A Dictionary-Based Approach for Gene Annotation

LIOR PACTHER,1,2 SERAFIM BATZOGLOU,2 VALENTIN I. SPITKOVSKY,1,2 ERIC BANKS,2
ERIC S. LANDER,3 DANIEL J. KLEITMAN,1 and BONNIE BERGER1,2

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.

Key words: gene recognition, exon prediction, splice site detection, alternative splicing

1. INTRODUCTION

AN IMPORTANT CHALLENGE facing biologists today is the accurate annotation of newly sequenced genomic data, in particular, determining the coding regions, exons, and introns of the genes. Very large databases of proteins, ESTs, and smaller databases of annotated genes are available for this task.

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 (1998), our dictionary consists of four tuples of amino acids, and for each, the protein sequences in OWL that contain it. We can construct such dictionaries very quickly from OWL and the dbEST databases (1998). 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. Given such information we use a dynamic programming algorithm to produce a parse of the gene into introns and exons.

To predict splice sites we use a splice site predictor that is based on the one used in GENSCAN (Burge and Karlin, 1997), 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.

Our method was codified in the ICE (identifier of coding exons) program (http://theory.lcs.mit.edu/ice). We have tested our method on two benchmark data sets. On a test set derived from the Haussler–Kulp–Reese (1996) benchmark data set (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

1Department of Mathematics and 2Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts.
3Whitehead Institute and Biology Department, Massachusetts Institute of Technology, Cambridge, Massachusetts.
exactly with the published annotation of their genes. When testing on the Burset–Guigó (1996) benchmark data set, 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 to 90% per coding nucleotide and from 30 to 70% per entire exon (Guigó, 1997).

1.1. Related work

Many computational methods have been developed for the purposes of gene annotation (Batzoglou et al., 1998). 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 (Waterman, 1998). The most widely used program developed for this purpose is BLAST (Altschul et al., 1990, 1997), 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 (Rigoutsos and Califano, 1994) 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 has 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 (Burge and Karlin, 1997), GENTIE (Kulp et al., 1996), GENEMARK (Lukashin and Borodovsky, 1998), VEIL (Henderson et al., 1997) (all based on hidden Markov models), FGENEH (Solovyev et al., 1995) (an integration of various statistical approaches for finding splice sites, exons, etc.), and GRAIL (Xu et al., 1994) (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, coding regions of a gene can be successfully identified. The alignment-based PROCRUSTES (Gelfand et al., 1996; Mironov et al., 1998; Sze and Pevzner, 1997) 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 (Huang et al., 1997). Specifically, the AAT tool addresses this by automatically using BLAST-like information from protein or EST databases for exon prediction. The INFO program (Laub and Smith, 1998) 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 increases.

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 being built), 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 that 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 (1998) and another from the dbEST database (1998). 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 (terabytes) 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 $s_i$ are integers ranging from 1 to the number of sequences, and the $t_i$ 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 target sequence that they matched. This was accomplished by loading the target sequence for each candidate segment, and then finding the longest subsegments of the candidate segment appearing in the loaded target sequence. This final procedure was divided into two substeps. The first consisted of building a minidictionary, 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 target 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

\[
\text{score(exon)} = \sum_p f(p)
\]

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 OVL 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. A potential splice site was defined as occurring of any GT or AG consensus. Potential splice sites were assigned a score using a modified version of the splice site detector described in Burge and Karlin (1997). 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 \( g_{s_2} \) of the downstream splice site (the other splice site having score \( g_{s_1} \), respectively) by setting

\[
g_{s_2}' = g_{s_2} + \min(0, g_{s_2} - g_{s_1}).
\]

Thus, the score of a potential splice site was penalized, if the next splice site upstream had a better score. 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.

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 \( b(d)/t(d) \). For example, Fig. 1 contains the results computed for donor splice sites. The plots are labeled left and right depending on the direction tested. Note that the left and right plots are not always on the same scale. 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 (Fig. 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.

Tests were conducted on a data set of 570 vertebrate genes, compiled by Burset and Guigo (1996), as well as on a data set of 130 genes taken from the Haussler–Kulp–Reese (1996) 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 base pairs” 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.
FIG. 1. Left/right effects for donor splice sites.
FIG. 2. Left/right effects for acceptor splice sites.
3. RESULTS AND DISCUSSION

3.1. Output of the program

The ICE 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 Fig. 4.
Figure 3 contains a screenshot of a Java applet used to visualize genes, as well as various types of information including 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 Fig. 5. In coding exon 3, the entire first intron is

<table>
<thead>
<tr>
<th>Position in the given sequence</th>
<th>Length in nucleotides</th>
<th>Frame</th>
<th>Locus in the OWL database</th>
</tr>
</thead>
<tbody>
<tr>
<td>171</td>
<td>27</td>
<td>0</td>
<td>Y338 MYCGE</td>
</tr>
<tr>
<td>687</td>
<td>24</td>
<td>0</td>
<td>S43230</td>
</tr>
<tr>
<td>738</td>
<td>90</td>
<td>0</td>
<td>ID3 MOUSE</td>
</tr>
<tr>
<td>738</td>
<td>90</td>
<td>0</td>
<td>ID3 RAT</td>
</tr>
<tr>
<td>927</td>
<td>24</td>
<td>0</td>
<td>IS1316</td>
</tr>
<tr>
<td>927</td>
<td>24</td>
<td>0</td>
<td>IS1278</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>ID2 HUMAN</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>ID2 MOUSE</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>ID2 RAT</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>ID4 HUMAN</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>ID4 MOUSE</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>JC2007</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>A41689</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>AF049135</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>OMD2PROT</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>HSU 16153</td>
</tr>
<tr>
<td>954</td>
<td>27</td>
<td>0</td>
<td>HUMID2X</td>
</tr>
<tr>
<td>990</td>
<td>24</td>
<td>0</td>
<td>A27280</td>
</tr>
<tr>
<td>738</td>
<td>300</td>
<td>0</td>
<td>ID3 HUMAN</td>
</tr>
<tr>
<td>837</td>
<td>201</td>
<td>0</td>
<td>ID3 RAT</td>
</tr>
<tr>
<td>840</td>
<td>198</td>
<td>0</td>
<td>ID3 MOUSE</td>
</tr>
<tr>
<td>1053</td>
<td>57</td>
<td>0</td>
<td>S71404</td>
</tr>
<tr>
<td>1152</td>
<td>24</td>
<td>0</td>
<td>CELF21HI11</td>
</tr>
<tr>
<td>738</td>
<td>480</td>
<td>0</td>
<td>S71405</td>
</tr>
<tr>
<td>1128</td>
<td>90</td>
<td>0</td>
<td>S71404</td>
</tr>
<tr>
<td>1506</td>
<td>24</td>
<td>0</td>
<td>D9624</td>
</tr>
<tr>
<td>1671</td>
<td>24</td>
<td>0</td>
<td>PABL STRGR</td>
</tr>
<tr>
<td>1953</td>
<td>24</td>
<td>0</td>
<td>Y4EF RHISN</td>
</tr>
<tr>
<td>2310</td>
<td>24</td>
<td>0</td>
<td>AE0006769</td>
</tr>
<tr>
<td>438</td>
<td>24</td>
<td>1</td>
<td>MVIM SALTY</td>
</tr>
<tr>
<td>438</td>
<td>24</td>
<td>1</td>
<td>STYFLGA5</td>
</tr>
<tr>
<td>810</td>
<td>27</td>
<td>1</td>
<td>SCBLACABL</td>
</tr>
<tr>
<td>954</td>
<td>24</td>
<td>1</td>
<td>GUN1 TRIMO</td>
</tr>
<tr>
<td>954</td>
<td>24</td>
<td>1</td>
<td>GUN1 TRIRE</td>
</tr>
<tr>
<td>1239</td>
<td>24</td>
<td>1</td>
<td>AF041044</td>
</tr>
<tr>
<td>1239</td>
<td>24</td>
<td>1</td>
<td>AF041045</td>
</tr>
<tr>
<td>1548</td>
<td>24</td>
<td>1</td>
<td>CYAA SACKL</td>
</tr>
<tr>
<td>1596</td>
<td>24</td>
<td>1</td>
<td>DMU08282</td>
</tr>
<tr>
<td>1839</td>
<td>24</td>
<td>1</td>
<td>IMH1 YEAST</td>
</tr>
<tr>
<td>2283</td>
<td>24</td>
<td>1</td>
<td>S49247</td>
</tr>
<tr>
<td>2283</td>
<td>24</td>
<td>1</td>
<td>BTU04364</td>
</tr>
<tr>
<td>126</td>
<td>24</td>
<td>2</td>
<td>YC26 PORPU</td>
</tr>
<tr>
<td>210</td>
<td>24</td>
<td>2</td>
<td>CEC50F44</td>
</tr>
<tr>
<td>1146</td>
<td>60</td>
<td>2</td>
<td>ID3 HUMAN</td>
</tr>
<tr>
<td>1716</td>
<td>24</td>
<td>2</td>
<td>CEB03655</td>
</tr>
<tr>
<td>1785</td>
<td>24</td>
<td>2</td>
<td>AF005632</td>
</tr>
</tbody>
</table>
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 (Deed et al., 1996). 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 (1996). 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 nonhuman genes. The parameters for the program were calibrated 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 k-8-tuples to the OWL database, only 0.5% were predicted to be in exons. Only 0.05% 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 to 90%. Predictions of exact exons also vary between the programs, with estimates between 30 and 70% specificity and sensitivity. The homology-based 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, the more the results become dependent on the general redundancy of the database. For genes with few matches, we 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, 1998). 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 (1981) consensus for translation starts, which involves positions -3 and +4 around the ATG (the consensus is AGXXATGG).

- **Pseudogenes:** Reverse transcribed genes that 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 unprecedented 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 (Benson, 1997; Zhang et al., 1997). For example, Fig. 6 illustrates a case in which 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 (Altschul et al., 1990)], 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 \textsc{Flash} program relies on storing the \textit{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 \texttt{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 \textsc{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. Also in contrast to \textsc{Info}, the method takes full advantage of the presence of long \textit{and} short hits. The \textsc{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 \textsc{Kat} (Huang \textit{et al.}, 1997) 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 in which we want 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 \textsc{Owl} dictionary was built on a Pentium II (400 MHz) in 1 min, and the \texttt{dbEST} dictionary, in less than 10 min. The total space occupied by the dictionaries was manageable on modern drives. The \texttt{dbEST} database occupied about 1.75 GB. 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.

### 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. L.P. has been supported in part by an NIH training grant and a Program in Mathematics and Molecular Biology graduate fellowship and S.B. by a Merck/MIT fellowship. L.P. and S.B. contributed equally to this work.

REFERENCES


Address reprint requests to:
Bonnie Berger
Department of Mathematics
Massachusetts Institute of Technology
Cambridge, MA 02139

E-mail: bab@mit.edu