips and can be easily replicated and understood by identifying them on paper. On the other hand, while Multiple Sequence Alignment was one of the more difficult topics presented in the course, it was covered hastily during the last week of the semester, and as a result there was not enough time to develop a fully fledged tool for it.

For each learning tool developed, it was important that clearly written instructions were given to the student at each step of an algorithm, explaining what it is doing on the off-chance that the visualization is unclear. Each learning tool will now be discussed in detail, highlighting how they tackled the objectives stated previously.

3.1 METHOD: MOTIF FINDING

The algorithm that was focused on for the Motif Finding topic was the Branch and Bound Algorithm for the Median String Problem. Two different approaches for finding motifs were covered in this topic – the first by analysing the search space of motif starting positions in each sequence (Motif Finding), and the second by analysing the search space of possible candidate l-mers (Median String). As will be discussed shortly, the Median String representation of the problem was chosen as it allowed for a more natural way to visualize the algorithm's steps – furthermore, the median string approach is a much more practical algorithm as the search space does not grow alongside the length of the input sequences.

For the learning tool to be *effective*, there were three main factors that needed to be considered:

1.) How the algorithm goes about finding l-mers to analyse, and how it bypasses pointless l-mers.
2.) What the algorithm actually does when it is analysing an l-mer – that is, how does it calculate the total distance for an l-mer?

3.) The effectiveness of the bounded approach – that is, how many actual l-mers are bypassed as the algorithm progresses? The number of analysed prefixes?

This learning tool was the most difficult to develop out of the three – not in terms in technical difficulty, but in terms of figuring out the best way to interpret the algorithm's decisions at each step. At times, Objective 1 and Objective 2 from Section 2 were in direct conflict with another: the algorithm relies heavily on the use of a prefix tree, which is a fully balanced tree with $4^{l+1} - 1$ nodes, where l is the input target l-mer length. With a l-mer length of $l = 8$, which is a reasonable input size, this corresponds to 262143 nodes in the prefix tree, making it extremely difficult to provide a visualization where screen real estate is heavily constrained. Even initial attempts in visualizing a prefix tree with l-mers of length $l = 5$ proved to be extremely difficult without a sophisticated zooming mechanism, but even then, students may be confused when the algorithm is zoomed-in and is shifting from node to node.

Furthermore, another problem is that for students without a formal theoretical background, tree traversals may not overly intuitive; even though a tree-based structure is familiar to most computer scientists, the prefix tree itself and the algorithm's operations on the tree – *NextVertex()* and *BypassVertex()* – cause confusion, even to graduate students in CS such as myself. In actuality, there is a more natural and habituated way to represent these operations in a list-based structure:

1.) *NextVertex()* essentially increments a number in the usual way with carry-overs. For example, applying *NextVertex()* in succession to a number such as 189 will result in 190, 191, 192, and so forth. With the added complexity of number prefixes, a small modification would be made such that prefixes are "counted" as an additional digit. For example, 189 would result in 19-, 191, 192, 193, and so on.
2.) *BypassVertex()* is similar to *NextVertex()*, but is utilized on prefixes only, and with the last non-prefix digit being incremented. For example, applying *BypassVertex()* in succession to 18- would become 19-, 2--, 3--, 4--, and so on.

Thus, a decision was made to present the prefix tree in the algorithm as a list of traversals along the tree. This further allows a student to easily identify l-mers of particular interest: for example, l-mers that are found to be temporary candidates for the actual motif, could be easily highlighted in the list to denote importance. Furthermore, each step of the algorithm can be naturally defined as an l-mer; traversing through the steps of the algorithm would be no different than traversing through each of the analysed l-mers in the prefix tree (Objective 2). As an added benefit, the list-based approach allows the student to intuitively see that the number of *Bypasses* increases significantly as the algorithm progresses.

To show what the algorithm actually does when it is analysing an l-mer, it was necessary to show where the best alignment in each sequence a candidate l-mer is, and its corresponding

number of mismatches (hamming distance). This was a facet that was clearly conveyed through the lecture slides and the textbook, and as such, a similar representation is used in the learning tool.

To fulfill Objective 3 – that is, showing the strengths and weaknesses of the algorithm – it was necessary to provide adequate visual feedback as to how many prefixes and l-mers are analysed, how many are bypassed, and what the total number of possible l-mers are – this clearly conveys the importance of the bounded approach over the bruteforce approach. This was done both via color codes in the l-mer traversal list and via text labels showing these properties at each step. In combination with Objective 2, students would be clearly able to see that for large input sizes (e.g. $l = 12$ or with a large number of long input sequences), while the bounded approach is able to shave off a considerable number of l-mer analyses, there are still a significant number of prefixes that need to be analysed.

3.2 METHOD: RESTRICTION MAPPING

The algorithm that was deemed to be the most difficult in the Restriction Mapping topic was the Branch and Bound Algorithm for the Partial Digest Problem. Given that the only other algorithm that was covered in this topic was the brute force approach, this was a trivial decision.

The algorithm itself is straightforward and was used as an initial testbed for development techniques for subsequent learning tools. The algorithm was covered in detail in the lecture slides with a relatively simple visual representation of the restriction map shown for each remaining length insertion into the map, as shown in **[Figure 0]**. However, this visualization only works well when the length chosen at each step actually fits (i.e. no backtracking is involved); it does not take into account "in-progress" length decisions, nor does it take into account the branching nature of the algorithm and what can happen when an incorrect decision is made.

***[Figure 0]**: Lecture slides' visual representation of the Restriction Map*

As stated in Objective 2, students should be able to easily traverse from algorithm iteration to iteration; however, it was difficult to decide on a proper granularity for each step. If a step ignores the algorithm's decision making process – that is, a step only shows a length being placed on either the left or right side of the restriction map – the student may be to be confused as to *how* and *why* a length's position on the restriction map is inferred. Furthermore, it should be abundantly clear what it means for a length to 'fit'; the additional lengths created by adding a singular length to the restriction map should be shown, and if these additional lengths are not part of the available lengths left, then the singular length would not fit. In this vein, a step in the

learning tool was defined as separate decisions made by the algorithm, with additional steps for actual length placements.

To account for the need to visualize lengths that are under consideration of being placed on either the left or the right of the restriction map – that is, to show the student what additional lengths would be created by the placement of a specific length – a 2-dimension map was adopted in the learning tool. The extra dimension allows us to convey length placement attempts via the use of solid colors, and *actual* length placements via solid lines as in **[Figure 0]**.

To account for the need to visualize 'poor' decisions made by the algorithm, a colored decision tree was used, where a left branch on the decision tree infers that a length was placed on the left side of the restriction map, and a right branch infers that a length was placed on the right side of the restriction map. Without formally portraying the decisions and backtracks the algorithm makes, it is extremely difficult to understand why the algorithm has an exponential runtime, violating Objective 3. Furthermore, since recursion is a difficult topic for many students, the learning tool needs to be abundantly clear where each visual step corresponds to in the overall algorithm in order to adhere to Objective 1; this was accomplished by highlighting specific line numbers in the algorithm pseudo code provided in the lecture slides and textbook at each step.

3.3 METHOD: PAIR-WISE ALIGNMENTS

The algorithm that was focused on for the Pair-wise Alignment topic was the Dynamic Programming solution for the Global Pair-wise Alignment Problem. More specifically, a learning tool was developed for the simplest variation of the problem – the Longest Common Subsequence problem – where only nucleotide matches influence the score of an alignment. The rationale behind choosing this simple algorithm to visualize is that **all** other algorithms in the same topic (Chapter 6 in [1]) utilize some variation of this underlying framework – for example, the second solution covered is based on the same algorithm but with penalties based upon a simple lookup in a scoring matrix, and the most complicated algorithm incorporates affine gap penalties, which is also just a simple modification of the underlying scoring mechanism. Understanding fundamentally how the most basic sequence alignment works is imperative for success in this and subsequent topics; more specifically, while dynamic programming is a technique that is well known for most computer scientists, students who have never utilized this technique may misunderstand or misinterpret the algorithm, compounding the issue when alignments are generalized past paired sequence comparisons.

As dynamic programming is best visualized using a tabular structure, Objective 1 was accomplished by closely following the textbook's representation of the score matrix with backtracking during the creation of this learning tool. The algorithm itself is straightforward – given m- and n-long nucleotide sequences, an $m\ by\ n$ table is initialized with scores of zero in the first row and first column. Then, each cell in each row is populated based on the scores of its

neighbours, and a backtracking pointer is used to point to the neighbouring cell that it inferred its score from. When the table is finished populating, a backtrack path is generated using the backtrack pointers starting from the (m, n) cell, and a global alignment is inferred using this path. However, the algorithm presented in the textbook does not result in the same backtracking pointers generated via the examples in the same textbook; the algorithm described in the textbook results in backtrack arrows being assigned differently in tie-breaking situations that do **not** correspond to the example tables in the textbook. Ideally, if there are multiple optimal alignments (i.e. same LCS length), the learning tool's dynamic programming table should be identical to the examples shown in the textbook – this was remedied by modifying the algorithm slightly when it assigns backtracking pointers in tiebreaking scenarios.

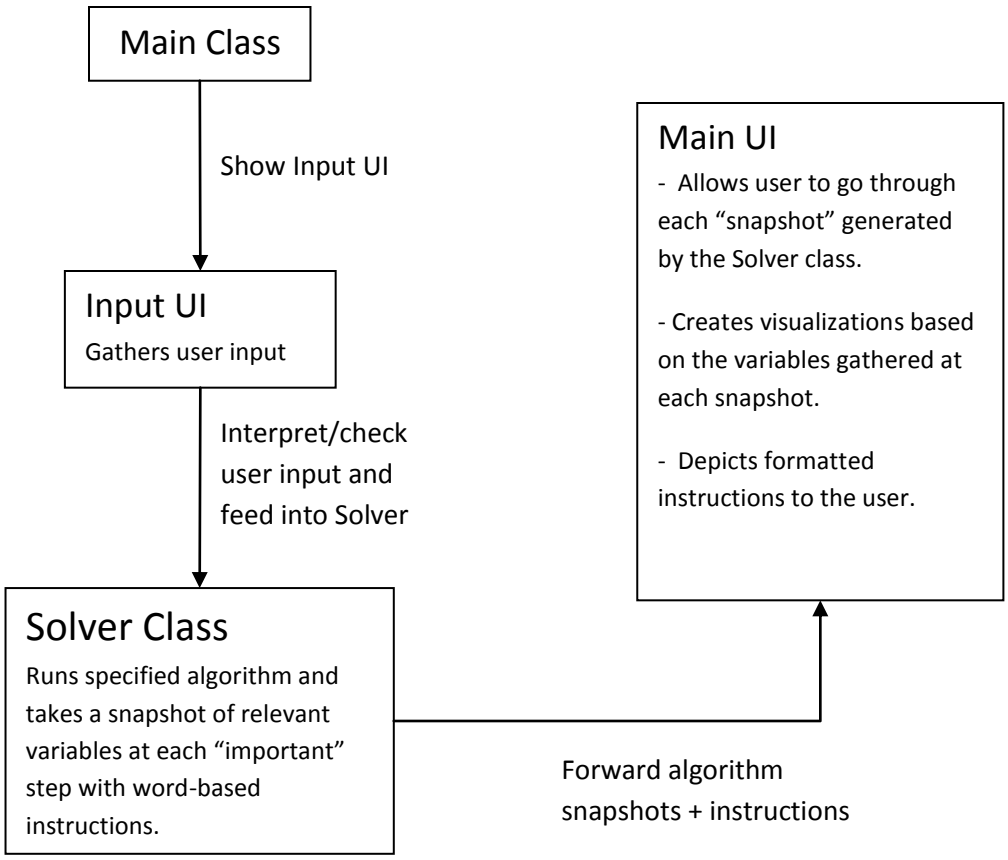
Similarly, there is ambiguity in the last step of the algorithm in the lecture slides. The formal LCS algorithm only prints out the nucleotide matches between the two sequences recursively – how the actual backtracking path is converted into a global alignment is unclear. While it is true that insertions and deletions can be inferred simply by the directionality of each backtracking pointer, *why* this is the case is vague and insufficient for a learning tool. Fortunately, the textbook uses a different representation of the backtrack path to more easily convey how an alignment is created ([1], pp. 170), and thus this representation was used in the final steps of the learning tool.

Another problem faced was that both the lecture slides and the textbook do not utilize backtrack pointers for the first row and the first column of the dynamic programming table – this leads to only a partial alignment when the input sequences are **not** ideal. For example, the two sequences "ACG" and "TTTACG" would result in a backtrack path that creates an alignment of "ACG" and "ACG", not the full alignment of "---ACG" and "TTTACG". This was corrected by creating initial backtrack pointers for the first row and first column pointing back to the $(0,0)$ cell, as shown in [6].

Since it is important that students are able to trace through the steps of the algorithm manually (Objective 2), choosing an ideal step granularity is an important concern. Surely, populating the entire table in one step (e.g. as shown in the textbook) trivializes the problem, and the generation of each cell of the table should be clearly defined as singular steps. The learning tool addresses this by counting the table initialization as one step, each subsequent table entry population as an additional step, the identification of the backtrack path as one step, and the generation of the alignment as several steps, with each step corresponding to an entry in the backtrack path. As an additional benefit of this step definition, students would be able to clearly see the amount of time and space it takes to analyse each step of the m by n table, and should be able to infer that further improvements can be made onto the algorithm itself (Objective 3) – for example, that it isn't entirely necessary to store the whole table in memory to calculate the best possible alignment.

4. TOOL

The learning tools were written in Java, and used Netbeans 6.9's GUI builder for developing the user interface of two of the three tools. The Model-View-Controller pattern[8] was used extensively throughout development, which decouples the actual algorithm computations and the GUI. A structural diagram of major components for the learning tools is provided in **[Figure 5]**.



***[Figure 1]**: Structural Diagram of Major Components*

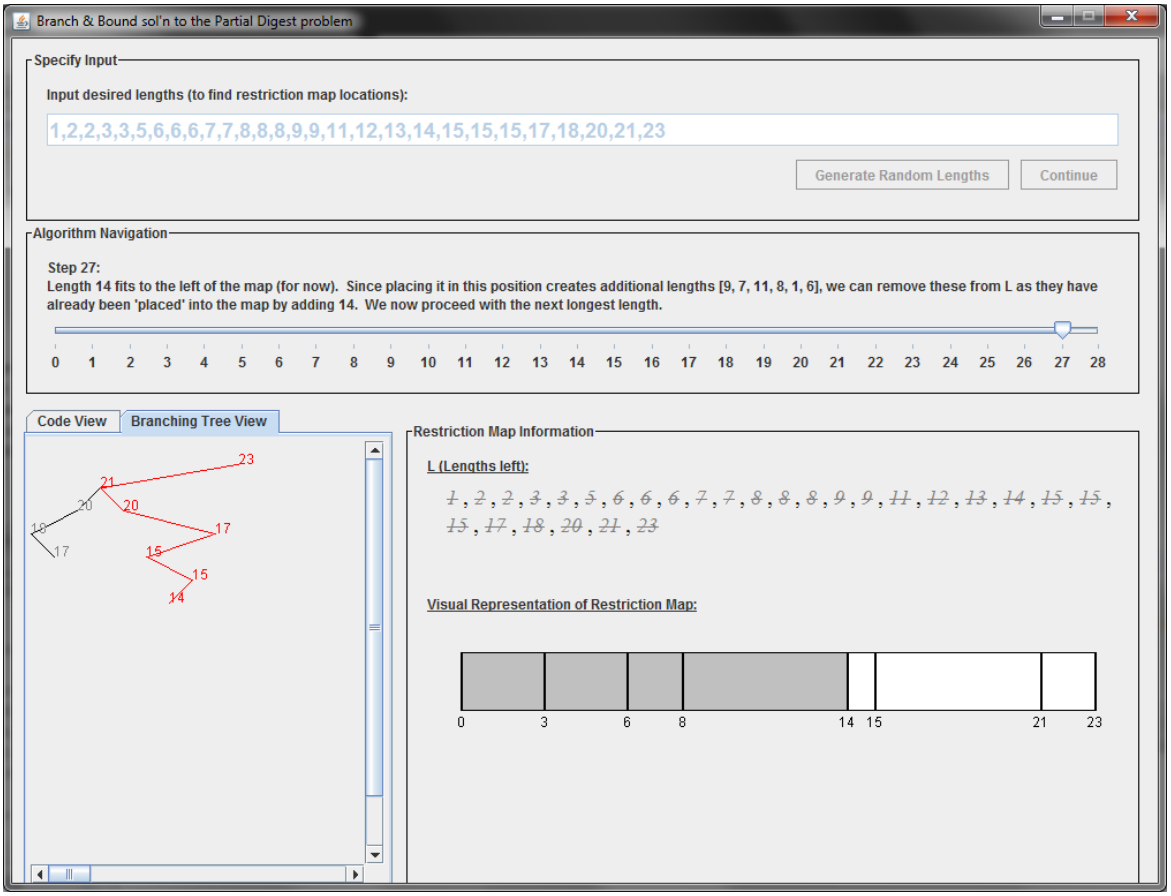
The main class for each learning tool simply creates an InputUI instance, which shows the initial GUI for querying user input for the underlying algorithm. Once the user has confirmed the input, the problem is solved using the corresponding algorithm. While the problem is being solved, a snapshot of the algorithm is taken at each step (which includes important variables utilized later in the visualization), and hand written instructions are also compiled describing what the algorithm is doing. These snapshots are stored in an *ArrayList* of *AlgStep* objects for the Restriction Mapping and Motif Finding learning tools, and, for efficiency, as an array of instruction *String*s for the Pairwise Alignment learning tool. The actual Main GUI is then created and **only** has access to these snapshots generated by the algorithm – it then portrays these snapshots in a visual manner to the user, and allows them to traverse through the snapshots as requested.

5. User Guide

The learning tools have been precompiled and can be launched in the by double-clicking **PartialDigest.jar**, **MedianString.jar**, or **PairwiseAlignment.jar**. Alternatively, using Netbeans 6.9, it is possible to select each learning tool's folder as a project and recompile it if necessary. Since the learning tools are meant to be user friendly, simple default inputs are provided at program startup, and randomized input can be generated using simple dialogs. Using these learning tools should be relatively self explanatory, given its intended purpose.

6. Sample Session

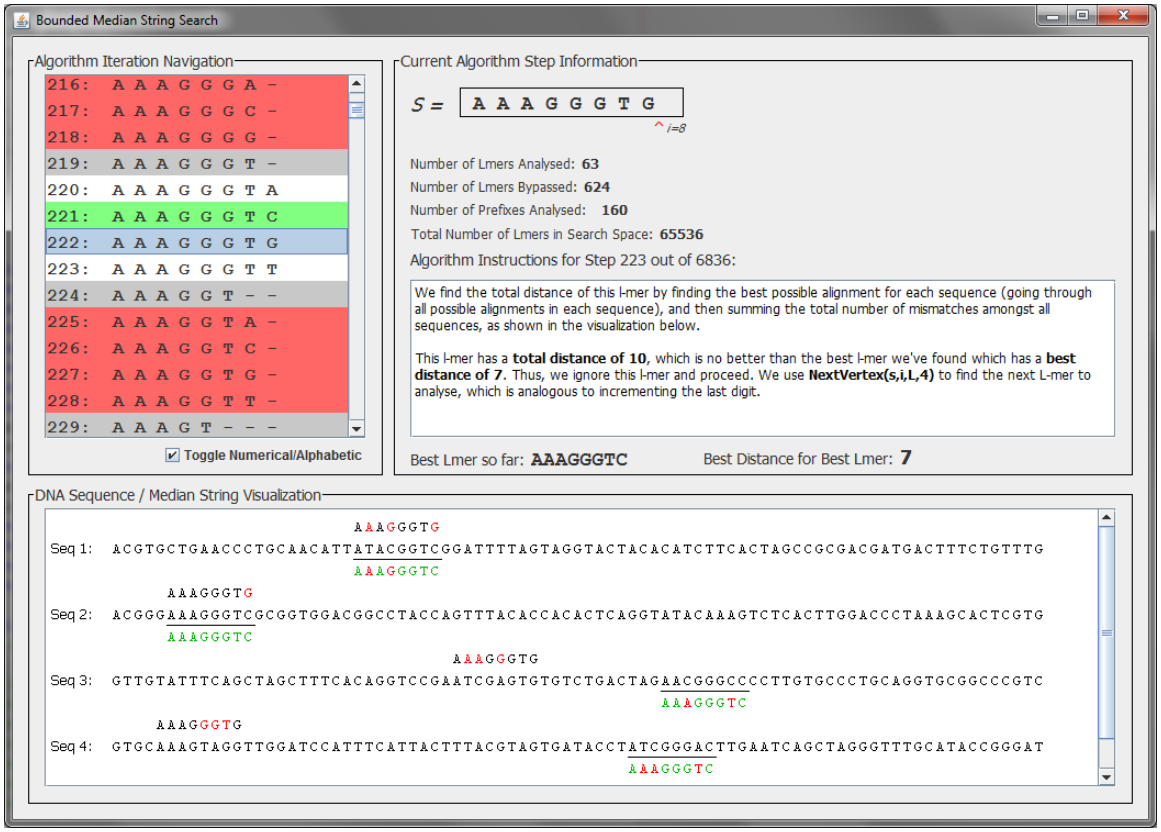
[Figure 2] shows an instance of the learning tool for the Restriction Map topic, with the branching tree visualization tab visible, which highlights where lengths have been placed in the past and removed (in gray) and are currently placed (in red). Users are only able to navigate through the algorithm using only a horizontal slider, purposely constraining the amount of possible actions a user can perform. The in-progress restriction map and current length decisions are shown with a corresponding visual representation.



***[Figure 2]**: Learning tool for the Restriction Map topic*

[Figure 3] shows an instance of the learning tool for the Motif Finding topic. As discussed in the methodology section, a list of l-mers and prefixes being traversed is shown with color codes denoting algorithm decisions at each particular step. A step outlined in red denotes a prefix that is *bypassed*, green denotes an l-mer that was recorded as having the best total distance up to that point, and grey denotes a prefix that was not bypassed. At a specific step, the user is shown the number of l-mers analysed and bypassed, as well as the prefixes analysed up to that point.

To illustrate how the total distance for the l-mer at a current step is calculated, the sequence visualization panel shows the best possible alignment (i.e. minimizes the hamming distance) for the l-mer in each sequence. Mismatching nucleotides are depicted in red. Furthermore, the best possible l-mer found up to that step is shown below the sequence highlighted in green, and similarly nucleotide mismatches are highlighted in red.

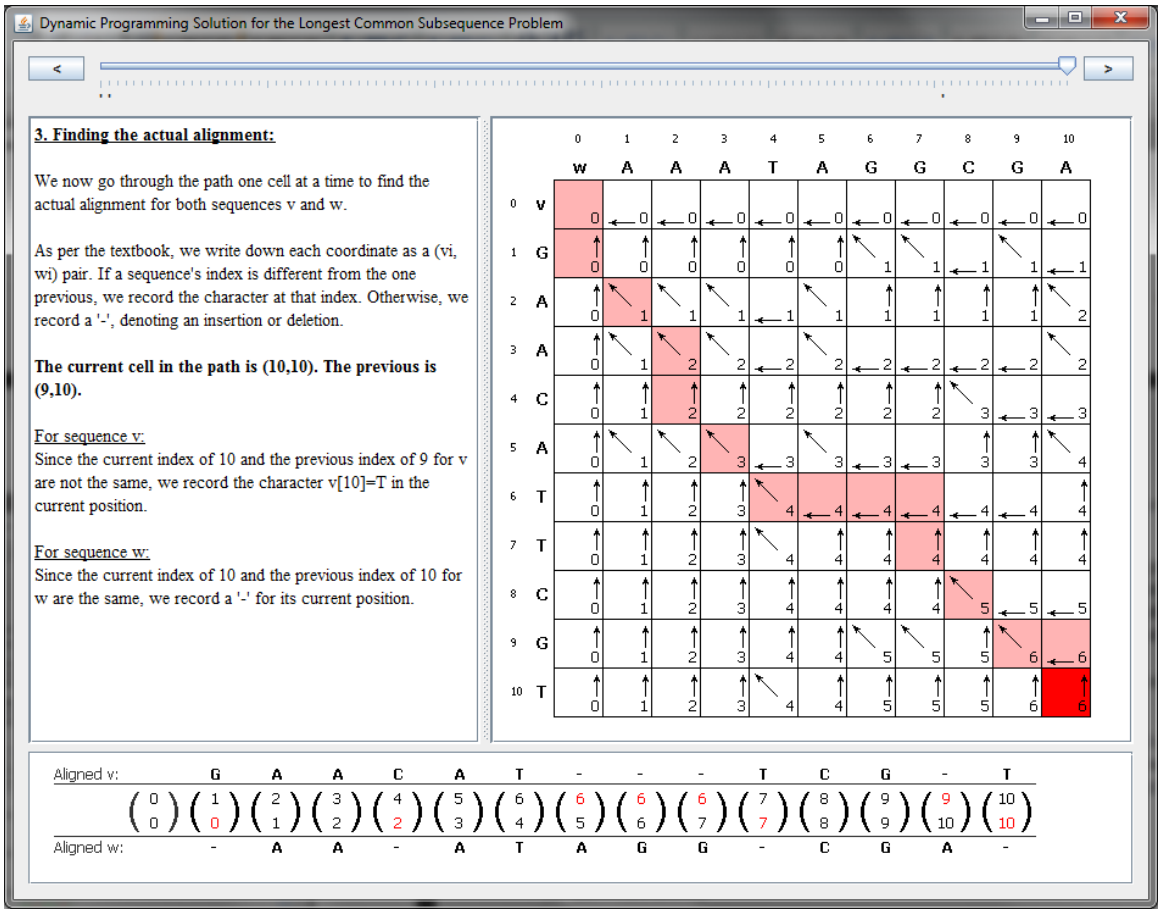


***[Figure 3]**: Learning tool for the Motif Finding topic*

[Figure 4] shows an instance of the learning tool for the Pair-wise Sequence Alignment topic, with the last step of the algorithm being currently selected. The dynamic programming table is shown on the right portion of the screen, and each cell shows the best possible score for the two subsequences at that cell, as well as the backtrack path leading up to that cell. This tabular representation is based heavily upon the examples shown in the textbook ([1], pp. 173).

At predetermined intervals, highlighted with a marking on the algorithm step slider, the tool goes through the table's initialization step, each cell's population step, and the actual alignment generation steps.

The backtrack path corresponding to the final alignment is shown in a panel at the bottom portion of the tool. This representation is also drawn heavily from the textbook ([1], pp. 170), and the red cell indices denote insertions and deletions.



***[Figure 4]**: Learning tool for the Pair-wise Sequence Alignment topic*

7. LISTING

The Java code for each learning tool is placed under its respective */src/* directory. The code is documented in-line, and non-trivial functions have method comments. Only the standard libraries *java.util.**, *java.awt.**, and *javax.swing.** were used, and all other code is hand-written with no external help. A full file listing is provided in *listing.txt* in the root directory.

8. RESULTS AND EXTENSIONS

Due to the nature of my project, I emphasized the objectives and methodology sections moreso than the results of my project; the only results I have are the tools I developed, and user testing is necessary for more in-depth discussion.

Prior to the creation of the learning tools, the objectives in Section 2 were defined and referred to throughout the design process. However, they are high level concerns and give a sense of what needs to be developed, but not exactly *how* they should be portrayed on the screen. When actually writing the GUI-centric code, Donald Norman's fundamental principles of design[7] were used as a guide for most design decisions such as widget placement and choice.

In the end, I came up with what I felt were the best solutions in accomplishing the initial objectives, but how do I know whether these assumptions are correct? Aside from testing with users, there is no other adequate way to judge the effectiveness or usefulness of these learning tools; heuristic evaluations may be able to address minor UI problems here and there, but it is absolutely necessary to involve users in an iterative process to come up with a well polished and final design – in this case, users would be current and future CMPT 711 students. More specifically, it is necessary to:

1.) Interview or survey users to see what algorithms covered in a particular topic are the most troublesome.
2.) Create a prototype (paper prototype initially) that attempts to facilitate the learning of said troublesome algorithms.
3.) Gauge the effectiveness of the prototype by testing it with users – this involves a qualitative study of what users feel are strong and weak points of the prototype. Ideally, it would be preferable to observe and record users trying to accomplish certain tasks with the prototype – for example, trying to figure out the complexity of a certain algorithm with predefined inputs.
4.) Refine and reiterate the prototyping process until there is a certain confidence that the application is useful and user friendly, based on the initial defined objectives.

Unfortunately, due to time constraints, none of these steps were accomplished; while I myself am a current student of CMPT 711 and the tools are deemed to be effective for my own uses – for example, in studying for the midterm – this is a biased metric. However, the tools developed are more than adequate initial prototypes that can be easily refined with one or two rounds of observational user testing with a few users; as such, the most important and necessary extension to this project would be to thoroughly test these learning tools with users. In their current state, the learning tools will most likely help students better understand the course material, but not quite possibly in the most user-friendly way possible.

9. CONCLUSIONS

While three visualizations for three different topics – Restriction Mapping, Motif Finding, and Pairwise Sequence Alignment – were developed, they are merely prototypes in creating a final suite of polished learning tools. However, they are steps in the right direction in making the course material in CMPT 711 more interactive and accessible for students; while extensive user testing will be necessary to determine the feasibility and effectiveness of these tools, these prototypes in conjunction with the lecture slides and textbook should still be at least as effective than learning from just text-based course material alone.

10. REFERENCES

1. Jones, N. and Pevzner, P., "An Introduction to Bioinformatics Algorithms," MIT Press, Cambridge, MA, 2004.
2. Layman, L., Cornwell, T., and Williams, L., "Personality Types, Learning Styles, and an Agile Approach to Software Engineering Education," *proceedings of ACM Technical Symposium on Computer Science Education* (SIGCSE '06), Houston, TX, 2006, pp. 428-432.
3. Wiese, K. C., "An Introduction to Bioinformatics Algorithms", Lecture Notes for CMPT 441/711, Simon Fraser University, 2011.
4. Topley, K., "JavaFX Developer's Guide," Addison-Wesley Professional, 2010.
5. JavaFX: http://javafx.com, accessed February 19th, 2011.
6. Eddy, S. R., "What is dynamic programming?". Journal of Computational Biology (2004), Volume 22, Issue 7, pp. 909-910.
7. Norman, D. A., "The Design of Everyday Things," Basic Books, 2002.
8. eNode, "Model-View-Controller Pattern," (2002). http://www.enode.com/x/markup/tutorial/mvc.html, accessed April 24th, 2011.