

# A New Approach for Gene Annotation Using Unambiguous Sequence Joining

Alexandre Tchourbanov\*, Daniel Quest\*, Hesham Ali\*,  
Mark Pauley\* and Robert Norgren\*

\* Department of Computer Science, College of Information Science and Technology,

University of Nebraska at Omaha, Omaha, NE 68182-0116, achurbanov@mail.unomaha.edu

\* Department of Genetics, Cell Biology and Anatomy, University of Nebraska, Medical Center, Omaha, NE 68198

## Abstract

The problem addressed by this paper is accurate and automatic gene annotation following precise identification/annotation of exon and intron boundaries of biologically verified nucleotide sequences using the alignment of human genomic DNA to curated mRNA transcripts. We provide a detailed description of a new cDNA/DNA homology gene annotation algorithm that combines the results of BLASTN searches and spliced alignments. Compared to other programs currently in use, annotation quality is significantly increased through the unambiguous junction of genomic DNA sequences. We also address gene annotation with both non-canonical splice sites and short exons. The approach has been tested on the Genie learning subset as well as full-scale human RefSeq, and has demonstrated performance as high as 97%.

**Key words:** Spliced alignments, BLAST, RefSeq, dynamic programming

## 1 Introduction

Correct gene annotation is essential for biology in the new millennium. A vast amount of nucleotide-level data has been deposited into databases worldwide, including most of the human genome [3]. Genomic data requires meticulous interpretation and annotation to extract meaningful data. This process is not keeping pace with the amounts of data deposited daily [10]. There is still a real need for accurate and fast annotation tools. The most widely used and precise methods of annotation are based on DNA/DNA, DNA/mRNA and DNA/Protein homology [10].

Pioneered by Gelfand *et al.* [8] with Procrustes, many approaches based on similarity searches have emerged in the last 6 years. The principle behind most of these programs is to combine similarity information with signal information obtained by signal sensors in order to find region boundaries. These programs inherit all the

strengths and weaknesses of signal sensors, and may fail when a non-canonical splice site is encountered [10, 2]. Homology-based programs combining alignment and splice site information are often referred to as *spliced alignment* programs. Existing software may be classified according to the homology type they explore (DNA/DNA, DNA/mRNA and DNA/Protein). Frequently, programs combine several homology types.

We have developed a new program, GIGOGene, to address the problem of correct human genome annotation. In our approach we combine several information sources, such as alignment, sensors, unambiguous allocation and all-pairs-longest-path sequences connection to raise gene annotation quality.

We introduce our homology algorithm by reviewing previous work in the following section.

## 2 Definitions and previous work

Definitions:

**BLAST search:** The BLASTN algorithm was used to search DNA database compiled from Primate (PRI) and High Throughput Genomic (HTG) sequence flatfiles downloaded from NCBI.

**Query sequence:** Highly curated mRNA transcripts originating from NCBI Reference Sequences (RefSeq).

**Query Result:** The set of alignments of transcripts with genomic DNA; the genomic sequences identified presumably contain a gene or a set of candidate exons.

**High-scoring Segment Pair (HSP):** Statistically significant alignment between segments in DNA and mRNA obtained from BLASTN. There are several parameters characterizing HSPs: location in the mRNA query and DNA result sequences as

well as different quality values, such as expectation value (E), percent identity, and score.

*Segment*: Portion of an HSP that may contain an exon in the genomic DNA sequence and that therefore requires further investigation.

*Test case*: Fc fragment of IgG binding protein (FCGBP) transcript NM\_003890 aligned to genomic clones AC00784.1, AC011536.6 and AC006950.1.

*Test set*: First 200 mRNA transcripts of the Genie data set (<http://www.fruitfly.org/sequence/human-datasets.html/>).

Many approaches have been taken towards solving the genome annotation problem, including Procrustes [8], GeneSeqer with SplicePredictor [16], the popular Sim4 [7] and GENOA (proprietary to the Burge Lab annotation script) mentioned in [6], GenomeScan [18], SPIDEY [17] and est2genome [12]. Here is a brief description of each program:

**Procrustes** finds the set of *candidate blocks* that supposedly contain all the true exons. This is done by selecting all blocks between potential acceptor and donor sites (i.e., between AG and GU dinucleotides) with further filtering of this set (in such a way that actual exons are not lost). The program then explores all possible block assemblies using a modified Needleman-Wunch algorithm with multiple tables (the sequential rule mentioned in Section 3.3). This patented software is no longer supported.

**GeneSeqer and SplicePredictor** use alignment algorithm implemented in terms of a Hidden Markov model. The approach uses global alignment through HMM without an affine gap penalty and sensor information. We were unable to annotate our test case correctly using the software.

**Sim4** does a very good job finding the potential splice sites for mRNA/DNA alignment, which is directly comparable to performance of our program. It finds all the sorted sets of HSPs in BLASTN-like fashion, disambiguates the sets to find the biggest unambiguous one and then applies the GT/AG rule to find the splice sites. The spliced alignment in this program is implemented in terms of multiple heuristics. The program predicted all the exonic structure and boundaries correctly for our test case.

**Spidey** does the BLASTN alignment of mRNA against DNA first. The BLASTN HSPs are sorted by score and then assigned into windows by a recursive function, which takes the first alignment and then goes down the

alignment list to find all alignments that are consistent with the first, etc. It then follows a set of filtering procedures and heuristics based on a greedy approach, resulting in exon structure prediction. The program partially failed on our test case.

**GenomeScan** is a combination of a probabilistic *ab initio* algorithm with homology DNA – Protein BlastX alignment information. The program is a tradeoff between quality and speed. Its prediction is frequently imprecise in terms of splice sites [15], and can not be directly compared to the pure homology method we use.

**est2genome** returns a set of exons resulting from the alignment of mRNA to DNA combined with the GT/AG rule. The program makes three alignments: first it compares both strands of the spliced sequence against the forward strand of the DNA, second it realigns the maximum-scoring orientation assuming the splice consensus CT/AC (i.e. in the reversed gene direction), and third it reports the overall maximum-scoring alignment. The program produced the correct results on our test case.

Other genome annotation software is described in [10, 11]. In our tests we have found that many of these methods have components of the correct solution, but don't provide a tool that can be applied to an genome. We thus decided to develop software able to run in batch mode that corrects the following problems with existing packages:

- Many programs use suboptimal heuristics that frequently result in unexpected behavior. Spidey was fooled with the test case alignment NM\_003890 versus AC011536.6, demonstrating occasionally inconsistent performance with repeating domains within a transcript.
- According to [1], the genome annotation error rate is at least 8%. New tools are needed to increase the precision (NCBI uses GenScan and GenomeScan extensively).
- Given several options of unambiguous HSP ordered set construction, described in Section 3.1, current homology-based gene annotation programs may incorrectly assign exons due to repeated sequences within the transcript. Specifically, these programs assume that the largest unambiguous set of HSPs will always correctly identify exons within a genomic sequence; this assumption is not always true.
- BLAST-based software suffers from the high granularity of the BLAST, which results in missing short exons. With the alignment for myosin binding protein C, cardiac (MYBPC3) gene (transcript NM\_000256.2 aligned to genomic clone Y10129, from the test set), containing two small exons of size 2 nucleotides, we have encountered the following problems:
  - Sim4, GenomeScan, est2genome and

Spidey missed short exons due to BLASTN granularity;

- GenomeScan keeps the other splice boundaries correct, while Sim4, est2genome and Spidey mistakenly change the adjacent boundaries due to misalignment;
- est2genome missed an exon after the short one.
- Non-canonical splice sites, such as AT/AC [2], are not really handled by any program yet [10].

To correct these problems, we developed a package that is at least as accurate as current methods and robust enough to run automatically over the entire genome. Since the plain affine gap penalty alignment is prohibitively expensive, running in  $O(n \times m)$ , we use information from the BLASTN search to find possible splice locations. The following section introduces the sequence of steps behind the heuristic.

Our gene annotation software has the following new features:

1. The ability to parse BLASTN output, since our program is designed to run in a batch mode for the whole human genome;
2. The ability to find exons much smaller than 12 nucleotides;
3. We increase annotation quality by considering an optimal junction of several ordered unambiguous sets of HSPs;
4. The spliced alignment we use is noise-tolerant (may recover from certain sequencing errors);
5. It addresses the AT/AC splice rule.

### 3 Total assembly process

We provide a description of the detailed process for gene annotation. The process is divided into 8 steps as explained below:

1. Run BLASTN, with the low-complexity BLASTN filter turned off, querying curated RefSeq mRNA sequences against a DNA database extracted from NCBI HTG and PRI flatfiles.
2. Parse the BLASTN output and identify sequences that score above a certain threshold. These sequences are the most probable candidates for the solution, containing potential exons.
3. Exclude HSPs that are subsegments included into bigger matching segments. Certain noise tolerance required.
4. Disambiguate the ordered sets of HSPs. Each sequence may result into several unambiguous ordered sets of HSPs, as explained in Subsection 3.1.
5. Build an interval graph of overlapping unambiguous ordered sets of HSPs. Special attention is paid to the continuity of mRNA segments, following the rules in Section 3.1. Edges between HSP sets coming from the

same sequence are not allowed.

6. Compact the interval graph. This way we identify the biggest composite sequence, containing the maximum number of possible exons. For these purposes we run the all-pairs-longest-path algorithm, presented in Subsection 3.2.
7. Use spliced alignment to identify possible intron boundaries in the DNA sequence. In order to save running time, we use *anchors* - short nucleotide sequences from mRNA and DNA that should contain exon/intron boundary fragments with donor/acceptor signals. A normal anchor does not have mismatches in  $M$  state (see Section 3.3). Once we have a mismatch, it may mean a short exon is present. If we need to, we can expand the anchor and rerun it with two full exons with an intron between to identify possible short exons as explained in Subsection 3.3.
8. Extract exonic structure from the ordered set of introns and write exon/intron structure into the database in a FASTA format.

Some algorithmic steps are self-explanatory, while others are considered in the following subsections.

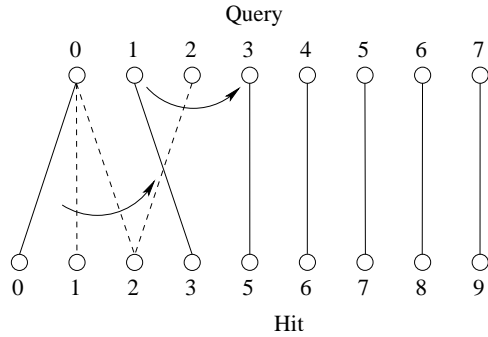
#### 3.1 Algorithm for an ordered set of HSPs allocation

The problem of disambiguating the ordered set of HSPs appears to be a part of gene assembly from several DNA sequences. Parts of a gene, or the entire gene itself, may get replicated during evolution, thereby introducing an ambiguity during the gene assembly process.

There are several fundamental principles from molecular biology that we use in our program:

1. Since transcripts are always linear, we require the set of HSPs to follow this sequential rule, i.e. we use an *ordered HSP set*.
2. Splicing of the pre-mRNA does not introduce any transpositions. Alternative splicing also does not affect the order of the HSPs.
3. Homologous fragments of a gene usually are not biologically protected by natural selection and gradually degrade from the initial functional copy due to random mutations. As a result, a real HSP usually gets a higher identity score, unless the replication was quite recent.
4. When we have an ordered set of HSPs, we require the whole mRNA transcript to be covered with segments continuously, without breaks, otherwise we stop the structure assembly at the break point and search for an alternative solution.
5. There may be some segments within a transcript which contain subsegments. Since an HSP may not span significantly further than exon boundaries, we assume that all segments that are contained in the biggest one,

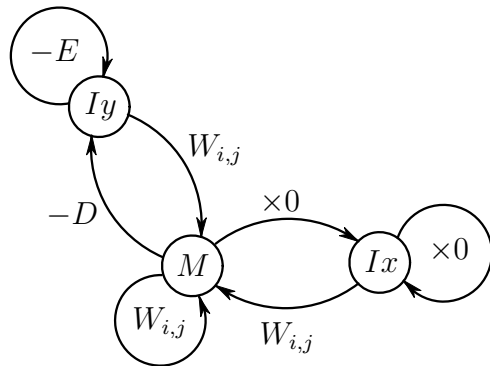
- for this particular HSP, are false copies.
- Unambiguous ordered set of HSPs with significant amount of segments of quality lower than 98% most probably should be rejected as a false set resulting from replication.



**Figure 1. Idea behind disambiguating algorithm**

The disambiguating dynamic programming algorithm to find unambiguous ordered sets of HSPs was implemented according to these principles.

The dynamic programming procedure searches for the largest, unambiguous, continuously-overlapping on the transcript side, an ordered set of HSPs with the maximum score possible. We may skip several segments on genomic clone side in favor of getting the largest ordered set of segments on mRNA side, schematically shown in Figure 1.



**Figure 2. State Diagram of the disambiguating algorithm**

For the purposes of the disambiguating algorithm we need to build a bipartite graph structure, where segments are nodes and HSPs are edges connecting the ordered sets of nodes. Examples of this type of structure are represented in Figure 5 - Figure 7 for the test set mentioned in Section 2.

For each HSP returned by the BLAST search, we need to distinguish between repeats and partial gene copies and real sequence matches. This is accomplished via a combination of the sequential rule and an exponential decay scoring function. For each HSP set we often have copies and real sequence homology. The real sequence homology HSPs are classified as real and the copy regions are classified as copies through this scoring function.

$$f(x) = Km^x, \quad (1)$$

Here  $x$  is the percent off the perfect score. For example, if the score is 98% the value of  $x = 2\%$ . Also  $K$  is the scaling constant, in our case 100, and  $m$  is the exponential function rate of descent.

Result of the scoring function (1) is called *weight*. Weight is an important characteristic for the scoring process in general. Average weight of a set of HSPs will let us make the right decision on the set's future, since the low-scored sets get dropped.

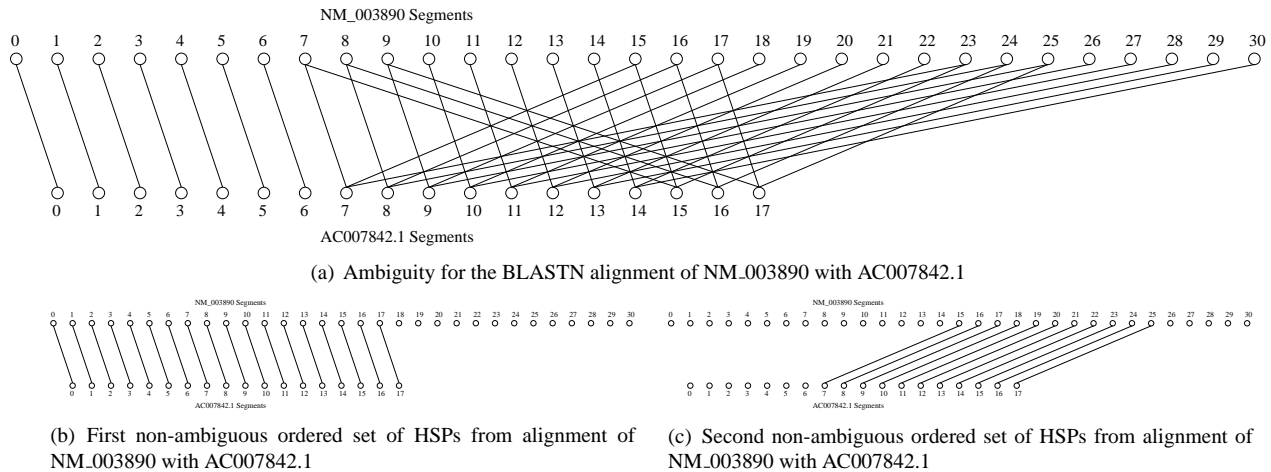
For the disambiguating algorithm shown in Figure 3, we used dynamic programming with an affine gap penalty. We may break the continuous order of genomic DNA segments, visiting  $Iy$  state on the state diagram shown in Figure 2. We loose the score once we break transcript ordered set of segments, resetting the whole score back to 0, visiting  $Ix$  state in Figure 2. Once we have a continuous ordered set of segments, we gain score at  $M$  state. On the state diagram (Figure 2)  $W_{i,j}$  is the weight of a HSP,  $-D$  is the penalty for missing one or more segments in an ordered set of DNA segments,  $-E$  is penalty for each additional segment missing.

For this algorithm we have to allocate score matrix  $F$  of dimensionality

$$2 \times (\# \text{ of RNA segments}) \times (\# \text{ of DNA segments})$$

and matrix  $C$  of the same size to record the intermediate ordered sets in the dynamic programming procedure. For the convenience of indexing, we introduce aliases  $M \leftarrow F_0$  and  $Iy \leftarrow F_1$ . We ignore matrix  $Ix$  as being unnecessary. The binary function  $\text{CONNECTED}(i, j)$  indicates whether we have overlap between segments  $i$  and  $j$  in the transcript.





**Figure 5. Disambiguating the ordered set of AC007842.1 HSPs**

Figure 9 shows the dynamic programming procedure, after we initialize matrix  $D$ .

Upon completion the procedure, we extract maximum element from matrix  $D_n$  and recover the solution. The modified pairwise Floyd-Warshal is shown in Figure 9.

```

FLOYDWARSHALL( $D, n$ )
//  $n$  - The number of nodes
for  $k = 1$  to  $n$ 
  for  $i = 0$  to  $n - 1$ 
    for  $j = 0$  to  $n - 1$ 
       $D_{k,i,j} = D_{k-1,i,j}$ 
      if ( $D_{k-1,i,k-1} \neq \emptyset \wedge D_{k-1,k-1,j} \neq \emptyset$ ) then
        // Try all possible connections between fragments
        COMBINATORIALCONNECT( $k, i, j$ )

```

**Figure 9. Modified Pairwise Floyd-Warshal**

Under certain circumstances, an attempt is made to connect two sets of unambiguous ordered sets of HSPs that contain different unambiguous ordered sets of HSPs from the same sequence. We can't connect these two fragments, since it restores the ambiguity. In this case, the program backs up and searches for possible connections resulting in a better structure, as shown in Figure 10.

```

COMBINATORIALCONNECT( $k, i, j$ )
// If there was a connection with an unambiguous ordered
// set of HSPs from a sequence with the same accession
// number we try to reconnect them in a better way
for  $s = k - 1$  down to 0
  if  $D_{s,i,k-1} = \emptyset$  break
  for  $t = k - 1$  down to 0
    if  $D_{t,k-1,j} = \emptyset$  break
    // If the unambiguous ordered sets of HTGs originate
    // from different sequences, then combine them
    if CANCONNECT  $D_{s,i,k-1}, D_{t,k-1,j}$  then
       $r = D_{s,i,k-1} \cup D_{t,k-1,j}$ 
      if ( $D_{k,i,j} \neq \emptyset \wedge r \neq \emptyset$ )
        // Is it the best move???
        if SIZE  $D_{k,i,j} < \text{SIZE}(r)$  then
           $D_{k,i,j} = r$ 
    else
       $D_{k,i,j} = r$ 
  else if  $s = t \wedge i = k - 1 \wedge j = k - 1$  then
     $D_{k,i,j} = D_{s,i,k-1}$ 

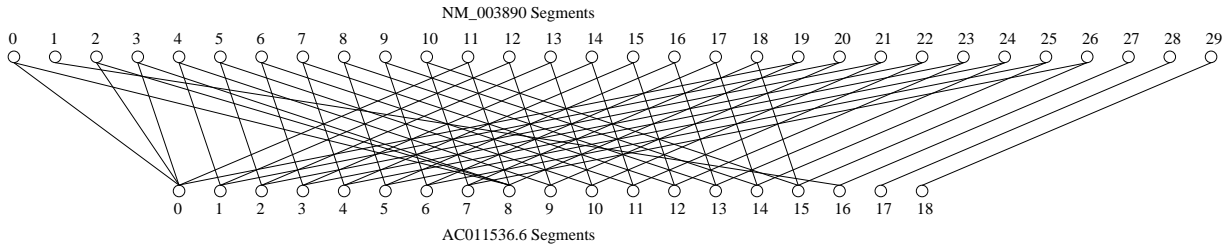
```

**Figure 10. Find the best combination of ordered HSP sets to connect**

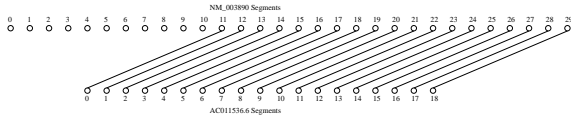
### 3.3 Spliced Alignment algorithm (Divide et impera!)

In order to identify precise intron/exon boundaries, we use modified Needleman-Wunsch affine gap penalty algorithms, commonly known as *Spliced Alignment*. The purpose of a *Spliced Alignment* is to combine both the alignment information and sensor information into optimal alignment [10].

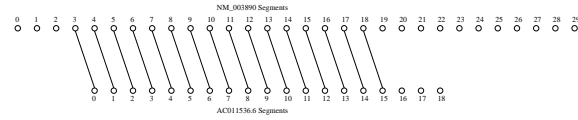
The problem with plain Needleman-Wunch alignment is a scattered alignment pattern, with no indication of exon/intron boundaries. Optimal score alignment may not correctly indicate GT/AG or AT/AC splice patterns based



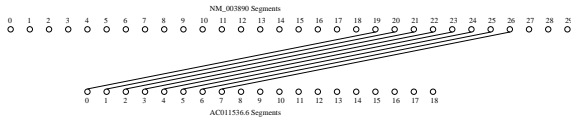
(a) Ambiguity for the BLASTN alignment of NM\_003890 matching AC011536.6



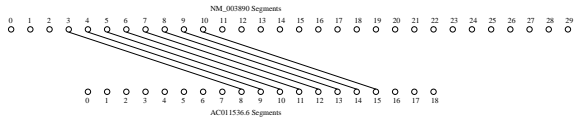
(b) First non-ambiguous ordered set of HSPs from alignment of NM\_003890 with AC011536.6



(c) Second non-ambiguous ordered set of HSPs from alignment of NM\_003890 with AC011536.6



(d) Third non-ambiguous ordered set of HSPs from alignment of NM\_003890 with AC011536.6



(e) Fourth non-ambiguous ordered set of HSPs from alignment of NM\_003890 with AC011536.6

**Figure 6. Disambiguating the ordered set of AC011536.6 HSPs**

on biological rules. With affine gap penalties, we penalize score each time we break an alignment [5]. Affine gap penalty alignment gives more consistent results, but still, most of the time we do not see exact exon/intron boundaries. An addition of sensor information (GT/AG or AT/AC rules [2]) results in precise spliced alignment.

There were at least several published spliced alignment algorithms mentioned in Section 2. Our implementation is a further modification of affine gap penalty algorithm, mentioned in [5].

Our version of Spliced Alignment algorithm could be explained in terms of transitions between states in Hidden Markov Model (HMM) [5]. There are 13 matrices of size  $n \times m$  introduced, corresponding to states, as shown in Figure 11. The matrices are reduced to linear size array  $F$  of size  $2 \times 13 \times m$ , since we need only two rows in scoring matrix  $F$  and backtracking matrices [9, 13].

In our algorithm we introduce the match state  $M$ , which uses scoring matrix shown in Table 1. The same PAM matrix is used in the BLAST. The state  $Iy$  corresponds to extending penalty in the mRNA sequence, while the rest of the states form gaps in the DNA sequence, some having nucleotide-specific score deduction.

We introduce several gap opening penalties. Gap opening matrices  $dA$  and  $dG$  express score preference to open gap with either matrices  $A$  or  $G$ , correspondingly.  $d$  is a generic gap opening penalty, and  $e$  is a generic gap extending penalty.

Gap extension penalty matrices  $eA$ ,  $eC$ ,  $eG$  and  $eT$  express scoring preference to extend gap with nucleotides  $A$ ,  $C$ ,  $G$  and  $T$ .

According to our model, introducing or extending a gap should be easy if we are inside of the GT/AG rule. The penalty becomes high if we try inserting gap outside of the rule, we would rather try using high-extension-penalty  $Ix$  state. The AT/AC rule works the same way, except insignificantly higher penalties for being infrequent.

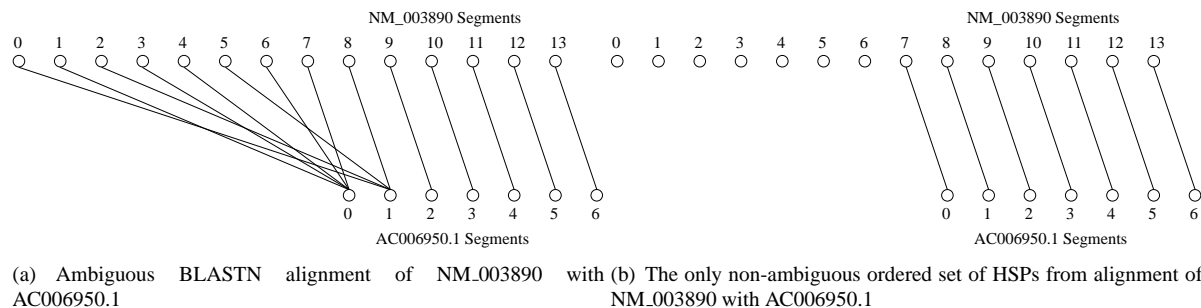
	$A$	$G$	$T$	$C$	
$s =$	$A$	5	-4	-4	-4
	$G$	-4	5	-4	-4
	$T$	-4	-4	5	-4
	$C$	-4	-4	-4	5

**Table 1. PAM matrix**

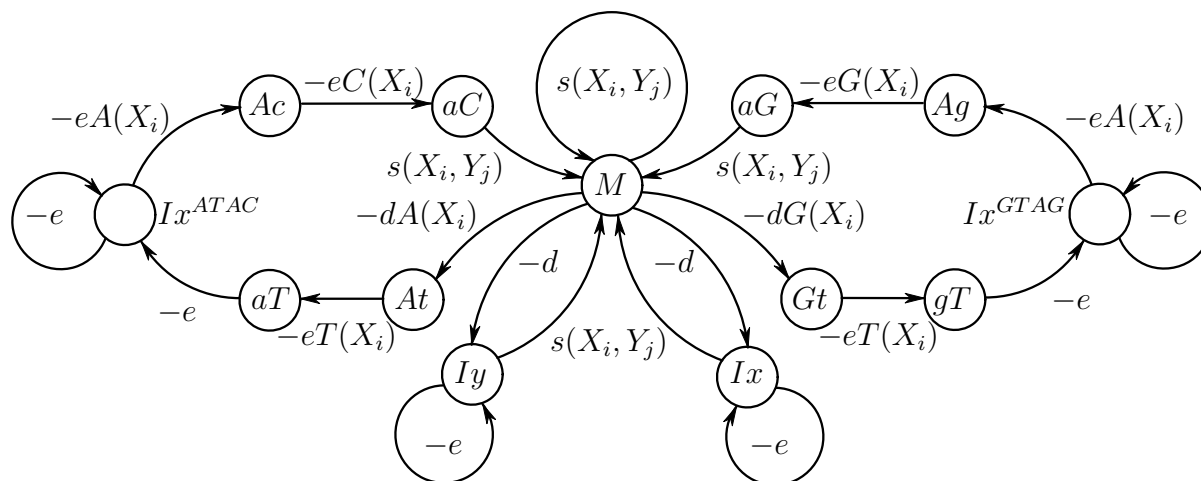
We implement the spliced alignment algorithm in a linear memory of size  $S(m+n)$  and running time  $O(n \times m)$ , where  $n$  is the size of a DNA fragment,  $m$  is the size of an mRNA fragment.

There are following steps in implementation of the algorithm:

1. Run the spliced alignment  $ALIGN(0 \dots n, 0 \dots m)$  alignment to find indexes of  $u$  and  $v$  - the split points for a recursive call. Here  $u$  is the vertical median index, and  $v$  is the horizontal index of a point where the optimal traceback intersects the median.



**Figure 7. Disambiguating the ordered set of AC006950.1 HSPs**



**Figure 11. State Diagram of the Spliced Alignment algorithm**

2. Restore the matrix context for the recursive calls, as well as prior state information for proper backtracking.
3. Make the recursive calls for subsequences  $\text{ALIGN}(0 \dots u, 0 \dots v)$  and  $\text{ALIGN}(u \dots n, v \dots m)$ , etc.
4. If either of the subsequences lengths in a recursive call is less or equal to one, call the ordinary spliced alignment for these pieces to get the alignment states

### 3.4 Experimental results

The program was tested on the test set, defined in Section 2. To measure exon level accuracy we ran our program on the results of BLAST for these 200 records. We estimated the *sensitivity* ( $ESn$ ) and *specificity* ( $ESp$ ) according to the formulas

$$ESn = \frac{TE}{AE} \quad ESp = \frac{TE}{PE}$$

Here  $TE$  is the number of exactly predicted exons (true exons),  $AE$  is the number of annotated exons and  $PE$  is the

number of predicted exons.

Then we calculate additional parameters characterizing the performance on exon level (Table 2). Here  $CRa$  is the proportion of annotated exons that are correctly predicted,  $CRp$  is the proportion of predicted exons that are exactly correct,  $PCa$  is the proportion of partially predicted annotated exons,  $PCp$  is the proportion of predicted exons that are partially correct,  $OL$  is the proportion of predicted exons that overlap the actual exons,  $ME$  is the proportion of missed exons,  $WE$  is the proportion of wrong exons.

$TE$	$AE$	$PE$	$ESn$	$ESp$
1287	1325	1323	0.97	0.97
$CRa$	$CRp$	$PCp$	$OL$	$ME$
0.98	0.97	0.0045	0.0068	0.014

**Table 2. Test results for GIGOfene 1.0**

Both specificity and sensitivity for our program are as high as 97% for the test set.

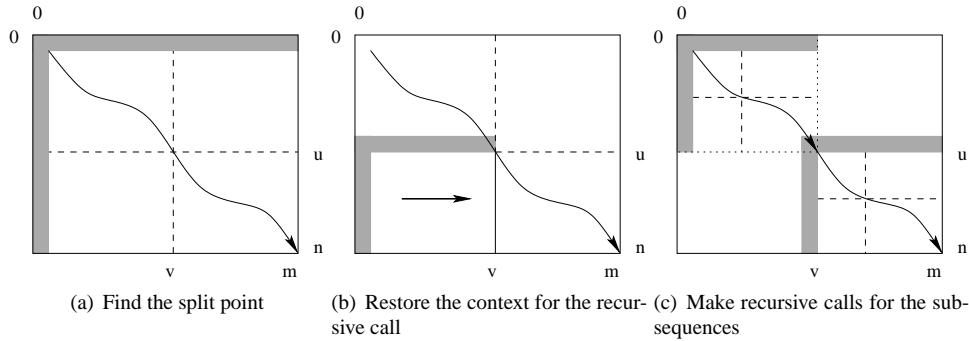


Figure 12. Linear memory spliced alignment algorithm steps

1. Initialize matrices
2. The dynamic programming implementation for the Spliced Alignment
  - for  $i \in 1 \dots n$
  - for  $j \in 1 \dots m$

$$M_{i,j} = \max \begin{cases} M_{i-1,j-1} + s(X_i, Y_j) \\ Iy_{i-1,j-1} + s(X_i, Y_j) \\ Ix_{i-1,j-1} + s(X_i, Y_j) \\ aG_{i-1,j-1} + s(X_i, Y_j) \\ aC_{i-1,j-1} + s(X_i, Y_j) \end{cases},$$

$$Gt_{i,j} = M_{i-1,j} - dG(X_i), gT_{i,j} = Gt_{i-1,j} - eT(X_i),$$

$$At_{i,j} = M_{i-1,j} - dA(X_i), aT_{i,j} = At_{i-1,j} - eT(X_i),$$

$$Ag_{i,j} = Ix_{i-1,j}^{GTAG} - eA(X_i), aG_{i,j} = Ag_{i-1,j} - eG(X_i),$$

$$Ac_{i,j} = Ix_{i-1,j}^{ATAC} - eA(X_i), aC_{i,j} = Ac_{i-1,j} - eC(X_i),$$

$$Ix_{i,j}^{GTAG} = \max \begin{matrix} Ix_{i-1,j}^{GTAG} - e \\ gT_{i-1,j} - e \end{matrix}, Ix_{i,j}^{ATAC} = \max \begin{matrix} Ix_{i-1,j}^{ATAC} - e \\ aT_{i-1,j} - e \end{matrix},$$

$$Iy_{i,j} = \max \begin{matrix} M_{i,j-1} - d \\ Iy_{i,j-1} - e \end{matrix}, Ix_{i,j} = \max \begin{matrix} M_{i-1,j} - d \\ Ix_{i-1,j} - e \end{matrix}.$$

3. Recover the solution

Figure 13. General structure of the spliced alignment algorithm

In the test run our program detected all of the internal exon boundaries correctly and one mistake was spotted in the test set annotation. The 3% loss for *ESn* and *ESp* performance values can be attributed to non-coding exon boundaries.

We also ran our program for the whole human genome, we keep updated version of exon/intron database in a FASTA format at <http://bioinformatics.ist.unomaha.edu/~achurban/>.

## 4 Concluding remarks

We have compared several homology-based programs. Of the currently available programs, Sim4 was the most accurate for our test set. Using a similar approach, we have been able to improve on Sim4 by addressing several possibilities of error.

We have designed a program for the genome annotation. The program has been tested on the Genie subset with the correctly predicted splice sites ratio as high as 97%. We also ran our program on the whole human genome in a

batch mode. The program, test results, manual and the current human exon/intron database are available on the web at <http://bioinformatics.ist.unomaha.edu/~achurban/>.

While demonstrating excellent performance for simple high-quality data sets, such as the test set, the program may fail to find a gene annotation due to absence of high quality homologs in a BLASTN results (especially it refers to HTG drafts). The program produces the best combination of data available for correct prediction, however, it still does not warranty the absolute correctness of the results. Since the spliced alignment algorithm we use to find exon/intron boundaries is time demanding heuristic, it may take few minutes for the program to annotate a certain transcript (on Pentium III processor).

We have already used the human exon/intron database generated by the program to extract 75,000 donor and acceptor motifs for our research projects.

## 5 Acknowledgement

We are thankful to Xutao Deng for finding first 200 transcript accession numbers for the test set. Members of the Bioinformatics group at the University of Nebraska at Omaha provided useful feedback on our progress and program. The work was partially supported by NIH NS44330 to RBN and NIH P20 RR16469 grants.

## References

- [1] S. E. Brenner. Errors in genome annotation. *Trends in Genetics*, 15(4):132–133, Apr. 1999.
- [2] C. B. Burge, R. A. Padgett, and P. A. Sharp. Evolutionary fates and origins of U12-type introns. *Molecular Cell*, 2:773–785, 1998.
- [3] I. H. G. S. Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, Feb. 2001.
- [4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition edition, Sep. 2001.
- [5] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis*. Cambridge University press, 1998.
- [6] W. G. Fairbrother, R.-F. Yeh, P. A. Sharp, and C. B. Burge. Predictive identification of exonic splicing enhancers in human genes. *Science*, 297:1007–1013, 2002.
- [7] L. Florea, G. Hartzell, Z. Zhang, G. M. Rubin, and W. Miller. A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Research*, 8:967–974, 1998.
- [8] M. S. Gelfand, A. A. Mironov, and P. A. Pevzner. Gene recognition via spliced sequence alignment. *Proc. Natl. Acad. Sci. USA*, 93:19061–9066, 1996.
- [9] D. S. Hirschberg. A linear-space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18:341–343, Jun. 1975.
- [10] C. Mathé, M.-F. Sagot, T. Schiex, and P. Rouzé. Current methods of gene prediction, their strengths and weaknesses. *Nucleic Acids research*, 30:4103–4117, 2002.
- [11] W. Miller. Comparison of genomic DNA sequences: solved and unsolved problems. *Bioinformatics*, 17(5):391–397, 2001.
- [12] R. Mott. EST\_GENOME: a program to align spliced DNA sequences to unspliced genomic DNA. *CABIOS*, 13(7):477–478, 1997.
- [13] E. W. Myers and W. Miller. Optimal alignments in linear space. *Computer Applications in the Bio-sciences*, 4:11–17, 1988.
- [14] P. J. Russel. *iGenomics*. Benjamin Cummings, 2002.
- [15] A. Tchourbanov. Research proposal, Jan. 2003. Available at <http://csce.unl.edu/~tchourba>.
- [16] J. Usuka, W. Zhu, and V. Brendel. Optimal spliced alignment of homologous cDNA to a genomic DNA template. *Bioinformatics*, 16:203–211, 2000.
- [17] S. J. Wheelan, D. M. Church, and J. M. Ostell. Spidey: A tool for mRNA-to-genomic alignment. *Genome Research*, 11:1952–1957, 2001.
- [18] R.-F. Yeh, L. P. Lim, and C. B. Burge. Computational inference of homologous gene structures in the human genome. *Genome Research*, 11:803–816, 2001.