HAGA: Human-Assisted Global Alignment

Robert Toscano and Shannon Iyo

December 14, 2007

Abstract

We describe and implement a methodology for creating global sequence alignments based on algorithms constrained by human-defined local alignments. In addition, we show a method for integrating varied pattern-recognition algorithms into an alignment algorithm. Our preliminary results are a proof of concept, while there is still room for improvement in the future.

1 Introduction

Global sequence alignment is the process of taking two sequences and computing the best element-to-element alignment of those sequences. The most common alignment candidates are DNA sequences, although the process is also applicable to amino acid strands and RNA sequences. In genetic biology sequence alignment is used when comparing the genomes of related species or individuals. This is important because genetic material changes across generations making it impossible to know a priori the alignment of DNA sequences from different organisms. Thus a sequence alignment must be computed before genetic comparisons can be made.

Sequence alignment presents many challenges. It is difficult to quantitatively assess the accuracy of an alignment. This because there is no way to observe evolution directly; hence there is never a ground truth to compare generated alignments with. The data being compared is very noisy - random insertions, deletions, replacements, transversions and duplications can and do occur. Further complicating matters is the fact that mutation rates vary significantly across different regions (for example, there is high conservation in coding regions and low conservation in non-coding regions). All of these factors make the sequence alignment process far from trivial.

Because it is both important and difficult, sequence alignment is an area of computational biology that has received much research in the past. The Needleman-Wunsch dynamic programming algorithm provides a simple method for generating a global sequence alignment, with a simple score matrix and gap penalty as its parameters. Many global alignment algorithms seen today are modifications or optimizations of the basic Needleman-Wunsch approach, such as the LAGAN algorithm, which anchors the search around high-scoring local alignments [2]. These traditional alignment methods suffer from some important limitations. They have a static model of the alignment problem, and thus incorporating new types of knowledge is difficult to impossible. For example, a researcher might have a very good method of identifying protein motifs, but this cannot be utilized for alignment unless the entire algorithm is re-written.

Human-Assisted Genome Alignment (HAGA) is an alignment methodology that addresses many of the issues found in traditional sequencing approaches. HAGA implements a framework for global alignment based on human-defined local alignments. The algorithm also generalizes from these annotated segments in order to improve the rest of the global alignment. In this way the pattern-matching expertise of a human can complement the calculational power of a computer to generate improved results. We also describe a methodology for the insertion of general pattern-detection systems into the alignment algorithm.

2 System Design

2.1 Labels

The HAGA system makes use of *labels*. A *label* is an annotation of a subsequence describing what type of region it is. For example, some regions may be labeled as exons and others as introns.

2.2 Overall Architecture

The architecture is broken into three modular components. The user interface allows a human to view alignments and define labels on the sequences. The labeler is responsible for learning the labels specified by the user, and applying these labels to unlabeled sequence regions. Our labeler implementation uses a HMM. The aligner performs global sequence alignment on two labeled sequences. Our system uses the Needleman-Wunsch algorithm, modified to favor bases with the same label. HAGA is designed so that any of these components may be interchanged, as long as the replacement follows the same interface. For example, human label definitions could be replaced or augmented with a miRNA recognizer, or an HMM labeler implementation could be replaced with a neural net solution.
The alignment process follows the steps below:

1. User (or other pattern recognizer) applies labels to sequences
2. Labeler is trained on defined labels
3. Labeler applies labels to unlabeled sequence regions
4. Aligner performs alignment on the labeled sequences
5. User interface is updated to display the new alignment

2.3 User Interface

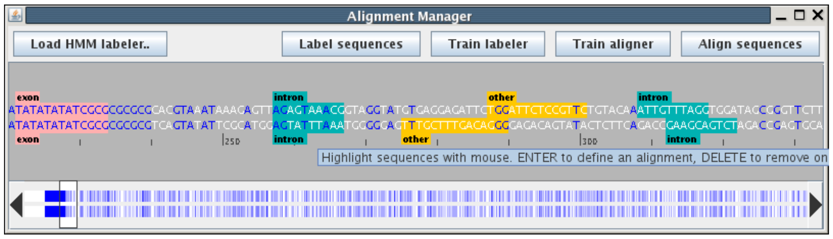
The user interface allows human viewing and editing of algorithmically generated alignments. It consists of a main window with a button toolbar, an editing area, and a slider control.

The toolbar has buttons controlling the backend parts of the system. There are buttons for loading an HMM labeler, training the labeler, running the labeler, training the aligner, and running the aligner. When performing any of these tasks, the editor window and slider control will be appropriately updated to show the changed state of the data model.

The editing area displays two aligned sequences of DNA. The region being viewed is controlled by the slider control under the editing area. In the editing area, matching base pairs are displayed in blue, while unmatching base pairs are displayed in white. Below the two sequences are numbered tickmarks to indicate the position in the sequence.

To specify a local alignment, the user begins by highlighting the matching regions of the sequences using the mouse. After highlighting a pair of regions, pressing the enter key will open a dialog box asking for the label name to be applied to the regions. Once the label name is entered and "Okay" is pressed, the labels will be created and appear in the editing area as a colored highlight on the applicable regions. Labels with different names will appear in different colors to help distinguish them. These local alignments can be selected by clicking on them, and deleted by pressing the delete key while selected.

The slider control shows a thumbnail view across the span of both sequences, and indicates region identity via color value. Coloration ranges from white to blue, with higher-scoring regions indicated by a more vivid blue. There is a transparent box whose positions and bounds indicate the part of the sequence that is currently viewable. Clicking or dragging on the thumbnail will scroll the view to that point in the sequence. The view may also be scrolled by clicking on the left or right arrow buttons, or by rolling the mouse wheel.


Alignment Manager
Load HMM labeler..
Label sequences
Train labeler
Train aligner
Align sequences
exon
intron
other
intron
ATATATATATCGCGCGCGCGCACGTAAATAAAGAGTTAGAGTAAACGGTAGGTATGTGAGGAGATTCTGGATTCTCCGTTCTGTACAAATTGTTTAGGTGGATAGCGGGTTCTT
ATATATATATCGCGCGCGCGTCAGTATATTCGCATGGAGTATTTAAATGGGCAGTTTGCTTTGACAGGGAGACAGTATACTCTTCAGACCGAAGCAGTCTAGACCGAGTGCA
exon
250
intron
other
300
intron

Highlight sequences with mouse. ENTER to define an alignment, DELETE to remove on

Figure 1: The user interface allows viewing and editing of sequence alignments.

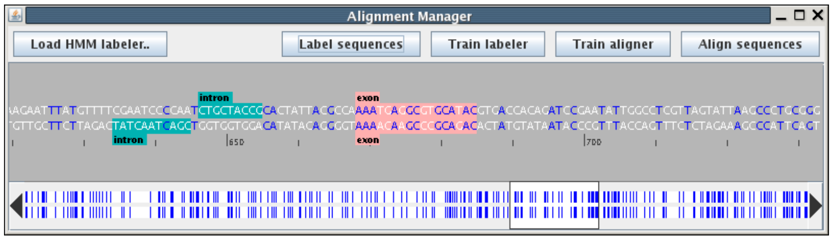


Figure 2: The user interface with two user-defined labels, prior to alignment.

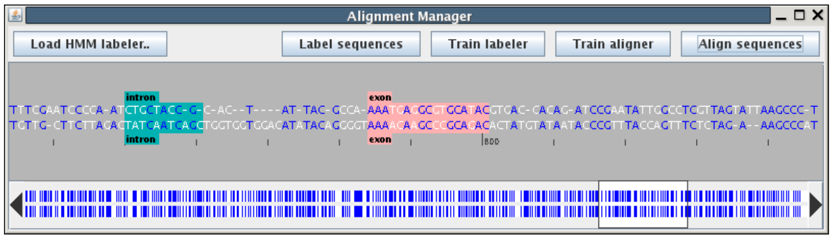


Figure 3: The user interface displays the results of the alignment. There are now more aligned bases, as indicated by the increase in blue regions on the scroll bar.

2.4 Labeler

The labeler component of HAGA works on a unit called a labeled sequence. A labeled sequence consists of a character array and a list of labels. Each label defines a start and end index (inclusive) in the character array that the label spans over and an arbitrary string that identifies the label (e.g. ”exon” or ”intron”). A user provides labeled sequences to the labeler. If there are any existing labels in the labeled sequence, it does not re-label them. Instead, it extracts the parts of the sequence that are not labeled and executes the Viterbi algorithm, powered by an underlying HMM, to label those parts. The result is a character sequence that is fully labeled. In other words, the first label in the labeled sequence's list of labels has a start index of 0, and the last label has an end index of length - 1 of the character sequence, and all of the labels are contiguous.

The underlying HMM consists of several states, where each state represents a specific label. Each state emits characters with some probability distribution function. We use a 3rd party library [3] to keep track of the state of the HMM and to run the Viterbi algorithm. The HMM library stores transition and emission probabilities.

The labeler is capable of machine learning to increase its accuracy using a supervised learning algorithm. This happens by giving the labeler a labeled sequence. The labeler reads in the sequence and identifies the parts of the character sequence that have a label spanning them. The labeler then calculates frequencies and probabilities of label transitions and characters and then computes new transition and emission probabilities. These new probability distributions are merged with the existing ones contained in the HMM by using a weighted average. The weight in this average is one of the input variables and so can be made to reflect some significant information (for example the length of the training labeled sequence).

The HMM can be loaded from a text file using a custom XML schema. The XML file specifies the number of states, label names, transition probabilities, and emission probabilities. From the GUI the user has the opportunity to load an arbitrary HMM XML file to start using their HMM. In our current implementation, there is no functionality for a user to save the current HMM to this XML file. Although such functionality would not be difficult to implement, time alotted for the project did not permit. An example of an HMM that consists of two states, ”exon” and ”intron”, is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>

<hmm>

    <state label="exon" initialProbability=".5">
        <transitionProbability from="exon">.8</transitionProbability>
        <transitionProbability from="intron">.2</transitionProbability>
        <probabilityDistribution a=".15" c=".35" g=".35" t=".15"></probabilityDistribution>
    </state>

    <state label="intron" initialProbability=".5">
        <transitionProbability from="exon">.2</transitionProbability>
        <transitionProbability from="intron">.8</transitionProbability>
        <probabilityDistribution a=".25" c=".25" g=".25" t=".25"/>
    </state>

</hmm>
```

2.5 Aligner

The aligner component of HAGA is responsible for aligning two labeled sequences together. The aligner is given two labeled sequences and first separates each sequence into two groups, parts of the sequence that are unlabeled, and parts of the sequence that have user-defined labels. To begin, the aligner first aligns the sequences that are within user-defined labels using an implementation of a global alignment algorithm. HAGA uses an implementation of the Needleman-Wunsch algorithm. All sequences labeled with user-defined labels are aligned in this way. This requires that the input labeled sequences each have an equal number of user-defined labels, and the order of each list matches (for example if one labeled sequence contains ”exon”, ”intron”, ”exon”, ”exon”, all the other labeled sequences must have this same list of labels). Each pair of matching labels is aligned using an instance of the Needleman-Wunsch algorithm that is appropriate for that label. This implies that each label has a separate instance of the Needleman-Wunsch algorithm. Each instance has its own score matrix and gap penalty function trained specifically for a given label. After all of the pairs of user-defined labels have been aligned, the aligner uses the labeler to label the rest of the sequence (the parts not labeled by user-defined labels).

The aligner then runs a modified version of the Needleman-Wunsch algorithm. The modification is that when computing the score or gap penalty, the labels of the current characters are considered. If the two labels match, a favorable score bonus is given to the match score or gap penalty so that the algorithm favors aligning two matching labels. This bonus can be adjusted so that for a low bonus, the algorithm behaves like standard Needleman-Wunsch, and for a high bonus, the algorithm simply tries to align the sequences based on their labels, ignoring base types.

The aligner can be trained on the user-defined local alignments. The goal of this training is to improve alignments by adjusting the score matrix for bases with the same label. In our implementation, aligner training follows a simple statistical model where the probabilities of each pairing are computed and then scaled by a constant factor to produce a score matrix.

3 Evaluation & Results

Our first method of evaluation was to use the labeler's HMM to generate a sequence of switching states (for example "exon" and "intron"). That sequence would then "evolve" by having evolutionary changes made to it using another probabilitic model (i.e. substitutions and insertions/deletions). The result would be two sequences that have some evolutionary differences between them. Using these sequences we would compare the standard Needleman-Wunsch alignment with our own alignment and use the edit distance between the alignments and the original to quantify the results. Unfortunately, due to restrictions in time, we did not get a chance to complete the evolutionary model and make this comparison. We did however qualitatively compare the alignments made by standard Needleman-Wunsch alignments, and our alignments. What we found was that in some cases, our alignment performed better. This was in the cases where the pre-labeling of the sequence in our algorithm aided the alignment. An example of this is shown below:

```
Standard Needleman-Wunsch:
    -*C-----GC----CTA--CG--G-G---GCCGAC--C-AG-CG*--TA-CGAGACAGT----...
    ACGTGTAGCATAGCTATGCGATGAGAT*CGCCTACGTCGGGCCGACCAGCG*AGACAGTACAG...

 Our alignment:
    eeeeeeeeeeeeeeeeeeeeeeeeeee eeeeeeeeeeeeeeeeeeeeeee iiiiiiiiii...
    ---------------------------*CGCCTACG--GGGCCGACCAGCG*-TACGAGACAG...
    iiiiiiiiiiiiiiiiiiiiiiiiiii eeeeeeeeeeeeeeeeeeeeeee iiiiiiiiii...
    ACGTGTAGCATAGCTATGCGATGAGAT*CGCCTACGTCGGGCCGACCAGCG*AGACAGTACAG...
```

Notice the extra lines in our alignment. These lines of e's and i's represent the label currently assigned to the nucleotide that it is above. The sub-sequence enclosed in *'s delimit the matching sub-sequence of interest. You can see in the example that our alignment performs much better in aligning these sub-sequences. This is because the labeler tries to keep the labels aligned with each other. This results in aligning nucleotides of one label with gaps of another label rather than mixing nucleotides of different labels. In the Needleman-Wunsch case, the alignment algorithm just spreads out the alignment regardless of label.

We also performed evaluation by taking a gene sequence from an online database and annotated it using GENSCAN [1]. This sequence, of a gene on the human chromosome 1, was about 4600 base pairs long and was annotated with exon and intron labels. As a way to evaluate our Labeler, the gene sequence was used to train our labeler's internal HMM. We then ran the Labeler on this 4600 base pair sequence to see how different our labeler would label it. It performed poorly–the entire sequence was labeled as "exon", where the original sequence contained about 3 exons and the rest were introns. This was probably due to the fact that there weren't many examples of exon to intron transitions and so the transition probability from one state to the other was low. Our conclusion was that our labeler would require considerable training in order to perform at GENSCAN's level.

Nielson's 10 heuristics for user interface design provide a way to gauge the usability of a UI [4]. Below are the subjective results when evaluating our user interface against Nielson's heuristics. In summary, the interface is generally efficient, simple, and powerful but lacks documentation and error handling.

1. Visibility of system status: The user interface is generally acceptable in this measure, as all changes to the system model are immediately reflected in the GUI. Visibility could be improved by adding a busy cursor or status bar when performing longer tasks such as loading an HMM labeler.

2. Match between system and the real world: The interface does not match the real world well, but this is to be expected given the representation of the data.

3. User control and freedom: User control is somewhat limited in that the user does not have a large number of control options.

4. Consistency and standards: The interface is consistent in its terminology and behavior. It follows the convention of representing nucleotides with their one-character abbreviations.

5. Error prevention: The interface falls short in this category, as it does not enforce proper ordering of steps. The user can press buttons out of order and cause errors.

6. Recognition rather than recall: To improve in this heuristic, the dialog for creating labels should show a list of possible labels instead of requiring the user to type the label name in each time.

7. Flexibility and efficiency of use: The interface is subjectively fairly efficient for the task of labeling DNA strands, although the application of an existing label could be improved.

8. Aesthetic and minimalist design: The design is minimalist and uses reasonably aesthetic coloration.

9. Help users recognize, diagnose, and recover from error: This heuristic was neglected due to time constraints and the scope of the project.

10. Help and documentation: Similar to the above heuristic, this heuristic was neglected.

4 Contributions

HAGA acheives the fusion of human capabilities and computational power to gene sequence alignment. Our graphical user interface developed for HAGA allows the user to contrain the global alignment algorithm to ensure human knowledge is taken into account. It also acts as a control panel to allow the human to control the start and stop of any algorithm and other functionality. HAGA implements a novel alignment algorithm that calculates scores not only based on matching nucleotides, but also on their associated labels. This algorithm can give hints to the standard Needleman-Wunsch algorithm for a better alignment by pre-labeling the sequence before running the global alignment algorithm. Another notable feature of HAGA is its pluggable software architecture–labeling algorithms can be swapped for others, users can load previously trained HMMs, and alignment algorithms can be tweaked using different score matrices and gap penalty functions.

5 Conclusion & Future Work

During the creation of HAGA, we came across many challenges. Most of these challenges boiled down to the fact that labeling and alignment are fundamentally different things. We had issues of allowing the user to run an alignment twice. This was because during the first alignment, the sequences that were algined would contain dashes to represent gaps. These needed to be removed in order to re-run the alignment or run the labeler because these algorithms did not support the dash character. Nevertheless, we believe that HAGA provides a new dimension in sequence alignment–one that allows the human to intervene in a predominantly computational task. In the future we would like to extend the functionality of HAGA to allow the user to persist an existing HMM to disk, allow the loading/persisting of gap penalty functions via a scripting language, and finally to allow the human to make subtle changes to the alignment algorithm in real-time along side the running algorithm. The last improvement could make for a much more enjoyable user experience because the user feels more empowered and the alignment process would be made much more transparent. In addition to these improvements, we want to pursue further evaluation of our algorithms by quantitatively comparing it with other standard alignment algorithms and effective labelers (such as GHMMs).

References

[1] The new genscan web server at MIT. `http://genes.mit.edu/GENSCAN.html`.

[2] Michael Brudno, Chuong B. Do, Gregory M. Cooper, Michael F. Kim, Eugene Davydo v, NISC Comparative Sequencing Program, Eric D. Green, Arend Sidow, and Serafim Batzoglou. LAGAN and Multi-LAGAN: Efficient tools for large-scale multiple alignment of genomic DNA. *Genome Research*, 13:721–731, 2003.

[3] Jean-Marc Francois. Jahmm - an implementation of HMM in Java. `http://www.run.montefiore.ulg.ac.be/~francois/software/jahmm/`.

[4] Jacob Nielson. Ten usability heuristics. `http://www.useit.com/papers/heuristic/heuristic_list.html`.