Nucleic Acids Research, 1993, Vol. 21, No. 3 ***607–613***

Identification of coding regions in genomic DNA sequences: an application of dynamic programming and neural networks

Eric E.Snyder and Gary D.Stormo
Department of Molecular, Cellular and Developmental Biology, University of Colorado, Boulder, CO 80309-0347, USA

Received October 7, 1992; Revised and Accepted December 28, 1992

ABSTRACT

Dynamic programming (DP) is applied to the problem of precisely identifying internal exons and introns in genomic DNA sequences. The program GeneParser first scores the sequence of interest for splice sites and for these intron- and exon-specific content measures: codon usage, local compositional complexity, 6-tuple frequency, length distribution and periodic asymmetry. This information is then organized for interpretation by DP. GeneParser employs the DP algorithm to enforce the constraints that introns and exons must be adjacent and non-overlapping and finds the highest scoring combination of introns and exons subject to these constraints. Weights for the various classification procedures are determined by training a simple feed-forward neural network to maximize the number of correct predictions. In a pilot study, the system has been trained on a set of 56 human gene fragments containing 150 internal exons in a total of 158,691 bps of genomic sequence. When tested against the training data, GeneParser precisely identifies 75% of the exons and correctly predicts 86% of coding nucleotides as coding while only 13% of non-exon bps were predicted to be coding. This corresponds to a correlation coefficient for exon prediction of 0.85. Because of the simplicity of the network weighting scheme, generalization performance is nearly as good as with the training set.

INTRODUCTION

Until recently, the cloning and sequencing of a gene was one of the most difficult steps in the characterization of a protein. Often years of research into the physical and biochemical properties of a protein were amassed before determination of the genomic DNA sequence. Recent advances in DNA sequencing technology have changed this pattern. Current techniques in molecular biology allow the cloning of a gene with only the most basic knowledge of the protein in question. Often only an antibody, a few amino acids of protein sequence, or even a mutant phenotype in an experimental organism is sufficient to obtain a cDNA clone of a gene of interest. Obtaining the genomic sequence given a cDNA clone is straightforward. As a consequence,DNA sequence repositories such as GenBank have been growing at an astonishing rate.

Perhaps the most important advance in DNA sequencing technology is a conceptual one. With the initiation of projects to sequence the human genome and the genomes of the major experimental organisms, the determination of sequence data will no longer be dictated by the progress of biochemistry or genetics. Instead, the sequencing of genomic DNA will be done *en masse*, creating a huge database of unannotated sequence data for each organism. It is likely that by the turn of the century or shortly thereafter the sequence of the entire genomes of many organisms will be known. This will result in a fundamental change in the way we think of biology.

Current methods

Many useful sequence classification algorithms have been developed over the past ten years. Only recently have methods been developed which combine multiple pieces of evidence to predict coding regions in genomic DNA. To date, these approaches fall into two categories: methods using a rule-based approach for gene structure prediction and methods which use connectionist AI techniques to predict coding regions. In the former class, typified by the programs GeneID [1] and GeneModeler [2], sequence motifs such as start and stop codons and splice sites are evaluated and these features used to define candidate exons and introns. The sequences are then scored for properties such as codon usage and assembled by the application of rules to produce likely coding regions. The program GRAIL [3] is representative of the latter class. GRAIL identifies exons by combining information from numerous content statistics and weighting these scores using a neural network. GRAIL performs this task well, however its output shows only the positions of candidate exons on a linear sequence and does not attempt to produce assembled genes.

Advantages of current approach

To date, GeneModeler and GeneID are the only integrated packages that predict gene structure from genomic DNA sequence. While these programs are useful, they fall short of the accuracy required for interpretation of data from the genome sequencing projects. Furthermore, there may be fundamental limitations to the rule-based approach to gene identification. With this in mind, we propose a new approach to solving this problem.

Dynamic programming (DP) is a recursive optimization procedure first used in sequence analysis by Needleman and Wunsch in 1970 [4]. This method was originally applied to sequence alignment and later to RNA folding [5] and can be readily adapted to 'RNA splicing', the joining together of exons to form a complete coding sequence. In DNA sequence alignment, an alignment score is maximized by finding the path through an alignment matrix which maximizes the number of matches and minimizes the number of gaps and mismatches. In analogy, a sequence is divided into introns and exons by finding the best internally consistent set of high-scoring intron and exon subsequences. In contrast to the heuristic, rule-based approaches of GeneModeler and GeneID, DP accomplishes an exhaustive and mathematically rigorous search for the globally optimum solution. Furthermore, since DP deals with numerical scores rather than just lists of sequences which satisfy a given criterion, DP can produce ranked solutions with a more meaningful dependence on the scores of component introns and exons than GeneModeler or GeneID.

This ability to deal with numerical scores gives another critical edge to the current approach: machine learning algorithms can be used to fine-tune the sequence classification parameters. *A priori* there is no way to weight the individual tests used to classify sequences. For example, coding sequences are known to show codon preference and possess high local compositional complexity, but it is unclear how these parameters scale with respect to one another. If the contribution of each bit of evidence is additive and independent of the other parameters, it may be possible to use simple regression techniques to weight these parameters. In the more likely case that this is not true, connectionist learning algorithms such as the back-propagation neural network may be the best way to find a function that maps the various classification parameters to a score that DP can use. This process works by repeatedly presenting correct solutions to the neural network along with the raw input scores. The network calculates a solution based on the raw input scores and the current weight values. If the solution is incorrect, the network determines the error and makes a small change in the weights to decrease the error. This process is repeated until the weights accomplish the mapping to the desired degree of accuracy. In principle at least, it is possible to train a network to correctly find all known genes in GenBank. One then hopes the learned weights are sufficiently realistic to find and predict the structure of genes in unknown sequences.

METHODS

Dynamic programming is the pivotal element of this new approach to coding sequence identification. This application of this technique will be discussed first. Next, the intron and exon classification statistics will be introduced. Finally, the methods used to weight these statistics will be outlined.

Dynamic programming

Imagine that given an *N*-long sequence of genomic DNA, one could produce two half-matrices, $\mathbf{L_E}$ and $\mathbf{L_I}$, in which for every subsequence starting at position *i* and ending at position *j*, there were corresponding matrix elements, $L_E(i,j)$ and $L_I(i,j)$, which represent the log-likelihood that the subsequence is an exon or an intron, respectively. Given such a matrix, it is possible to apply dynamic programming [7] to find the optimum splicing pattern based on the evidence presented in the matrices. DP is used to enforce the constraints that introns and exons alternate in pre-mRNA and that these sequences are contiguous. DP is used to produce two vectors, $\mathbf{D_E}$ and $\mathbf{D_I}$, in which each element contains the score for the best combination of introns and exons ending at position *j*. Element $D_E(j)$ contains the score for a solution ending in an exon; $D_I(j)$ contains the score for a solution ending in an intron. The elements of can be calculated by the recursion:

$$D_E(j) = \max \begin{cases} L_E(1,j) \\ \max\limits_{k:2\rightarrow j-m} [L_E(k,j) + D_I(k-1)] \\ 0 \end{cases}$$

where *m* is the arbitrary minimum sequence length considered (usually 20 nucleotides). $\mathbf{D_I}$ is obtained in an analogous way. Taking the equation for $\mathbf{D_E}$ as an example, $D_E(j)$ is the maximum of three possibilities:

- $D_E(j) = L_E(1,j)$
  The segment from *1* to *j* exactly defines an exon.
- $D_E(j) = max_{k:2\rightarrow j-m}[L_E(k,j) + D_I(k-1)]$
  The segment from *1* to *j* ends in an exon. The 5′-end of the exon is at position *k*. $D_E(j)$ is then the score for the best combination of the exon $L_E(k,j)$ plus the best combination ending in an intron at position $k-1$. The score of this sequence is found in $D_I(k-1)$ (as determined earlier in the recursion). The value of $D_I(k-1)$ may be zero if there are no apparent introns preceding the exon at position *k*.
- $D_E(j) = 0$
  The scores for the two previous cases are both negative and are not considered further (*i.e.* there is no legitimate combination of sequences with an exon ending at position *j*).

Thus, $D_E(j)$ represents the score for the best internally consistent set of introns and exons with an exon ending at position *j*.

Sequence characterization statistics

DP is a rigorous method for interpreting the data presented in the **L** matrices. One must now devise a way to obtain such data. Unfortunately, there is no single statistic which exactly captures what it means to be an intron or an exon. However, there exist a multitude of tests which can quantify the various properties characteristic of these sequences. These tests will be discussed in this section. It is hoped that by weighting these parameters appropriately, these statistics can be combined into a single number which is a close approximation of the log likelihood information required for interpretation by DP.

Codon usage. The 64 triplets of the genetic code are used with unequal frequency in protein coding regions due to the amino acid composition of proteins and the unequal usage of synonymous codons [8]. In addition, there exists a strong correlation between adjacent codons in coding regions [9]. To capitalize on both of these effects, a table of in-frame 6-tuples was generated from the database of human genes used by Hutchinson and Hayden [10] to generate the codon usage table used in the program SORFIND. An unknown sequence can be compared to this standard and a likelihood estimate can be calculated [11]. This statistic allows one to determine whether a sample of in-frame 6-tuples is better modeled by random selection or one whose frequencies are that of the 6-tuples usage of the organism. In this statistic each reading frame is independent. Furthermore, the statistic is equally valid in coding and non-coding regions as well as regions of mixed coding and non-coding character.

Local compositional complexity. Eukaryotic genomes contain large amounts of repetitive DNA sequences referred to as simple sequence DNA. These sequences are typically found in noncoding regions of the genome. In contrast, coding regions tend to be informationally rich. This property is quantified by the Shannon information [12]. Konopka and Owens [13] have defined local compositional complexity as the Shannon information content of short subsequences (in our case, 8 nucleotides) averaged over the length of the sequence and have used this property to distinguish between coding and non-coding sequences.

Length distribution. Exons and introns have characteristic length distributions which can be used as evidence for their classification. Frequency histograms of exons and introns from the SORFIND database were prepared using bin sizes of 25 and 100 nucleotides, respectively. These values were normalized such that the most frequent sequence length of each type received a score of 1.0. In addition, putative intron sequences of length less than 70 nucleotides receive a score of zero to enforce the apparently hard constraint on the lower limit for intron length [14][15].

k-Tuple frequencies. A nucleotide sequence can be interpreted as a series of overlapping *k*-long words called *k*-tuples. The frequencies of these *k*-tuples can differ greatly between functionally different classes of sequences [16][17] [18] and can be used to discriminate between these classes. A table of 6-tuple frequencies for introns and exons was complied from the sequences in the SORFIND database. As in the codon usage statistic, the log-likelihood ratio for a given subsequence being either an exon or intron can be calculated from their respective 6-tuple frequency tables.

Splice signals. Using the collection of splice junction sequences of Senapathy and Shapiro [19], weight matrices were calculated for the donor and acceptor sequences [20]. An intron must be bounded at the 5′-end by a donor site and at the 3′-end by an acceptor site. Similarly, internal exons must be bounded by a 5′ acceptor site and a 3′ donor site. These constraints on the sequence surrounding intron-exon junctions are very useful in precisely defining the ends of these sequences when used in conjunction with search-by-content measures.

Integration of dynamic programming and machine learning

Using a set of arbitrary initial weights for the classification parameters, dynamic programming will fail to parse sequences correctly. The power of this method is that it is exactly these errors that are fed back to the learning algorithm to fine-tune the network weights to give better predictions. Let $T_{xi}(a,b)$ represent the raw score for statistic *i* (*e.g.* codon usage) of the exon starting at position *a* and ending at position *b*. This number is stored in layer *i* of the matrix $\mathbf{T_E}$ and need be calculated only once in the training session. Similarly, $T_{\mu i}(a,b)$ represents the score for statistic *i* given an intron and is stored in layer *i* of $\mathbf{T_I}$. We define the $\mathbf{L_E}$ matrix value for exon starting at position *a* and ending at position *b* as the weighted sum of all contributing statistics, calculated as follows:

$$L_E(a,b) = \frac{1}{1 + e^{-\sum_i^n w_{xi}(T_{xi}(a,b)+c_{xi})}}$$

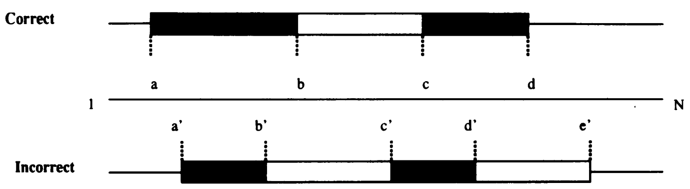

Figure 1. An example sequence. The figure above the line shows the correct partitioning of the sequence, the figure below the line an incorrect partitioning. The solid boxes correspond to exons, the open boxes to introns. Internal positions are labeled corresponding to the ends of the corresponding exon.

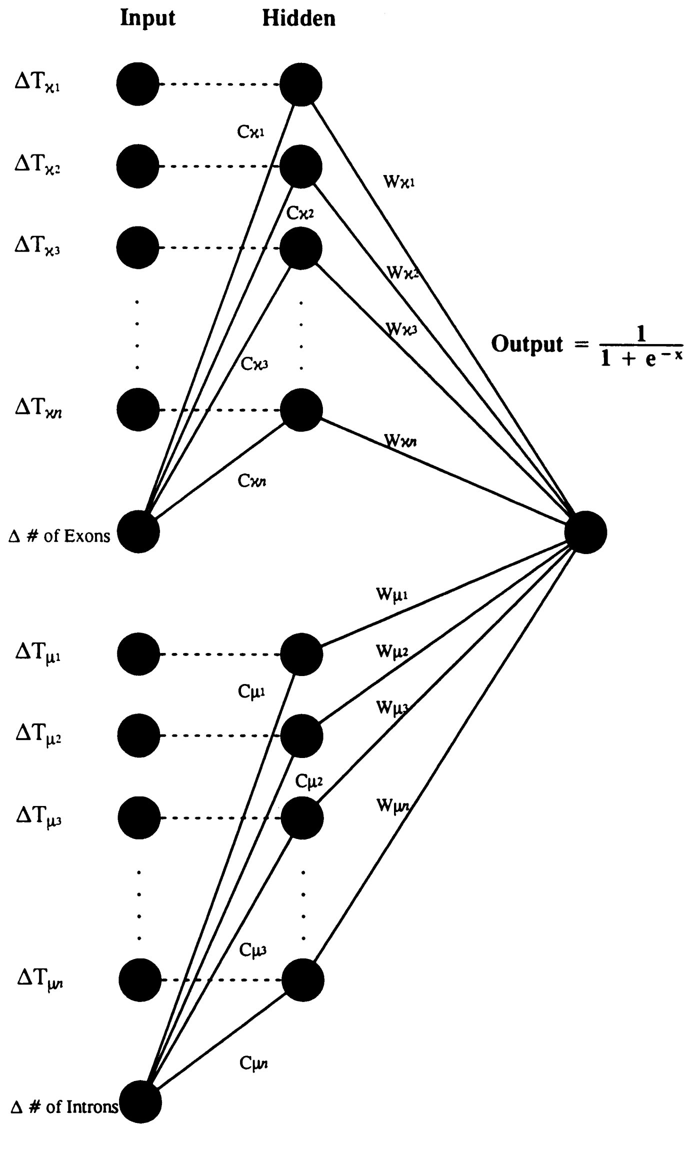

Figure 2. Training by inequality is accomplished with this network design. The input values are the differences between the correct and incorrect solutions for each statistical test (ΔT_{xi} for exons and $\Delta T_{\mu i}$ for introns) and the difference in the number of predicted exons and introns. The nodes in the hidden layer calculate the sum of the score-difference inputs and the number-difference inputs times the biases (the c_{xi} and $c_{\mu i}$ values). These sums are multiplied by the weights w_{xi} and $w_{\mu i}$ to give the input to the output node. The output node calculates the logistic function on its input. The inequality is satisfied when $0.5 < output \leq 1$, so we train to a target value of 1.0.

where w_{xi} is the weight and c_{xi} is the bias for statistic i. The elements of $\mathbf{L_I}$ are calculated in an analogous way.

Given the information in $\mathbf{L_E}$ and $\mathbf{L_I}$, DP finds the highest scoring combination of adjacent and non-overlapping introns and exons in the sequence. Take for example the correct and incorrect solutions shown in Figure 1. The DP score for each solution, X, is given by:

$$D(X{=}actual) = L_E(a,b)+L_I(b+1,c-1)+L_E(c,d)$$
$$D(X{=}incorrect) = L_E(a',b')+L_I(b'+1,c'-1)+L_E(c',d') + L_I(d'+1,e'-1)$$

One wishes to find a set of weights and biases that satisfy the following expression:

$$D(X{=}actual) > \mathrm{D}(X' \in \text{(incorrect)})$$

Thus, the actual structure of the gene must have a higher score than any possible incorrect solution. A neural network [21] has been designed to find weights that satisfy this inequality (see Figure 2). The input to this network consists of the difference between the sum of the scores of each statistic for the correct solution and a given incorrect solution. Using the example, the input value for statistic i for exons and introns would be calculated as follows:

$$\Delta T_{xi} = T_{xi}(a,b)+T_{xi}\mathrm{b,c})-T_{xi}(a',b')-T_{xi}(b',c')$$
$$\Delta \mathrm{T}_{\mu i} = T_{\mu i}(b+1,c-1)-T_{\mu i}(b'+1,c'-1)-T_{\mu i}(d'+1,e'-1)$$

The inputs from which the constant terms are derived are simply the difference between the number of actual exons or introns and the number in the incorrect solution. In the example, this input would be zero for exons and *-1* for introns.

Training is accomplished using the following procedure:

```
1. Initialize random weights
2. Construct T matrices for each sequence
   (contain raw sequence classification statistics for all
   subregions)
3. Construct L matrices for each sequence
   (contains weighted composite score from corresponding
   elements in T)
4. Run dynamic programming on each set of L matrices
5. Test: Is desired accuracy obtained?
     Yes: STOP
     No: CONTINUE
6. Train neural network: Iterate training procedure to
   convergence
7. Update weights to reflect training
8. GOTO 3.
```

Training and test data

A training set was assembled from the human genes used in the training of the program GeneID. Three loci (HUMP45C17, HUMPAIA and HUMPRPH1) were omitted from the training set to avoid duplication in the independent test set used to evaluate GRAIL. These sequences were cropped for the purposes of training to encompass the first and last introns plus approximately 50 nucleotides 5′ and 3′, respectively. This was done to avoid down-weighting the splice site parameters if the program were forced to find splice sites for the distal boundaries of the terminal exons when in fact these boundaries are defined by start and stop codons.

Three test sets were used to evaluate the performance of GeneParser. The first set was based on the genes used to evaluate the program SORFIND. All genes except HUMFCREB (CDS not indicated) and HUMFIBRA (CDS listed as 'putative') that were not found in the training set or in the GeneID and GRAIL test sets (see below) were used. The sequences were cropped to embrace all but the terminal exons, as in the training set. In order to facilitate the comparison of this program with previous methods, the test sets of both GeneID and GRAIL were used. Locus HAMRPS14A was omitted from the GeneID set because it was not present in GenBank release 71; HUMTPA was omitted from the GRAIL set because it exceeded the upper limit for the length of sequences which GeneID could analyze. When evaluating the performance on these sequences, the entire locus was used, including terminal exons and intergenic DNA. The data on the performance of the current email server versions of GRAIL and GeneID were generously provided by Steen Knudsen and Roderic Guigo (personal communication).

RESULTS AND DISCUSSION

Comparison with GeneID and GRAIL

The results on the training and test sets at the levels of complete intron and exon sequences and nucleotide prediction are summarized in Tables 1 and 2. Two scoring schemes were used to evaluate the performance of GeneParser. In scheme 1, only sequences between the first and last exons were considered;exons and introns outside this region were not included in the totals. This scheme was chosen because it mirrors the type of data that was used for training the neural network. Scheme 2 analyzes the entire GenBank locus, including terminal exons and intergenic DNA. This scheme was chosen to allow direct comparison with the performance figures for GRAIL and GeneID.

Of the 56 training sequences, GeneParser learned to exactly predict the intron-exon structure of 19 of these examples (34%). At the level of complete exon sequences, 113 of 150 exons were precisely identified (75%) and an additional 11 exons were predicted by overlap, bringing the total number of exons identified to 124 (85%). The remaining 36 predictions did not overlap actual exons. When the performance on the training set is analyzed at the nucleotide level, 86% of the coding bases are predicted to be in exons and 86% of the predicted coding bases are actually coding. Over all, the correlation coefficient [22] for exon prediction (C_E) is 0.85.

The generalization performance of GeneParser was tested under scoring scheme 1 using the SORFIND test set. Of the 58 sequences, the structure of only 10 were exactly predicted (17%). However, the overall performance on this data was actually slightly better than on the training set. 211 of 279 exons were precisely identified (76%), with another 42 predicted by overlap. In all, 253 exons (91%) were at least partially predicted. This increased performance probably reflects the use of this large pool of sequences to compile the 6-tuple and in-frame 6-tuple tables.

GeneParser was tested on the GeneID and GRAIL test sets using scoring scheme 1. Considering the combined results at the level of complete sequences, the performance was not significantly reduced relative to the training set. Small reductions in sensitivity for both exon and intron prediction were offset by small increases in specificity. At the level of nucleotides, C_E went from 0.85 in the training set to 0.82 in the combined test set, C_I showing a larger decrease from 0.80 to 0.71.

Using scoring scheme 2, a direct comparison between our program and GeneID and GRAIL can be made. One should keep in mind the following when comparing these results: GRAIL and GeneParser were specifically trained on human genes. GeneID was developed for use on vertebrate genes in general. The GeneID training set also contains sequences from non-human vertebrates. Furthermore, GeneID and GeneParser were developed to analyze more restricted classes of DNA sequences. GeneID was intended for the analysis of pre-mRNA. GeneParser was developed to parse internal exons and introns.

With this in mind, the performance of GeneParser under these conditions was not surprisingly reduced. Considering the GRAIL and GeneID test sets together, the ability to precisely predict exons fell to 46%. However, 73% of these exons were still predicted by overlap. At the nucleotide level, 72% of the coding region was predicted to be coding while 73% of the predicted coding region was in fact coding, corresponding to a correlation coefficient of 0.68. It should be noted that the sensitivity of GeneParser exceeds that of GeneID and GRAIL on both data sets. This number is partially offset by reduced specificity, however C_E for GeneParser on this data exceeds that of the other methods by a small margin.

Table 1. Results by sequence.

Data Set †	Seq Type	Total	True Pos	False Pos	False Neg	GeneParser Sn	GeneParser Sp
Training	Exon	150	111	67	39	74%	62%
	Intron	202	111	114	91	55%	49%
Test SORFIND	Exon	279	211	102	68	76%	67%
	Intron	335	213	151	122	64%	59%
Test GRAIL	Exon	61	41	36	20	67%	53%
	Intron	78	41	53	37	53%	44%
Test GENEID	Exon	113	79	33	34	70%	71%
	Intron	140	74	62	66	53%	54%
Test COMBINED	Exon	174	120	69	54	69%	64%
	Intron	218	115	115	103	53%	50%
Data Set ‡	**Seq Type**	**Total**	**True Pos**	**False Pos**	**False Neg**	**GeneParser Sn**	**GeneParser Sp**
Test GRAIL	Exon	98	41	77	57	42%	35%
Test GENEID	Exon	167	82	85	85	49%	49%
Test COMBINED	Exon	265	123	162	142	46%	43%

Results in terms of complete exons and introns precisely predicted. Sn, Sensitivity = *tp/(tp+fn)*, Sp, Specificity = *tp/(tp+fp)*. † Performance is scored over internal exons and introns. ‡ Performance is score over the entire GenBank entry, including terminal exons and intergenic sequences.

Table 2. Results by nucleotide.

Data Set †	Seq Type	Total	True Pos	False Pos	False Neg	GeneParser Sn	GeneParser Sp	GeneParser CC	GeneID Sn	GeneID Sp	GeneID CC	GRAIL Sn	GRAIL Sp	GRAIL CC
Training	Exon	22302	19317	2950	2985	86%	87%	0.85						
	Intron	136389	130414	2317	5975	95%	98%	0.80						
Test SORFIND	Exon	43124	40445	5141	2679	93%	89%	0.89						
	Intron	209490	198509	2538	10981	94%	99%	0.83						
Test GRAIL	Exon	8024	7380	2473	644	91%	75%	0.81						
	Intron	77263	73380	451	3883	94%	99%	0.77						
Test GENEID	Exon	16720	13535	1536	3185	80%	90%	0.83						
	Intron	82988	75828	3116	7160	91%	96%	0.67						
Test COMBINED	Exon	24744	20915	4009	3829	84%	84%	0.82						
	Intron	160251	149208	3567	11043	93%	98%	0.71						
Data Set ‡	**Seq Type**	**Total**	**True Pos**	**False Pos**	**False Neg**	**GeneParser Sn**	**GeneParser Sp**	**GeneParser CC**	**GeneID Sn**	**GeneID Sp**	**GeneID CC**	**GRAIL Sn**	**GRAIL Sp**	**GRAIL CC**
Test GRAIL	Exon	17239	12944	6617	4295	75%	66%	0.66	67%	74%	0.66	65%	86%	0.72
Test GENEID	Exon	25358	17299	4795	8059	68%	78%	0.69	65%	78%	0.67	48%	87%	0.61
Test COMBINED	Exon	42597	30444	11211	12153	72%	73%	0.68	66%	77%	0.67	55%	87%	0.66

Results analyzed at the level of individual nucleotides. Sn, Sensitivity = *tp/(tp+fn)*, Sp, Specificity = *tp/(tp+fp)*. Correlation Coefficients (CC) were calculated as in [22]. † Performance is scored over internal exons and introns. ‡ Performance is score over the entire GenBank entry, including terminal exons and intergenic sequences. The data for GeneID and GRAIL was generously provided by and Roderic Guigo and Steen Knudsen.

Strengths and weakness of GeneParser

The performance of the current implementation of GeneParser is quite promising. Given a region known to contain internal exons, the program does an excellent job at predicting the intron-exon structure of the gene. GeneParser does particularly well on short exons, a problem that most programs find especially difficult. Of the 18 exons in the training set of 60 nucleotides or less, 12 (67%) were correctly predicted. Similarly, 6 of 10 of these short exons were predicted in the SORFIND test set, indicating that this property is capable of generalizing to novel data. On the other hand, GeneParser has difficulty accurately predicting long introns. Of the 13 introns in the training set greater than 2000 nucleotides in length, only 5 (38%) were accurately predicted. When these introns are missed, a short exon is often included within the intron. This is a property of the DP model--given a choice of a single high-scoring long intron and two shorter introns and an exon with intermediate scores covering the same interval, DP will choose the later if the sum of their scores exceeds the former. However, these long introns comprise only 6% of the introns in the training set. This clearly represents a tendency of the network to learn the weights which optimize performance on the majority of the data at the expense of performance on outliers in the population.

Computational aspects

There is no fundamental limit to the length of sequences which GeneParser can analyze. The current version has been tested on sequences as long as 73,000 nucleotides. The memory requirement of the program grows linearly with sequence length and the run time increases with the square of the sequence length. However, if the user wishes to limit the maximum intron or exon length considered (thereby limiting the number of matrix elements calculated and searched), the program will run in linear time once the length of the input sequence exceeds this limit. Run time is not dependent on the number of predicted introns or exons in the solution.

Conclusions

We do not claim that the training or test sets are free from homologous sequences nor do we claim that these sequences used in this study are necessarily representative of GenBank. However, the ability of GeneParser to perform as well on novel data indicates that generalizability of the method or network weights learned in training is not a major problem. This criticism is often leveled at neural network approaches to problem solving. In our case, the reason that this is not a problem is undoubtedly due to the small number (24) of adjustable parameters in the network. Since thousands of examples are accumulated in the training stage, it is not possible to over fit or 'memorize' the training set. This ensures the good generalization properties we see with GeneParser.

In its current implementation, GeneParser is a two-state classifier: internal exon and intron. For both classes, boundaries are determined by splice site weight matrices. The program knows nothing about the signals that determine the ends of coding sequences. Not surprisingly, GeneParser performed less well on the GRAIL and GeneID test sets which contain not only terminal exons but intergenic sequences as well. However, it is remarkable that GeneParser still does slightly better than GeneID, which uses specific weight matrices to find the boundaries of terminal exons. We are confident that when the DP algorithm is expanded to a four-state classifier, the performance can be significantly improved.

FUTURE WORK

The development of GeneParser is still underway. We plan a number of additions which should increase performance on the biologically important task of precisely identifying coding in uncharacterized genomic DNA sequences. First, procedures can be added to allow a more precise determination of the ends of terminal exons. This also requires a modification of the DP algorithm to search two additional **L** matrices, one for 5′ terminal exons and one for 3′ exons. However, since these exons may well differ in the character of their content statistics, the improved performance will certainly be worth the additional complexity.

In its current form, GeneParser does not use reading frame compatibility as a constraint when linking predicted exons. However, this is an important constraint which can be incorporated into the DP algorithm with a modest computation expense. This will help the program avoid placing small, low scoring exons within long introns which can sometimes cause frame shifts when translating the predicted gene product.

Finally, there are additional statistics which can be used to better classify sequences. The Fickett TESTCODE statistic [23] and Konopka's distance analysis method [24] measure characteristic periodicities in intron and exon sequences. GeneID makes use of the first derivative of the codon usage function to help reduce the number of false-positive splice sites.In addition, the use of a branchpoint weight matrix could also help identify genuine acceptor sites. Finally, searching protein databases for similarity with short translated segments from putative exons could also be used as evidence that a segment is actually coding.

In addition, we are not limited to the simple linear weighting model used in the current work. It is possible to add additional hidden layers to the neural network. This can allow us to model non-additive interactions between the various classification statistics. For example, there is evidence in at least some species that the 5′ splice sites of longer introns more closely resemble the consensus than those of shorter introns [25]. Thus, it might be desirable to give the $\mathbf{L_I}$ value for a short intron with a poorer acceptor site a slightly higher score than either of the two statistics alone might indicate. If such non-linear relationships exist between scoring parameters, a network with hidden layers should be able to exploit them to model the data more closely.

Sub-optimal solutions

Extending this method to give suboptimal solutions is straightforward. Unlike sequence alignment procedures in which great pains are taken to avoid 'trivial' variations on the optimum alignment [26], this is often what is desired in gene structure determination. In many examples of alternative splicing, there may be small changes in the use of particular splice sites or choice of possible exons. These suboptimal solutions can be found by systematically eliminating specific introns and exons from consideration by setting their corresponding **L** matrix score to $-\infty$ and re-running the DP procedure. Doing this for each intron and exon in the optimal solution would generate a set of suboptimal solutions, each with a DP-derived score which can be used to rank these solutions.

Application to non-human organisms

While GeneParser was developed and tested for use on human DNA sequences, the program is easily adapted for use on sequences from other organisms. Having prepared splice site weight matrices, *k*-tuple and length distribution tables appropriate for that species, the network weights can be learned as before. In future versions of GeneParser, we plan to include tables and network weights optimized for performance on a variety of experimental organisms including *Drosophila* and *C.elegans*.

Error tolerance

In the development of GeneParser, we have consciously chosen statistics which use predominantly local information. With the exception of the codon usage statistic, all sequence classification parameters are quite tolerant of frameshift errors in the input sequence. Since uncharacterized genomic sequences are very likely to contain such errors, this should greatly enhance the utility of this program. Furthermore, the ability to train GeneParser on sequences with different levels of simulated sequencing errors allows the development of network weight profiles specifically adapted to the anticipated error rate in the sequences one plans to analyze. For example, if frameshift errors were known to be common, this might be reflected in a down-weighted codon usage statistic and an increased weight for the 6-tuple frequency statistic.

AVAILABILITY

GeneParser was developed on a Silicon Graphics Indigo workstation and provides graphic plots of **T**- and **L**-matrix data to aid in analysis and interpretation. Versions without graphics output can be ported to a variety of Unix platforms. Contact the authors at the above address or by electronic mail to eesnyder@boulder.colorado.edu.

ACKNOWLEDGEMENTS

We would like to thank Roderic Guigo and Steen Knudsen for helpful discussion regarding performance analysis and test data selection as well as generously providing their data on the performance of GeneID and GRAIL on the test data. This work benefitted from discussions at the 'Recognizing Genes' workshop at the Aspen Center for Physics. This work was supported by NIH grant HG00249.

REFERENCES

1. Guigo, R., Knudsen,S., Drake, N., and Smith, (1992) J. Mol. Biol. 141–157.
2. Fields, C. A., Soderlund, C. A. (1990) Computer Appl. Biol. Sci. 36: 263–270.
3. Uberbacher, E. C., Mural, R. J. (1991) Proc. Natl. Acad. Sci. USA 388: 11261–11265.
4. Needleman, S. B., Wunsch, C. D. (1970) J. Mol. Biol. 348: 443–453.
5. Nussinov, R., Jascobson, A. B. (1980) Proc. Natl. Acad. Sci. USA 377: 6309–6313.
6. Zucker, M., Stiegler, P. (1981) J Nucl. Acids. Res. 39: 133–148.
7. Nemhauser, G. L. (1966) Introduction to Dynamic Programming. John Wiley and Sons, Inc., New York.
8. Staden, R., McLachlan, A. D. (1982) Nucl. Acids. Res. 310: 141–156.
9. Farber, R., Lapedes, A., Sirotkin, K. (1992) J. Mol. Biol. 3226: 471–479.
10. Hutchinson, G. B., Hayden, M. R. (1992) Nucl. Acids.
11. Gribskov, M. Devereux, J., Burgess, R. R. (1984) Nucl. Acids. Res. 312: 529–549.
12. Shannon, C. E., Weaver, W. (1964) The Mathematical Theory of Communication. The University of Illinois Press, Urbana, Illinois.
13. Konopka, A. K., Owens, J. (1990) Gene Anal. Techn. Appl. 37: 35–38.
14. Wieringa, B., Hofer, E., Weissmann, C. (1984) Cell 337: 915–925.
15. Ulfendahl, P. J., Pettersson, U., Akusjarvi, G. (1985) Nucl. Acids. Res. 313: 6299–6315.
16. Bougueleret, L., Tekaia, F., Sauvaget, I., Claverie,J.-M. (1988) Nucl. Acids. Res. 316: 1729–1738.
17. Claverie, J.-M., Bougueleret, L. (1986) Nucl. Acids. Res. 314: 179–196.
18. Claverie, J.-M. Sauvaget, I., Bougueleret, L. (1990) Meth. Enzymol. 3183: 237–252.
19. Shapiro, M. B., Senapathy, P. (1987) Nucl. Acids. Res. 315: 7155–7174.
20. Stormo, G. D. (1987) Identifying Coding Sequences. In: Nucleic Acid and Protein Sequence Analysis: A Practical Approach. Eds. M. J. Bishop, C. J. Rawlings, IRL Press.
21. Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1988) Learning internal representations by error propagation, In: Parallel Distributed Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations, pp. 318–362, eds.: Rumelhart, D. E., McClelland, J. L. and the PDP Research Group, Cambridge, MA, MIT Press.
22. Brunak, S., Engelbrecht, J., Knudsen, S. (1991) J. Mol. Biol. 3220: 49–65.
23. Fickett, J. W. (1982) Nucl. Acids. Res. 310: 5303–5318.
24. Konopka, A. K., Smythers, G. W., Owens, J., Maizel, J. V. (1987) Gene Anal. Techn. Appl. 34: 63–74.
25. Fields, C. (1990) Nucl. Acids. Res. 318: 1509–1512.
26. Waterman, M. S., Eggert, M. (1987) J. Mol. Biol. 3197: 723–728.