A Dictionary Based Approach for Gene Annotation

Lior Pachter[*,†], Serafim Batzoglou[‡,†], Valentin I. Spitkovsky, Eric Banks,

Eric S. Lander[§], Daniel J. Kleitman[¶], and Bonnie Berger[*,‖]

July 1, 1999

[*]Department of Mathematics and Laboratory for Computer Science, MIT.

[†]L.P. and S.B. contributed equally to this work.

[‡]Laboratory for Computer Science, MIT.

[§]Whitehead Institute and Biology Department, MIT.

[¶]Department of Mathematics, MIT.

[‖]Author to whom correspondences should be addressed. Email: bab@mit.edu. Room 2-389, Department of Mathematics, MIT, Cambridge, MA 02139.

Abstract

This paper describes a fast and fully automated dictionary based approach to gene annotation and exon prediction. Two dictionaries are constructed, one from the nonredundant protein `OWL` database and the other from the `dbEST` database. These dictionaries are used to obtain $O(1)$ time lookups of tuples in the dictionaries (4 tuples for the `OWL` database and 11 tuples for the `dbEST` database). These tuples can be used to rapidly find the longest matches at every position in an input sequence to the database sequences. Such matches provide very useful information pertaining to locating common segments between exons, alternative splice sites, and frequency data of long tuples for statistical purposes. These dictionaries also provide the basis for both homology determination, and statistical approaches to exon prediction.

1 Introduction

One of the main challenges facing biologists today is the accurate annotation of newly sequenced genomic data. In particular, one seeks to determine the coding regions, exons and introns of the genes. One has available for this task very large databases of proteins, EST's and smaller databases of annotated genes.

In this paper we describe a dictionary based approach to gene annotation. The method is best described as applied to a particular database. Using the nonredundant protein `OWL` database [3], our dictionary consists of 4 tuples of amino acids, and for each, the protein sequences in `OWL` which contain it. We can construct such dictionaries very quickly from `OWL` and the `dbEST` databases [2]. We use the `OWL` dictionary to find the longest common subsequences of length at least k, for any $k \geq 4$, between sequences in `OWL` and the translation of the genomic sequence under investigation. A similar approach is applied to the `dbEST` database, after orienting the `dbEST` database and eliminating non-coding exon regions. Given such information we use a dynamic programming algorithm to produce a parse of the gene into introns and exons. In order to predict splice sites we use a splice site predictor that is based on the one used in `GENSCAN` [8], enhanced with the use of a directional effect that we observed. Namely, almost all potential splice sites upstream of a true splice site, are much more unlikely (i.e. score more poorly) than the true splice site.

We have tested our method on two benchmark data sets. On a test set derived from the Haussler–Kulp–Reese benchmark data set [13] (genes with inconsistent annotation were removed, see section 2.2), we find 88% of coding nucleotides (*i.e.* sensitivity), and 98% of our predictions of coding nucleotides are correct (*i.e.* specificity). Also, 81% of coding exons are predicted exactly, while 78% of our predictions of exons agree exactly with the published annotation of their genes. When testing on the Burset–Guigó data set [9], we find 86% of coding nucleotides, with 95% of our predictions correct. We predict 78% of the coding exons exactly, with 73% of our predictions correct. These results are obtained after removing sequences from the database with exact amino acid homology to genes in the data set. Accuracy claims of most current statistical based exon prediction programs range from 60–90% per coding nucleotide, and 30–70% per entire exon [12].

1.1 Related Work

Many computational methods have been developed for the purposes of gene annotation [6]. Indeed, of the many applications of computer science in biology, perhaps the most successful has been the implementation of algorithms for finding similarities between sequences [25]. The most widely used program developed for this purpose is `BLAST` [4, 5], which is an alignment tool. `BLAST` is often manually applied for the purposes of gene annotation, including exon prediction and repeat finding. Other similarity search approaches include the `FLASH` [22] program which is an example of a clever use of a hash table to keep track of matches and positions of pairs of nucleotides in a database. The resulting information can be used to extract close matches to a given sequence. Nevertheless, neither of these search approaches have been designed for gene annotation.

Programs designed specifically to address the problem of exon prediction can be broadly categorized as either statistical or homology based. Statistically based programs include `GENSCAN` [8], `GENIE` [17], `GENEMARK` [19], `VEIL` [14] (all based on hidden Markov models), FGENEH [23] (an integration of various statistical approaches for finding splice sites, exons, etc.), and GRAIL [26] (based on neural networks). Homology based approaches exploit the fact that protein sequences similar to the expressed sequence of a gene are often in databases. Using such a target, one can successfully identify the coding regions of a gene. The alignment based PROCRUSTES [11, 24, 20] program represents a very successful implementation of this idea. When a related mammalian protein is available, this program gives 99% accurate predictions and guarantees 100% accurate predictions 37% of the time; however, the user supplies the target protein sequence. This difficulty is addressed in another recent approach [15]. Specifically, the AAT tool addresses this by automatically using BLAST-like information from protein or EST databases for exon prediction. The INFO program [18] is based on the idea of finding similarity to long stretches of a sequence in a protein database, and then finding splice sites around these regions. Such programs are becoming more important as the size of protein and EST databases increase.

2 Methods

2.1 Dictionary Lookups and Fragment Matching

A central component of our gene annotation approach is the **fragment matching problem**. That is, given a gene and a database (for example the `dbEST` database), we would like to find all the matches of length above some threshold between the gene and the database. This is a classic string matching problem, and there are linear-time algorithms for it. The problem with such an approach is that the size of the databases we are interested in matching against precludes the possibility of real time computation. Instead, we do some precomputation on the database so that we do not have to look at all the sequences in the database whenever we are looking up the matches for a particular gene.

2.1.1 Dictionary Construction

The data structure we precompute is a **dictionary**. Conceptually, the idea is to record for each tuple (either of DNA or of protein, depending on the dictionary one is building), the list of sequences in the database in which it appears.

Formally, a **dictionary** is based on a "plain" sequence file, consisting only of accession codes (identifying codes) and corresponding sequences of strings, from an alphabet of size 4 for DNA and size 20 for proteins. A **tuple** is a sequence of length 11 for DNA sequences and length 4 for protein sequences. A **hit** is a match between some segment in the input sequence and a **target sequence** in the database.

Sequences and tuples are indexed by integers for the purpose of lookups in the dictionary. The dictionary is organized into six components which collectively enable the following operations to be performed in $O(1)$ time:

- Find a sequence given its number.
- List all the sequences that contain a given tuple.
- Find the accession code of a sequence from its number.

Finally, the accession code of a sequence can be used to find the sequence number in $O(\log n)$ time using binary search. The first two sequence and tuple lookup functions are used by the end-user. The last function is a helper utility for enabling the $O(1)$ lookups.

Two dictionaries were constructed, one from the `OWL` database [3] and another from the `dbEST` database [2]. The more difficult step was constructing the tuple lookup table, *i.e.*, for each tuple, a list of the sequences in which it occurs. The naive approach to building the dictionary would be to first construct a matrix indexed by tuples and the accession numbers of the sequences. For instance, in the case of the `dbEST` database this would be a 4 million by 1 million entry binary matrix (the `OWL` database consists of roughly 250,000 sequences). The entries of the matrix would be flagged according to which tuples occur in which sequences, and the dictionary would be built by reading off the entries for each tuple. Unfortunately, this would require too much space (terrabytes) or time (trillions of operations).

Instead, the dictionary was constructed by sorting pairs of sequence/tuple identifiers by the tuple coordinate. Specifically, every occurrence of a tuple in a sequence was recorded as a pair (s_i, t_i), where the s_i's are integers ranging from 1 to the number of sequences, and the t_i's are integers ranging from 1 to the number of tuples. The list was originally ordered by the sequences from the database. A linear time radix sort was used to sort the list according to the second coordinate. The large number of pairs necessitated that the large list was sorted in pieces and then merged at the end. The size of the individual pieces to be sorted was set as a command line parameter, so that the dictionary construction could be tuned to take full advantage of the memory of our machine. The final list of sequence/tuple pairs ordered by tuple numbers was used to look up the sequences in which a specified tuple occurs in $O(1)$ time.

2.1.2 Using the Dictionary to Find Matches

The dictionaries were used to quickly find exact matches of subsequences between a given input sequence and a database. This information was used to compute hits. For a given hit, the following information was returned:

- The position in the input sequence where the hit began.
- The length of the hit.
- The accession number of the target sequence.
- The position in the target sequence where the hit began.

Returned hits correspond to longest segments in the input sequence that matched segments of each target. These hits were also required to be longer than a threshold k. The first three pieces of information were useful for obvious reasons. The position in the target sequence was used to determine if nonadjacent hits in the input sequence corresponded to consecutive segments in the target (thus indicating the presence of an intron).

This information was computed in two phases. In the first phase, the dictionary was used to find, for each tuple appearing in the input sequence, the list of target sequences containing that tuple. The input sequence was then scanned from the beginning to the end to find all segments longer than k that contained tuples from the same target sequence in the database. This resulted in a list of candidate segments.

A second phase was necessary to ensure that the tuples in these candidate segments were actually consecutive in the database sequence which they matched. This was accomplished by loading the database sequence for each candidate segment, and then finding the longest subsegments of the candidate segment appearing in the loaded database sequence. This final procedure was divided into two substeps. The first consisted of building a mini-dictionary, used to return in $O(1)$ time a list of the positions in the sequence where a given tuple occurred. This dictionary was then used in the second step in a manner analogous to the first phase described above to scan through the candidate segment to find all subsegments consisting of *consecutive* tuples in the database sequence.

2.2 Finding parses

A dynamic programming approach was used to find the best parse of a gene. A valid **parse** was defined to be a subdivision of the gene into coding exons and introns satisfying the following requirements:

- The coding region begins with the codon ATG.

- The coding region ends with a stop codon (TAG, TGA, or TAA).

- All donor splice sites contain the GT consensus. All acceptor splice sites contain the AG consensus.

- The assigned frames of the exons must be consistent.

The **best** parse was defined to be the parse with a maximal score, where the score of a parse was defined to be the sum of the scores of the exons. For example, in this paper one simple score for a potential exon was computed by

$$score(exon) = \sum_p f(p)$$

where p ranges over all the positions in an exon. The function $f(p)$ was defined by

$$f(p) = \begin{cases} 1 & h(p) > 8 \\ -1 & 4 < h(p) \leq 8 \\ -2 & h(p) < 5 \end{cases}$$

where $h(p)$ was the length of the maximal hit at position p. Other scoring schemes are being tested.

When using the `OWL` database, the sequence was converted into protein in all three possible frames to look for matches (computed with a threshold of $k = 5$), and it was this information that was used to ascertain the frame of a potential exon.

Parses were constructed by first determining all potential splice sites. This was done using a modified version of the splice site detector described in Burge and Karlin [8]. Specifically, a directional "effect" observed for donor and acceptor splice sites was taken into account.

The effect is that almost always, the score of a false splice site upstream of a true splice site, is smaller than the score of the true splice site. The effect is directional, because it is much stronger to the left than to the right. Given two neighboring splice sites s_1, s_2 with s_2 downstream of s_1, we modified the `GENSCAN` score gs_2 of the downstream splice site (the other splice site having score gs_1 respectively) by setting

$$gs_2' = gs_2 + MIN(0, gs_2 - gs_1).$$

Thus, the score of a potential splice site was penalized, if the next splice site upstream had a better score. Figures 1 and 2 show a detailed analysis of the strength of the directional effect as a function of distance from a true splice site. For a given parameter d and fixed direction (either left or right), we computed three quantities. First we computed $t(d)$, the total number of splice sites at most d away from a true splice site in the fixed direction, with the additional property that they had no intervening splice sites. Then we computed $b(d)$, the number of such splice sites with a better score than the closest true splice site. Finally, we computed $r(d)$, the ratio $\frac{b(d)}{t(d)}$. For example, Figure 1 contains the results computed for donor splice sites. The plots are labeled left and right depending on the direction tested. There is a tremendous difference between the left and right directions in terms of the number of adjacent splice sites with better scores. A similar effect is evident for acceptor splice sites (Figure 2). Interestingly however, the situation is a bit different than for donor splice sites. In particular, there are *very few* potential acceptor splice sites immediately upstream of true acceptor sites. For both acceptor and donor sites, it seems that there is a significant effect up to a distance of 30–40 basepairs away from the splice site. The directional effect at both splice sites strongly suggests that the splicing machinery has a directionality associated with it.

The potential splice sites were then used to dynamically construct a parse using the scoring scheme described above. In addition, repeats were masked (see section 3.4), reducing considerably the locations available for exons.

Tests were conducted on a data set of 570 vertebrate genes, compiled by Burset and Guigó [9], as well as on a data set of 130 genes taken from the Haussler–Kulp–Reese [13] human benchmark test set. The Haussler–Kulp–Reese test set was filtered with the following criteria: Genes were required to have a *single* annotation covering the whole sequence, the CDS annotation was checked for consistency with the annotated nucleotides, and sequences with "unknown basepairs" were removed. The tests were performed once using the entire database, and once where sequences in the database matching all of the exons in the input sequence were removed.

3 Results and Discussion

3.1 Output of the Program

The program gives a list of maximal exact matches of length at least k, which is a parameter one may choose, in the database in question in various frames. The results depend on k and some matches may occur in introns and some in exons. Typically, those in introns tend to be scattered and it is usually easy to distinguish them from those in exons. Table 1 contains an example output obtained for the Id3 gene (an HLH type transcription factor, `GENBANK` Accession: X73428) using the `OWL` dictionary with a minimum threshold of $k = 8$. The number of hits returned was 46. A corresponding table for $k = 7$ contained 440 hits. The table for $k = 6$ contained more than 8000 hits. The frame of a hit was defined by its position in the DNA sequence *modulo 3*. Note that positional information in the target sequence has been omitted. The annotated structure of the gene in `GENBANK` is represented in Figure .

Figure 3 contains a screenshot of a java applet used to visualize genes, as well as various types of information including the the dictionary hits described above.

3.2 Alternative Splice Sites

These methods are particularly amenable to showing where exons can be read in different frames. For example, consider the Id3 gene mentioned above.

Using the protein dictionary, we found that the second exon matched in two different frames. There is an alternatively spliced form of the gene which is shown in Figure 5. In coding exon 3, the entire first intron is used to make protein, and the fact that its length is not divisible by 3 means that coding exon 2 is expressed in a different frame. The alternatively spliced version occurs only 10% of the time [10]. The algorithm we used to find exons in genes, given the protein matches, can be forced to select either alternative and return both answers, thus automatically identifying an alternative splice site. This particular example also illustrates the difficulty in finding a universal "good" target. Even though the alternatively spliced version of the gene is somewhat rare, `BLAST` reports it as a better match to the whole gene because the expressed protein is longer. Furthermore, analysis of the above gene with the `dbEST` dictionary revealed coding exons 1 and 2, and also the noncoding exon, but the alternatively spliced variant was not evident. Thus it can be useful to analyze genes using different databases.

Table 2 contains a list of all the genes we found in the Burset–Guigó data set that contained exons that matched proteins in two distinct frames. The criteria used was that the candidate exon had to have two segments (each at least 24 base pairs long) overlap it in two distinct frames, with the overlap between the segments and the exon being at least half the length of the exon. Furthermore, the overlap between the two segments was also required to be half the length of the exon. The candidate exon was at least 30 base pairs long. The strict criteria were chosen to ensure that the overlapping hits observed in the selected exons were statistically significant.

3.3 Exon Prediction

The results of tests using the entire `OWL` protein database are reported in Table 3 for the Burset–Guigó and Haussler–Kulp–Reese data sets. The sensitivity and specificity are based on the definitions in Burset and Guigó [9]. The results of tests with exact matches of each test sequence removed from the `OWL` database are also shown in Table 3. The results for the Burset–Guigó runs may have been affected by the presence of non-human genes. The parameters for the program were callibrated on a human training set.

The power of combining dictionary hits with a gene recognition program is emphasized by the following statistic (computed with the removal of exact matches of test sequences from the database): Out of the 10% of intron positions covered by matches of 8-tuples to the `OWL` database, only 0.5% were predicted to be in exons. Only .05% of the total intron base pairs were incorrectly classified as coding.

The results in Table 3 compare favorably with other statistical methods. Estimates for sensitivity and specificity per nucleotide position range from 60–90%. Predictions of exact exons also vary between the programs, with estimates between 30–70% specificity and sensitivity. The homology based `AAT` approach predicts nucleotides with a sensitivity of 94% and specificity of 97%, and exons exactly with a sensitivity of 74% and specificity of 78%.

The quality of our results is, of course, directly related to the presence or absence of related matches to our test genes in the database. The larger the minimum tuple length threshold k, the more the results become dependent on the general redundancy of the database. For genes with few matches, one can resort to a smaller tuple size and take the *number* of hits into account. Thus, our approach can be tuned to work either as a statistical method or as a homology based method, as well as all the hybrids in between. Furthermore, as the size of the databases grow, the results can be expected to improve.

3.4 Other Applications

The dictionary approach we have described lends itself to a number of other applications:

- Repeat masking: We have built dictionaries from repeat databases and used these for rapidly finding repeat segments in genes. This technique provides an alternative to alignment based repeat maskers such as `RepeatMasker` (A. F. A. Smit and P. Green [1]). The method is especially useful for exon prediction, where it is advantageous not only to mask complete repeats, but to mask segments (perhaps from repeats) that do not occur in exons.

- Different tuple patterns: The construction of the dictionary can be based on arbitrary tuple patterns and does not need to be restricted to consecutive tuples. Such patterns may be important biologically. For example, the third position in codons is less conserved in exons than the other two, so a pattern skipping every third position may lead to interesting results. Another example is the Kozak consensus [16] for translation starts, which involves positions -3 and $+4$ around the ATG (the consensus is AGXXATGG).

- Pseudogenes: Reverse transcribed genes which lack introns are often pitfalls for gene recognition programs. The identification of neighboring exons in inconsistent frames with no room for an intron immediately suggests the presence of a pseudogene. This can be easily checked and automated, in the same vein as the alternative splicing detection. Indeed, we discovered two such examples in a newly sequenced genomic segment (`GENBANK` Accession: AC001226).

3.5 Discussion

The dictionary approach has a number of advantages over standard similarity search techniques. Despite the uprecedented success of alignment algorithms in biology, the algorithms are all handicapped by the problem that short matching segments can be difficult to find in certain cases [7, 27]. For example, Figure 6 illustrates a case where two subsequences of a large sequence agree in small regions (dark areas) and differ elsewhere. An alignment between them may be overlooked if mismatches are penalized less than gaps. Such a penalty scheme will produce an alignment of the two subsequences with each other where the dark regions are not aligned, rather than the desired alignment where the dark regions are superimposed. Even though this problem can be addressed by suitably modifying the alignment parameters, the number of such extreme examples, combined with the myriad of parameters necessary to address biological phenomena involved in sequence evolution (*e.g.,* `BLAST` [4]), creates a fundamental difficulty. Every given problem has a set of parameters associated with it that provides a "good" alignment, but there is no universal set of parameters that works for every problem. An advantage of our dictionary method is that when performing a database search, *all* the exact matches of segments in a sequence are rapidly detected. The `FLASH` program is also designed to find similarities of segments to sequences in databases. Unlike our dictionary method, which involves $O(1)$ time lookups for tuples of a given size (4 tuples for protein dictionaries and 11 tuples for DNA dictionaries), the `FLASH` program relies upon storing the *positions* of shorter tuples in a hash table. The use of larger tuple sizes in a dictionary renders this unnecessary because longer tuples appear in fewer sequences. The advantages of our dictionary method become apparent when the databases involved become very large, which is the case with the current `dbEST` database. Furthermore, this method enables rapid calculation of frequency counts of arbitrary length tuples, which can be applied to statistics based programs that rely on such information.

This approach also has advantages over other exon prediction methods. The entire process of database search and exon prediction is automated, and the prediction is based on many good target sequences (and their fragments) simultaneously. Furthermore, in contrast to the `INFO` program, exons that have partial or no matches in a database can sometimes be accurately predicted by using the fact that exons are constrained independently in many ways: they require splice sites, must be frame consistent and cannot contain high complexity repeats [21]. Also in contrast to `INFO`, the method takes full advantage of the presence of long *and* short hits. The `INFO` program uses 26 tuples of amino acids, whereas the dictionary based approach can use all the tuple information starting with 4 tuples of amino acids. The `AAT` [15] tool is designed specifically to automate the process of finding a good target sequence. However, the reliance on one good target limits the ability of the program to predict exons from fragments.

The effectiveness of the dictionary method we propose is not at all obvious *a priori.* Indeed, the method can be inferior to alignment in the case where one wants to compare two similar DNA sequences, perhaps from different species. A few gaps and mismatches every 5–6 nucleotides will tend to exclude 11-tuple DNA hits. On the other hand, the method has proved to be very effective for exon prediction, especially when used in conjunction with a protein database (where mutations in the third position of codons do not necessarily alter the resultant protein), or an EST database (where exact matching fragments of a gene provide excellent candidates for the dictionary to find).

3.6 Running Times

The `OWL` dictionary was built on a Pentium 2 (400MhZ) in 2 minutes, and the `dbEST` dictionary, in less than 40 minutes. The total space occupied by the dictionaries was manageable on modern drives. The `dbEST` database occupied about 1.75GB. The analysis of a typical gene (computation of dictionary hits and solution of the best parse) was accomplished on the order of seconds.

The construction of the dictionaries, a routine exercise for small databases, was complicated by the extremely large size of the databases. The solution described in the methods section solves the problem by finding a tradeoff between space and time that enables a realistic solution to the problem.

4 Acknowledgments

We thank Wes Beebee, Ben Cooke, John Dunagan, Nick Feamster, Aram Harrow, Ben Ho, Julia Lipman, Theo Tonchev, Tina Tyan and Bill Wallis for helping in countless ways with the implementation of the ideas outlined in this paper. This project has been supported by Merck. Pachter has been supported in part by an NIH training grant and a Program in Mathematics and Molecular Biology graduate fellowship.

References

[1] RepeatMasker. http://ftp.genome.washington.edu/cgi-bin/RepeatMasker, 1998.

[2] dbEST. http://www.ncbi.nlm.nih.gov/dbEST/index.html, 1998.

[3] OWL. http://bmbsgi11.leeds.ac.uk/bmb5dp/owl.html, 1998.

[4] S. F. Altschul, S. F. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[5] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[6] S. Batzoglou, B. Berger, D. J. Kleitman, E. S. Lander, and L. Pachter. Recent developments in computational gene recognition. *Documenta Mathematica, Extra Volume ICM 1998*, 1:649–658, 1998.

[7] G. Benson. Sequence alignment with tandem duplication. *Journal of Computational Biology*, 4(3):351–367, 1997.

[8] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *Journal of Molecular Biology*, 268:78–94, 1997.

[9] M. Burset and R. Guigó. Evaluation of gene structure prediction programs. *Genomics*, 34(3):353–367, 1996.

[10] R. W. Deed, M. Jasiok, and J. D. Norton. Attenuated function of a variant form of the helix-loop-helix protein, Id-3, generated by an alternative splicing mechanism. *FEBS Lett.*, 393:113–116, 1996.

[11] M. S. Gelfand, A. A. Mironov, and P. A. Pevzner. Gene recognition via spliced sequence alignment. *Proc. Natl. Acad. Sci. USA*, 93:9061–9066, 1996.

[12] R. Guigó. Computational gene identification: an open problem. *Computers and Chemistry*, 21(4):215–222, 1997.

[13] D. Haussler, D. Kulp, and M. Reese. A representative benchmark gene data set. http://www-hgc.lbl.gov/inf/genesets.html, 1996.

[14] J. Henderson, S. Salzberg, and K. H. Fasman. Finding genes in DNA with a hidden markov model. *Journal of Computational Biology*, 4(2):127–141, 1997.

[15] X. Huang, M. D. Adams, H. Zhou, and A. Kerlavage. A tool for analyzing and annotating genomic sequences. *Genomics*, 46:37–45, 1997.

[16] M. Kozak. Possible role of flanking nucleotides in recognition of the AUG initiator codon by eukaryotic ribosomes. *Nucleic Acids Research*, 9:5233–5252, 1981.

[17] D. Kulp, D. Haussler, M. G. Reese, and F. H. Eeckman. A generalized Hidden Markov Model for the recognition of human genes in DNA. *Proceedings of the 4th Conference on Intelligent Systems in Molecular Biology*, 1996.

[18] M. Laub and D. W. Smith. Finding intron/exon splice junctions using INFO, INTer-ruption finder and organizer. *Journal of Computational Biology*, 5(2):307–321, 1998.

[19] A. V. Lukashin and M. Borodovsky. GENEMARK.HMM: new solutions for gene finding. *Nucleic Acids Research*, 26(4):1107–1115, 1998.

[20] A. Mironov, M. A. Roytberg, P. A. Pevzner, and M. S. Gelfand. Performance-guarantee gene predictions via spliced alignment. *Genomics*, 51:332–339, 1998.

[21] L. Pachter, S. Batzoglou, E. Banks, W. Beebee, N. Feamster, V. I. Spitkovsky, T. Tyan, B. Wallis, E. S. Lander, D. J. Kleitman, and B. Berger. Unpublished manuscript. 1998.

[22] I. Rigoutsos and A. Califano. Searching in parallel for similar strings. *Computational Science and Engineering*, 1(2):60–75, 1994.

[23] V. V. Solovyev, A. A. Salamov, and C. B. Lawrence. Identfication of human gene structure using linear discriminant functions and dynamic programming. *Proceedings of ISMB*, pages 367–375, 1995.

[24] S. Sze and P. A. Pevzner. Las vegas algorithms for gene recognition: Suboptimal and error-tolerant spliced alignment. *Journal of Computational Biology*, 4(3):297–309, 1997.

[25] M. S. Waterman. *Introduction to Computational Biology: Maps, sequences and genomes.* Chapman and Hall, 1998.

[26] Y. Xu, R. J. Mural, M. Shah, and E. C. Uberbacher. Recognizing exons in genomic sequences using GRAIL II. In J. Setlow, editor, *Genetic Engineering: Principles and Methods*, volume 16. Penum Press, 1994.

[27] Z. Zhang, W. R. Pearson, and W. Miller. Aligning a DNA sequence with a protein sequence. *Journal of Computational Biology*, 4(3):339–349, 1997.

.

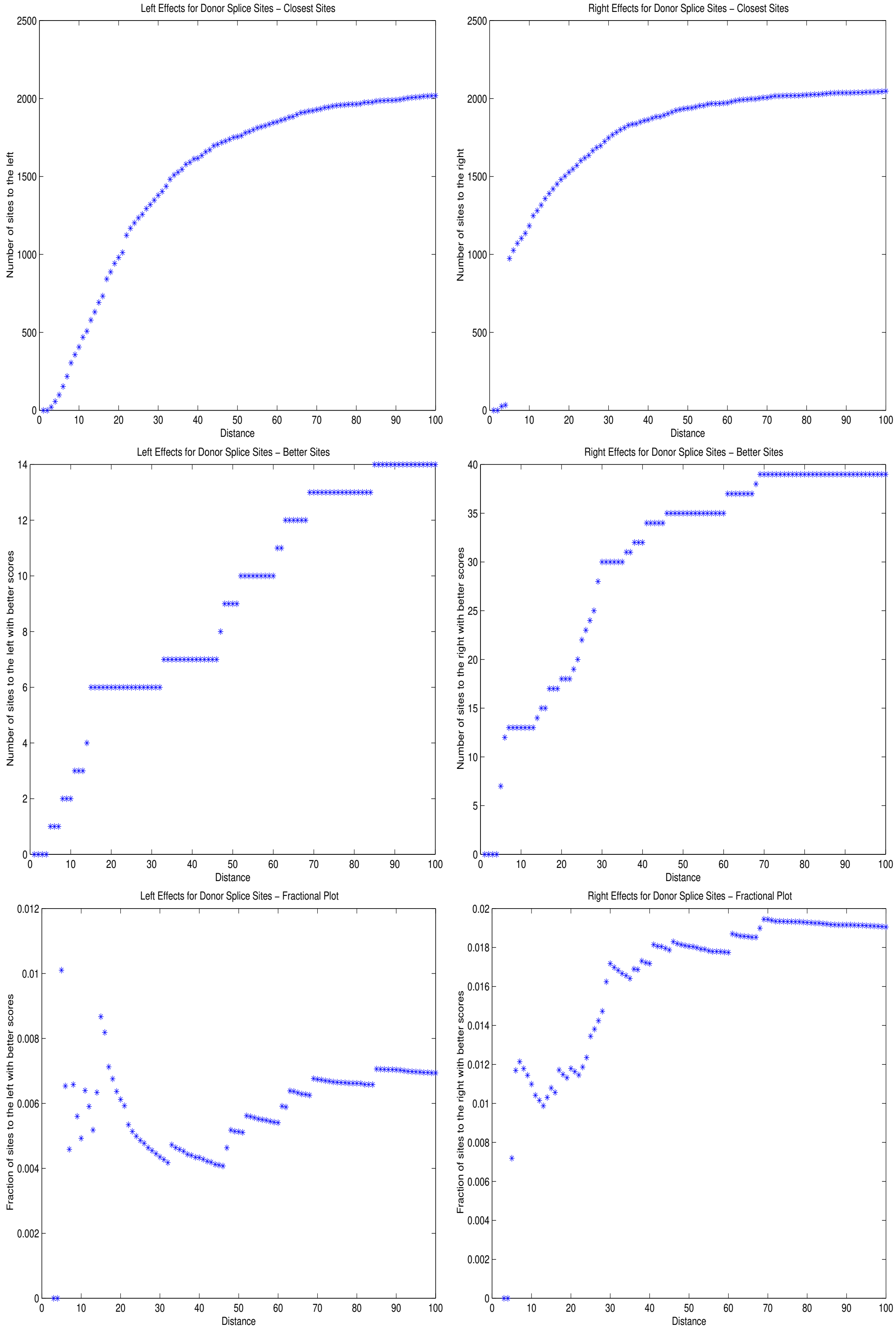

Figure 1: Left/Right effects for donor splice sites

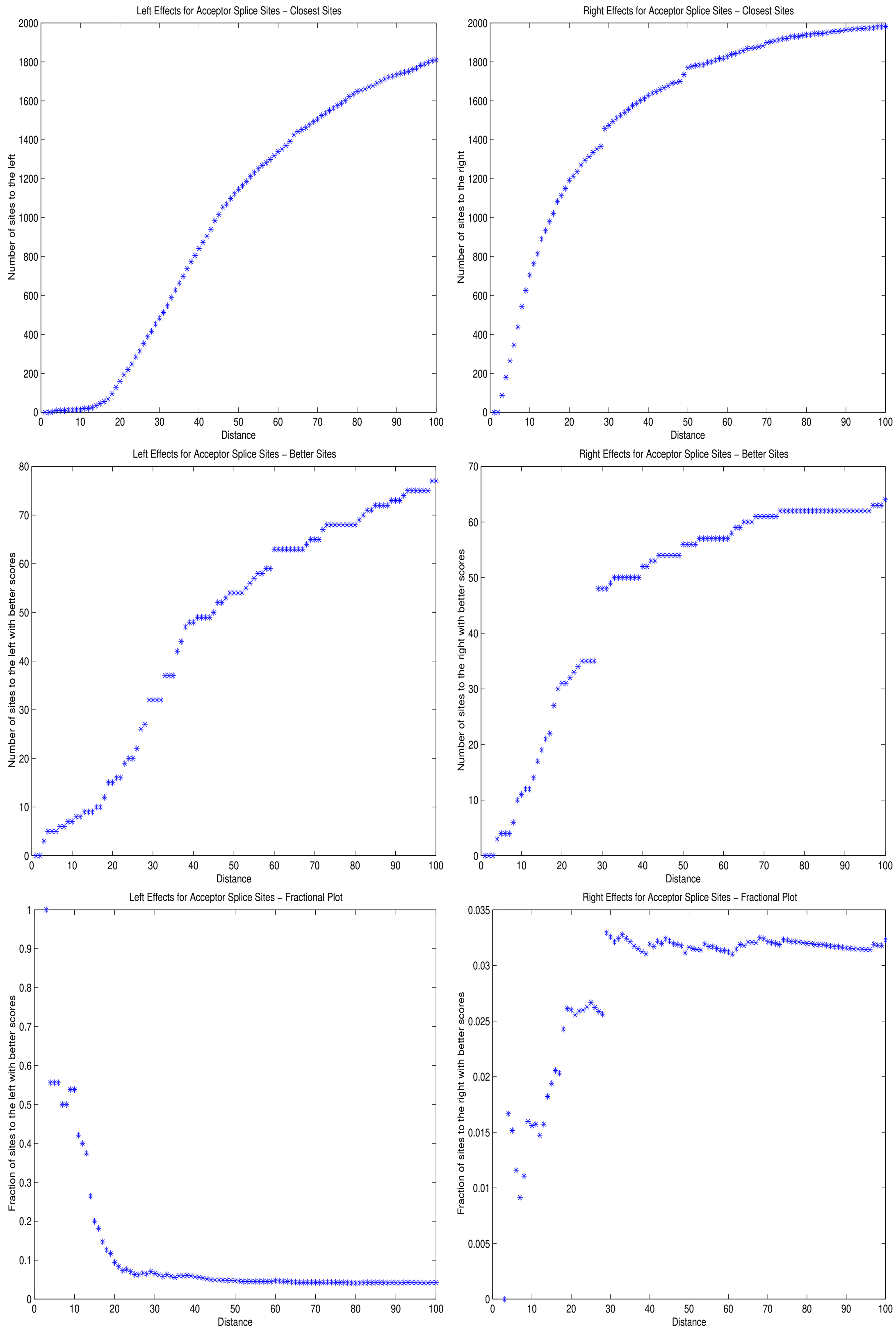

Figure 2: Left/Right effects for acceptor splice sites

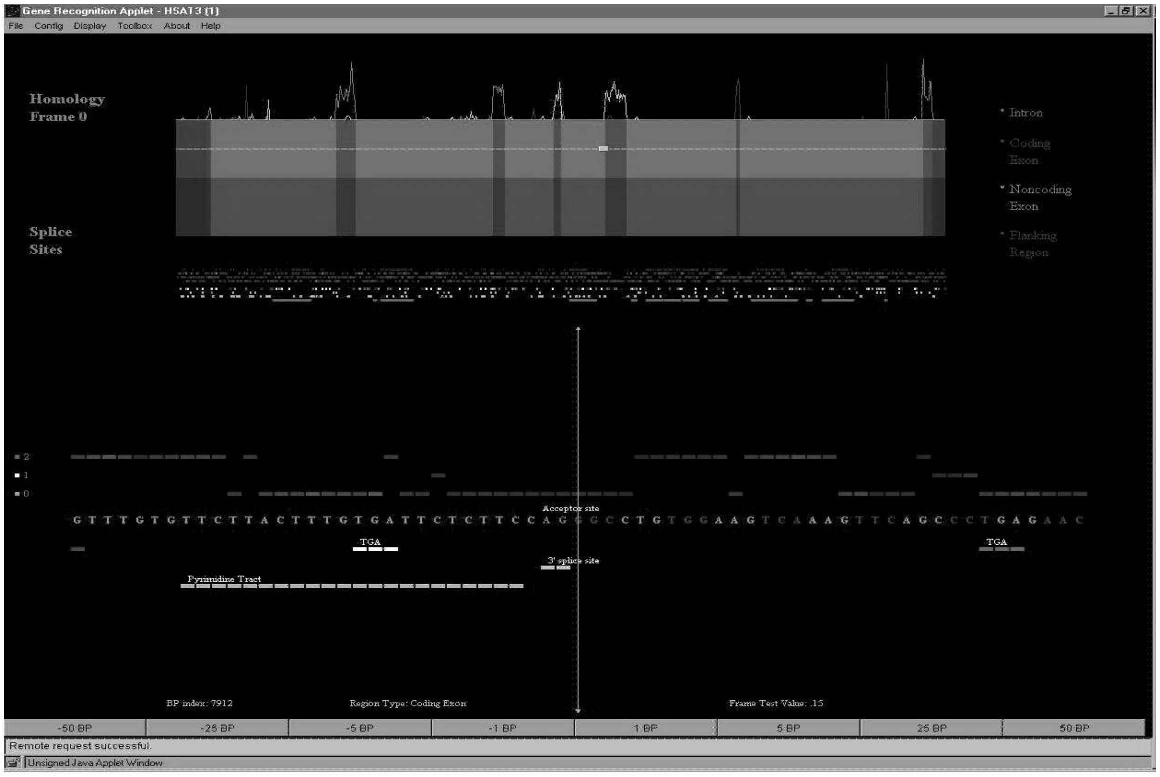

Figure 3: Java applet display: The sequence is displayed in a diagram on top. The darker interior regions correspond to coding exons. The number of hits to the OWL database at every position is displayed in the graph above the diagram. Subsequences of the actual sequence can be displayed below together with a variety of computed annotations. These include: potential splice sites, pyrimidine tracts, stop codons (with different colors corresponding to different frames), tuples much more common in introns than exons, repeats, as well as tuple frequencies in exons with respect to frame.

.

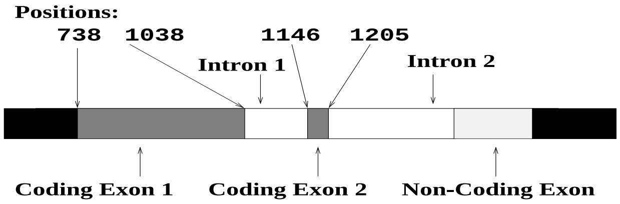

Figure 4: The Id3 gene.

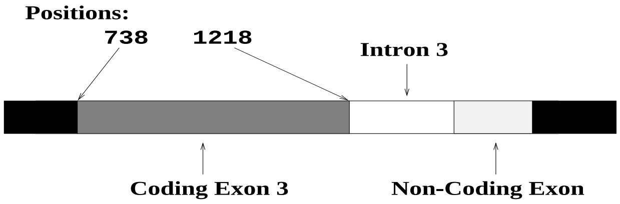

Figure 5: An alternative form of the Id3 gene.

.

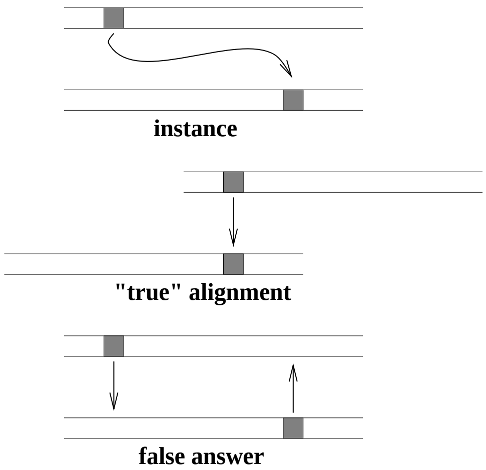

Figure 6: A difficult alignment problem.

Position in the given sequence	Length in nucleotides	Frame	Locus in the `OWL` database
171	27	0	Y338 MYCGE
687	24	0	S43230
738	90	0	ID3 MOUSE
738	90	0	ID3 RAT
927	24	0	I51316
927	24	0	I51278
954	27	0	ID2 HUMAN
954	27	0	ID2 MOUSE
954	27	0	ID2 RAT
954	27	0	ID4 HUMAN
954	27	0	ID4 MOUSE
954	27	0	JC2007
954	27	0	A41689
954	27	0	AF049135
954	27	0	OMID2PROT
954	27	0	HSU 16153
957	24	0	HUMID2X
990	24	0	A27280
738	300	0	ID3 HUMAN
837	201	0	ID3 RAT
840	198	0	ID3 MOUSE
1053	57	0	S71404
1152	24	0	CELF21H11
738	480	0	S71405
1128	90	0	S71404
1506	24	0	D89624
1671	24	0	PABL STRGR
1953	24	0	Y4EF RHISN
2310	24	0	AE0006769
438	24	1	MVIM SALTY
438	24	1	STYFLGA5
810	27	1	SCBLACABL
954	24	1	GUN1 TRILO
954	24	1	GUN1 TRIRE
1239	24	1	AF041044
1239	24	1	AFO41045
1548	24	1	CYAA SACKL
1596	24	1	DMU08282
1839	24	1	IMH1 YEAST
2283	24	1	S49247
2283	24	1	BTU04364
126	24	2	YC26 PORPU
210	24	2	CEC50F44
1146	60	2	ID3 HUMAN
1716	24	2	CEB03655
1785	24	2	AF005632

Table 1: `OWL` hits returned with a minimum length cutoff of $k = 8$ amino acids.

Locus	Description	Exon Position
HS1D3HLH	H. Sapiens Id3 gene	1146 – 1205
HSCYP216	H. Sapiens CYP21 gene	2036 –2214
HSDAO	H. Sapiens diamine oxidase gene	4856 – 6425
HSGROW2	Human germ line for growth hormone	1602 – 1799
HSMT1H	H. Sapiens MT1H gene	1409 – 1474
HSPRB3L	Human gene PRB3L	2087 – 2916
HSPRB4S	Human PRB4 gene	2213 – 2793
HSPSAG	Human DNA for prostate specific antigen	1688 – 1847
HSU12421	Human mitochondrial benzodiazepine receptor gene	3684 – 3872
HUMADAG	Human adenosine deaminase gene	35100 – 35202
HUMCP21OH	Human 21-hydroxylase B gene	2728 – 2906
HUMCP21OHC	Human mutant 21-hydroxylase B gene	2729 – 2907
HUMGHN	Human growth hormone gene	1827 – 2024
HUMGHV	Human growth hormone variant gene	1834 – 2031
HUMLYTOXBB	H. Sapiens lymphotoxin-beta gene	3559 – 3630
HUMMCHEMP	Human monocyte chemotactic protein gene	1472 – 1589
HUMMET2	Human metallothionein-II gene	1167 – 1232
HUMMET2	Human metallothionein-II gene	1436 – 1527
HUMNTRI	Human neutrophil peptide-1 gene	2627 – 2801
HUMNTRI	Human neutrophil peptide-1 gene	3382 – 3491
HUMNTRIII	Human neutrophil peptide-3 gene	2627 – 2801
HUMNTRIII	Human neutrophil peptide-3 gene	3382 – 3491
HUMPRCA	Human protein C gene	10516 – 11105
HUMTHROMA	Human thrombopoietin gene	5053 – 5718
LEBGLOB	Lepus europaeus adult beta-globin gene	2492 – 2620
MMGK5	Mouse glandular kallikrein gene	1645 – 1804
OAMTIB	Sheep metallothionein MT-Ib gene	1655 – 1720
OAMTIC	Sheep metallothionein MT-Ic gene	1229 – 1294

Table 2: Genes from the Burset–Guigó database with exons expressed in two frames.

Data Set	Nucleotide Sn.	Nucleotide Sp.	Exon Sn.	Exon Sp.
BG (exact matches removed)	86	95	78	73
BG	97	96	90	84
HKR (exact matches removed)	88	98	81	78
HKR	96	98	91	87

Table 3: Statistics for the `OWL` protein database. BG=Burset–Guigó, HKR=Haussler–Kulp–Reese.