J. Mol. Biol. (1995) **248**, 1–18

JMB

Identification of Protein Coding Regions In Genomic DNA

Eric E. Snyder and Gary D. Stormo

Department of Molecular Cellular and Developmental Biology, University of Colorado, Boulder CO 80309-0347 U.S.A.

We have developed a computer program, GeneParser, which identifies and determines the fine structure of protein genes in genomic DNA sequences. The program scores all subintervals in a sequence for content statistics indicative of introns and exons, and for sites that identify their boundaries. This information is weighted by a neural network to approximate the log-likelihood that each subinterval exactly represents an intron or exon (first, internal or last). A dynamic programming algorithm is then applied to this data to find the combination of introns and exons that maximizes the likelihood function. Using this method, we can rapidly generate ranked suboptimal solutions, each of which is the optimum solution containing a given intron-exon junction. We have tested the system on a large collection of human genes. On sequences not used in training, we achieved a correlation coefficient for exon nucleotide prediction of 0.89. For a subset of G + C-rich genes, a correlation coefficient of 0.94 was achieved. We have also quantified the robustness of the method to substitution and frame-shift errors and show how the system can be optimized for performance on sequences with known levels of sequencing errors.

Keywords: gene identification; exon structure; coding sequence; artificial intelligence; dynamic programming

Introduction

The initiation of the large-scale genome sequencing projects such as the Human Genome Project has provided a strong impetus for the development of computer programs that deduce gene structure from primary sequence alone. One approach is to search for homologous sequences in databases such as GenBank. However, recent experience with large-scale sequencing in yeast (Koonin *et al*., 1994) and *Caenorhabditis elegans* (Wilson *et al*., 1994) suggests that only a fraction of genes have identifiable homologs in the current literature. Furthermore, there is evidence from the study of ancient conserved regions (Green *et al*., 1993) that the fraction of sequences with inter-phylum homology is unlikely to increase beyond about 40 to 50% as more sequences are accumulated because most of these motifs have already been identified.

In addition to sequence similarity, there are numerous types of evidence that can be applied to this problem (for a review, see Stormo, 1987). Content statistics that measure properties of long stretches of sequence help to distinguish exons from other sequence types. Looking for open reading frames and codon usage bias are classic methods but many others have been developed in recent years (for a review, see Fickett & Tung, 1992). While these methods give a rough indication of the location of exons in a sequence, the boundaries are not well defined. In contrast, the ends of introns (donor and acceptor splice sites) are characterized by specific sequence motifs that can be searched for directly by consensus sequence or weight matrix (Stormo, 1988, 1990) or neural network (Brunak *et al*., 1991). However, even the best of these methods still predict many more false sites than true sites (Brunak *et al*., 1991).

Because of the limitations of the individual statistics, several groups have developed gene identification programs or algorithms that combine multiple pieces of evidence into a single framework. GeneID (Guigó *et al*., 1992) and GeneModeler (Fields & Soderlund, 1990; Soderlund *et al*., 1992) identify candidate exons that satisfy certain criteria and then use heuristic, rule-based approaches to assembling these exons to make predictions of gene structure. GenLang uses formal language theory and generative grammars to model the structure of tRNAs

Present address: E. E. Snyder, Sequana Therapeutics, Inc. 11099, North Torrey Pines Road, Suite 160, La Jolla, CA 92037, U.S.A.

Abbreviations used: DP, dynamic programming; MSP, maximal-segment pair; CDS, protein coding sequence.

0022–2836/95/160001–18 $08.00/0

(Searls & Dong, 1993) as well as protein-coding genes (Searls, 1993; Dong & Searls, 1994). Probably the best known is GRAIL (Uberbacher & Mural, 1991), which uses a neural network to combine content statistics and make predictions on whether a window of sequence is coding or non-coding. Other methods such as cluster analysis are used to define the boundaries of the exons and assemble them into complete genes.

We have developed a program, GeneParser (Snyder & Stormo, 1993), which combines the connectionist approach of GRAIL with the power of the recursive optimization procedure dynamic programming (DP), to make predictions of the intron–exon structure of genes. The first version of GeneParser attempted to identify only internal exons and introns. In a side-by-side comparison of this version of GeneParser with GRAIL and GeneID, our program out-performed the others by a small margin when tested on a novel data set (Snyder & Stormo, 1993).

We present here a dynamic programming algorithm that can parse a sequence into an arbitrary number of classes subject to a number of ''grammatical'' constraints. This method is similar in spirit to DP methods for speech segmentation developed by Cohen (1981) and Bridle & Sedgwick (1977). However, since biological sequences consist of intrinsically discrete elements, the problem of parsing can be mapped more naturally into a DP algorithm. Given some measure of likelihood that a given sequence interval belongs to one of several classes, DP can be used to find the maximum likelihood solution while enforcing the syntactical constraints of gene structure, namely that introns and exons alternate in the genomic sequence and that they are adjacent but do not overlap.

Recently, others have applied DP to gene parsing, including Gelfand & Roytberg (1993) and Xu *et al*. (1994*b*). Knight & Myers (1995) have developed a ''super pattern matching'' algorithm that uses DP to parse biological sequences into domains of different classes, even allowing those domains to overlap.

As our first result, we show how to apply this method to the problem of identifying four classes of sequence in eukaryotic coding regions: introns and first, internal and last exons†. This permits the identification and assembly of all protein coding regions in eukaryotic genes. In our second result, we show how to train a neural network to weight the several content and site statistics used to score intervals in the sequence of interest. In particular, we show a gradient descent method that will update weights in the context of the DP algorithm. In our third result, we show how to use DP to find suboptimal solutions to the parsing problem and illustrate the use of a graphical tool to display these solutions. This method, reminiscent of Zucker's suboptimal RNA folding method (Zucker, 1989), is efficient and can be used to identify the optimum solution containing an intron–exon transition at any arbitrary position. Coupled with the graphical user interface developed for this purpose, the method gives the user a clear indication of which predicted splice junctions are most likely to be correct and which have nearby suboptimal solutions of similar score. In our final result, we demonstrate the flexibility of the neural network system by optimizing the performance of GeneParser on sequences of atypical base composition and on sequences with simulated sequencing errors.

There is a growing body of evidence that the statistical properties of genes vary in non-trivial ways dependent on the local base composition of the region in which the gene resides (Mouchiroud *et al*., 1991; Bernardi, 1989). Furthermore, the performance of current gene identification programs vary significantly depending on the G + C content of the locus under examination (Snyder & Stormo, 1994). It is also important to consider the effects of sequencing errors, which typically occur in experimentally generated data. All methods are assumed to degrade in performance. However, only recently have current methods been tested to quantify this effect (Snyder & Stormo, 1994). Until now, no author has attempted to optimize the performance of their programs on ''error prone'' sequence data.

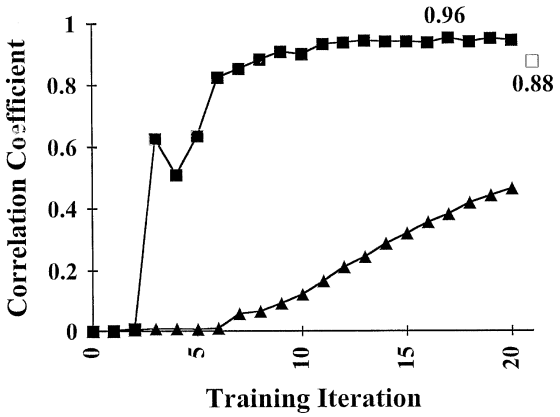

Figure 1. Learning curve for Training Set: By Nucleotide. A simple linear network (see Figure 8A) was trained on medium G + C content genes from training set I using all statistics including BLAST search. The correlation coefficient for nucleotide prediction on the Training Set (filled squares) is plotted as a function of the number of training iterations. Each training iteration consists of 1 pass of GeneParser through the training sequences followed by 1 training session of the neural network. The new weights are then passed to GeneParser for use in the next iteration. The generalization performance (open square) is the performance on the Test Set I using the best weights obtained in training. The training error (filled triangles) refers to the fraction of training vectors that the neural network fails to learn to classify by the end of the training session.

† For the purposes of this paper, we will consider an exon to be the coding portion of coding exons. This differs from the conventional usage of an exon representing the portion of the transcript retained in the final processed mRNA. Thus, a first exon will begin with a start codon and end with a donor site; a last exon will start with an acceptor site and end just before a stop codon.

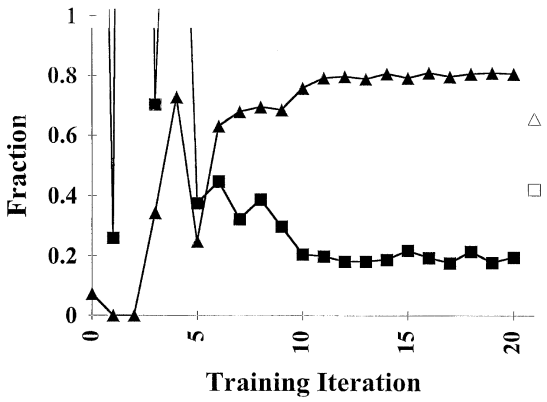

Figure 2. Learning curve for the Training Set: By Interval. For the same training session displayed in Figure 1, the performance is displayed in terms of the number of intervals correctly predicted (filled triangles) and incorrectly predicted (filled squares). To be considered correct, the exon must be predicted exactly. The number of intervals incorrectly predicted is plotted as the ratio of the number of incorrectly predicted intervals to the number of actual intervals in the training set. The number starts out very large because with random weights, many intervals look ''good'' and the DP score can be maximized by finding a solution that contains as many as possible. The generalization performance is shown (open symbols).

Results

Training: general features

GeneParser weights for a network of type A shown in Figure 8 were optimized as described in Methods. Figures 1 and 2 plot performance as a function of training iteration. Starting from random weights, the performance is initially poor. The correlation coefficient is near zero because the system is making essentially random predictions based on the randomly chosen initial weights. The number of incorrect exons and introns predicted is very large because DP can maximize the score of a parse by simply including many small intervals. This number is much higher than the actual number of introns or exons in the gene. As more and more training examples are accumulated, performance gradually improves, reaching a plateau after 10 to 15 passes through the data. From this plateau, the set of weights that score highest on the training data is used to evaluate generalization performance on the test data. Interestingly, the set of weights that gives the best performance on the training set does not necessarily give the best performance on the test set. However, these weights are used to evaluate generalization performance since they can be chosen most objectively and without reference to the test data.

The training error (expressed as the fraction of training vectors not correctly classified by the neural network) is instructive because it gives an indication of when maximal performance has been achieved. While the performance of GenParser is increasing, the training error remains small and relatively constant. When performance begins to level off, the training error begins to increase as GeneParser begins to pick more and more good but incorrect solutions that the network cannot correctly classify.

Generalization performance and comparison with other methods

Table 1 shows the performance of GeneParser on Test Sets I and II. Three versions of GeneParser were tested: GeneParser1, trained on the entire Training Set; GeneParser2, trained with separate networks for differing G + C contents; GeneParser3, like GeneParser2 but also including the BLAST statistic. For comparison purposes, the email server versions of GeneID (Guigó *et al*., 1992) and GRAIL2 (Uberbacher & Mural, 1991) were used to evaluate the performance of these programs on the Test Sets. GRAIL3 (GRAIL2 + exon assembly routines) was tested manually using the interactive client server XGRAIL (Xu *et al*., 1994*a,b*).

Optimization for G + C content

We also sought to optimize the performance of GeneParser by training on sequences grouped by G + C content. Three separate networks were trained using the Training Set divided as discussed in Methods. The results of generalization performance are shown in Table 1. On Test Set I, the data show an insignificant change in performance overall relative to the program trained on the Training Set as a whole. This, however, is due to a small decrease in performance in the large medium-G + C content training set that obscures an important increase in performance on G + C poor sequences. (CC_E = 0.59 for normal training, CC_E = 0.72 for G + C optimized training).

G + C-poor sequences have traditionally been the most difficult sequences to predict, in part because genes tend to be less frequent in G + C poor regions. It has been suggested that regions of differing G + C content may represent functionally distinct parts of the genome. There is some evidence that A + T-rich isochores favor genes that are more tissue-specific and less highly expressed. In contrast, G + C-rich isochores tend to contain constitutively expressed, housekeeping genes (Mouchiroud *et al*., 1991; Bernardi, 1989). Bernardi (1989) adds that these genes tend to be the most biased in codon usage, an observation that would facilitate their identification. These differences, if true, would have important implications for programs that attempt to identify genes based on ''universal'' statistical properties such as hexamer usage.

Performance with BLAST statistic

The results discussed thus far have not included the BLAST statistic. Since new genes are being added

Table 1

Performance of the programs of test sets I and II

Program	GeneID	GRAIL2	GRAIL3	GeneParser1	GeneParser2	GeneParser3
Set I: total						
CC	0.69	0.80	0.83	0.78	0.78	0.89
Sn	0.69	0.86	0.83	0.78	0.87	0.96
Sp	0.77	0.80	0.87	0.84	0.76	0.86
Exons correct	0.42	0.34	0.52	0.47	0.47	0.69
Exons overlapped	0.73	0.88	0.81	0.81	0.87	0.92
Set I: high G + C						
CC	0.65	0.81	0.88	0.87	0.89	0.94
Sn	0.72	0.86	0.87	0.90	0.90	0.98
Sp	0.73	0.84	0.95	0.90	0.93	0.93
Exons correct	0.38	0.27	0.67	0.58	0.64	0.80
Exons overlapped	0.80	0.96	0.89	0.96	0.96	1.00
Set I: medium G + C						
CC	0.67	0.80	0.83	0.77	0.75	0.88
Sn	0.65	0.88	0.86	0.77	0.86	0.95
Sp	0.77	0.77	0.84	0.82	0.70	0.83
Exons correct	0.37	0.37	0.51	0.37	0.41	0.66
Exons overlapped	0.67	0.85	0.83	0.79	0.84	0.89
Set I: low G + C						
CC	0.81	0.75	0.62	0.59	0.72	0.86
Sn	0.82	0.74	0.51	0.54	0.79	0.93
Sp	0.85	0.83	0.87	0.76	0.75	0.84
Exons correct	0.80	0.35	0.25	0.35	0.40	0.60
Exons overlapped	0.85	0.80	0.55	0.55	0.85	0.90
Set II: total						
CC	0.55	0.79	0.75	0.72	0.80	0.88
Sn	0.50	0.74	0.68	0.68	0.82	0.91
Sp	0.75	0.91	0.91	0.81	0.86	0.89
Exons correct	0.33	0.33	0.31	0.36	0.46	0.61
Exons overlapped	0.64	0.66	0.58	0.66	0.76	0.87
Set II: high G + C						
CC	0.73	0.79	0.80	0.61	0.71	0.88
Sn	0.85	0.83	0.80	0.86	0.65	0.98
Sp	0.73	0.82	0.88	0.56	0.87	0.83
Exons correct	0.43	0.43	0.50	0.57	0.57	0.71
Exons overlapped	0.86	0.86	0.79	0.86	0.79	1.00
Set II: medium G + C						
CC	0.52	0.79	0.75	0.74	0.82	0.88
Sn	0.47	0.75	0.68	0.67	0.84	0.91
Sp	0.76	0.91	0.91	0.86	0.87	0.91
Exons correct	0.29	0.32	0.32	0.34	0.46	0.63
Exons overlapped	0.62	0.68	0.58	0.65	0.79	0.88
Set II: low G + C						
CC	0.62	0.64	0.62	0.59	0.67	0.78
Sn	0.56	0.49	0.45	0.59	0.71	0.84
Sp	0.71	0.88	0.89	0.64	0.67	0.76
Exons correct	0.47	0.37	0.16	0.32	0.37	0.37
Exons overlapped	0.63	0.42	0.42	0.58	0.58	0.74

GeneID was run using the ''–no_blast'' option and GRAIL using the ''–2'' option to invoke exon termini identification. GRAIL3 indicates results obtained running GRAIL2 and ''assembly'' using the interactive XGRAIL package (exons scored as ''marginal'', ''good'' or ''excellent'' were used). Three versions of GeneParser were run. GeneParser1 corresponds to the simple 4-state program. GeneParser2 was run using hexamer tables and network weights optimized for G + C content. GeneParser3 was run as GeneParser2 but includes the BLAST statistic.

to the sequence database daily, the BLAST score provides a ''moving target'' that hampers comparison with other programs. Furthermore, not all authors filter exact matches using the same rules, making side-by-side comparison more difficult.

A network was trained using the standard procedure but now with the inclusion of the BLAST statistic. The generalization results are shown in Table 1. Over all, an 11% increase is realized on Test Set I and an 8% increase on Test Set II, attributable to increases in performance on all sequence types. The increases are not obviously correlated with G + C content since large changes in one test set are often offset by smaller changes in the other. However,

Table 2

Performance on error-prone data

Substitution rate (%)	Frame-shift rate (%)	Exon *CC*	Intron *CC*	Exon *CC*	Exon *CC*
		Training		Test set I	Test set I*
0.0	0.0	0.85	0.88	0.90	—
0.5	0.0	0.81	0.87	0.90	0.89
1.0	0.0	0.83	0.85	0.90	0.87
2.5	0.0	0.76	0.79	0.85	0.78
0.5	0.5	0.72	0.71	0.68	0.75
0.0	0.5	0.82	0.83	0.73	0.80
0.0	1.0	0.57	0.62	0.57	0.62
0.0	2.5	0.42	0.45	0.52	0.46

Networks were trained with differing rates and types of errors. The networks were then tested in the high G + C content genes in Test set I using these weights. For comparison, the same sequences (with errors) were run using weights obtained on sequences without errors (Test set I*).

low G + C content genes in Test Sets I and II both improved by greater than 10%. In most cases, and by most criteria, the GeneParser3 performance was better than any of the others tested.

Performance on data with simulated errors

The trade-off between quality *versus* quantity is a matter of considerable debate in the human genome project (Roberts, 1991). Given the greatly increased expense required to get a sequence absolutely correct (if this is indeed possible), it is often expedient to be satisfied with a sequence with some low frequency of errors. With this is mind, we have tested the performance of GeneParser on sequences with errors introduced at known frequencies.

GeneParser has been developed with error tolerance in mind. Because frame-shift errors are so devastating to frame-dependent statistics such as in-frame hexamer statistics or codon usage, we have emphasized the use of non-frame-dependent statistics. Since GeneParser is extremely flexible by virtue of its use of neural networks to weight the statistics, we have also retrained GeneParser to see if it is possible to teach the system to perform better, given a known error rate. The high G + C sequences from the Training Set had errors introduced randomly at specified rates for both substitution and frameshift errors. After training on those datasets the performance was tested on the high G + C sequences from Test Set I, which had the same types of errors introduced at the same rates. The results of these studies are shown in Table 2. This training was done without the use of the BLAST statistic so the results should be compared with the GeneParser2 column on the high G + C data of Test Set I of Table 1.

It is also important to note that GeneParser, trained on data without errors, is relatively robust. The correlation coefficient for exon nucleotides is 0.78 for substition rates as high as 2.5%, and is 0.80 for frameshift rates as high as 0.5%. However, training on data containing substitution errors can significantly increase performance on test data with similar errors. This difference increases as the error rate increased over the range of values tested. At a substitution frequency of 2.5%, performance was above 10% higher using the error-optimized weights. The results of training on data containing frame-shift errors is more difficult to interpret. At low error frequency, normal weights seem to work best. However, when the error frequency reaches 2.5%, the error-optimized weights become preferable. At this level, error-optimized weights allow a 13% increase in performance. This behavior is probably due to the stochastic nature of the simulated frame-shift errors. At 0.5% to 1% frame–shift errors, a given exon may or may not have its reading frame interrupted; the network thus received conflicting training data concerning the importance of frame-dependent statistics. As errors become more frequent, there is a very good chance that a given exon will contain a frame-shift, giving the network a clear signal to down-weight the importance of

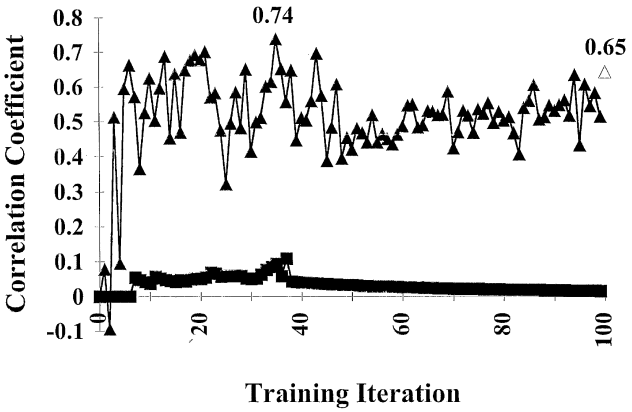

Figure 3. Learning curve for Multi-Layered Network. A network of type C shown in Figure 8 was trained on high G + C content genes from Training Set I without the BLAST search. The network contained a total of 24 hidden units (6 for each sequence type) and was fully connected to the input units for each type. The network proved difficult to train to the same level of performance as linear networks. Furthermore, generalization performance was poor. As discussed in the text, this is probably the result of the inability of the training scheme to generate sufficient input to saturate the more complex network.

frame-dependent statistics and up-weight those less sensitive to frame–shift errors. While this was only a small test of the ability of GeneParser to be trained for error tolerance, these results suggest that training with a large set of genes that have errors introduced at some frequency might do much better than other approaches at correctly predicting the genes in real genomic sequencing data with comparable expected error rates.

Multilayered networks

An example training run using a multilayered neural network containing 24 hidden units (six for each sequence type) is shown in Figure 3. The performance on high G + C content genes from Training Set I was somewhat worse with the multilayered network (CC_E = 0.74 compared with 0.87 for a linear network trained under identical conditions). Furthermore, generalization performance was considerably worse (CC_E = 0.65 compared with 0.79). While the computational power of multilayered networks is well documented, no gain compared with the single-layered neural network was realized in our system.

This failure is almost certainly related to the generalization properties of complex networks. As the system is trained, GeneParser selects more and more examples from the space of possible training patterns. In effect, we are measuring generalization performance at each training iteration. In training the simplest network (with 40 weights and biases), the number of patterns exceeds the number of free parameters after a few iterations. Introducing hidden layers into the network increases the number of free parameters (to 276 for the network used in Figure 3). Widrow's heuristic (Widrow & Walach, 1984), which relates network complexity to training set size, suggests that nearly 10^4 training examples would be required before good generalization could be expected from this network. In fact, the 78 training iterations shown in Figure 3 is equivalent to less than ten training iterations for the single-layered network. Interestingly, while the learning curve has apparently leveled off after about 50 iterations, the corresponding increase in training error is not seen. If this is taken to be indicative of the point at which maximal training has been achieved, it may be possible to obtain increased performance with multilayered networks. However, the computational requirements are considerably higher than for the single-layer networks.

Suboptimal solutions

An important feature of GeneParser is the ability to predict suboptimal solutions. Figure 4 shows the graphical output of GeneParser when using the bidirectional DP suboptimal parsing algorithm. In the display window, eight bar graphs are plotted, one for each interval type in each DP direction. In the ''forward'' direction, the height of the bar corresponds to the *S*-vector score for the best solution containing the 3'-end of the indicated sequence type at that position. The ''reverse'' plot contains the same information for the 5'-junction. The height is displayed as $e^{s(\phi) - s(\phi^*)}$, which approximately represents the ratio of the probability of the parse in question ($s(\phi)$) to the optimum parse ($s(\phi^*)$). The height can also be displayed as the ratio of the DP score for the parse in question to the score for the optimum parse.

The plot in Figure 4 displays the *S*-vector scores for the human growth hormone gene. Some regions, such as the middle of introns two and four, have many high-scoring potential junctions. This reflects a relative uncertainty or degeneracy in the position of the junctions in the optimum solution (although the optimum solution does correspond to the actual gene structure). For example, in the last exon, there are two peaks in the ''FORWARD-LE'' graph at the 3'-end of the gene. The highest peak corresponds to the actual structure, the smaller peak, to the third-ranked solution, shown in Figure 5. In contrast, other junctions in the optimum solution have few or no nearby junctions that score near the optimum, for example in exons three and four. This is indicative of a greater certainty of the parse in this region.

The graphical user interface illustrated in Figure 4 allows the user to display solutions interactively. By using the appropriate mouse button, users can (1) display the optimum solution containing a particular junction by clicking on that junction; (2) descend through the suboptimal solutions one at a time showing a strip plot of each; (3) descend through junction scores by lowering the cutoff by a fixed interval. This permits the user to explore gene parsings that may be remote from the global optimum but are suspected on the basis of experimental data. One of the higher scoring suboptimal junctions, indicated by the arrows, corresponds to an alternative splicing product for this gene. The optimal structure containing that junction is shown in the bars labeled ''Pred'' at the bottom of the figure.

Discussion

Unlike other gene assembly programs, GeneParser does not suffer from a combinatorial explosion in complexity when faced with sequences that are long or that have a large number of candidate exons. The complexity of GeneParser's dynamic programming algorithm is $O(N^2)$, where N is the sequence length. The complexity is reduced to $O(N)$ when an upper limit is put on the length of the longest subinterval (potential intron or exon) in the sequence. While the algorithm has a favorable upper time bound, the execution of the program is not fast. For example, a 5 kb sequence takes ten minutes to run on an 50 MHz R4000 SGI Indigo workstation. This has limited our ability to train complex multilayered neural networks that could, in principle, significantly improve performance. Faster training and prediction could be accomplished by limiting the number of intervals considered as possible junctions. Currently,

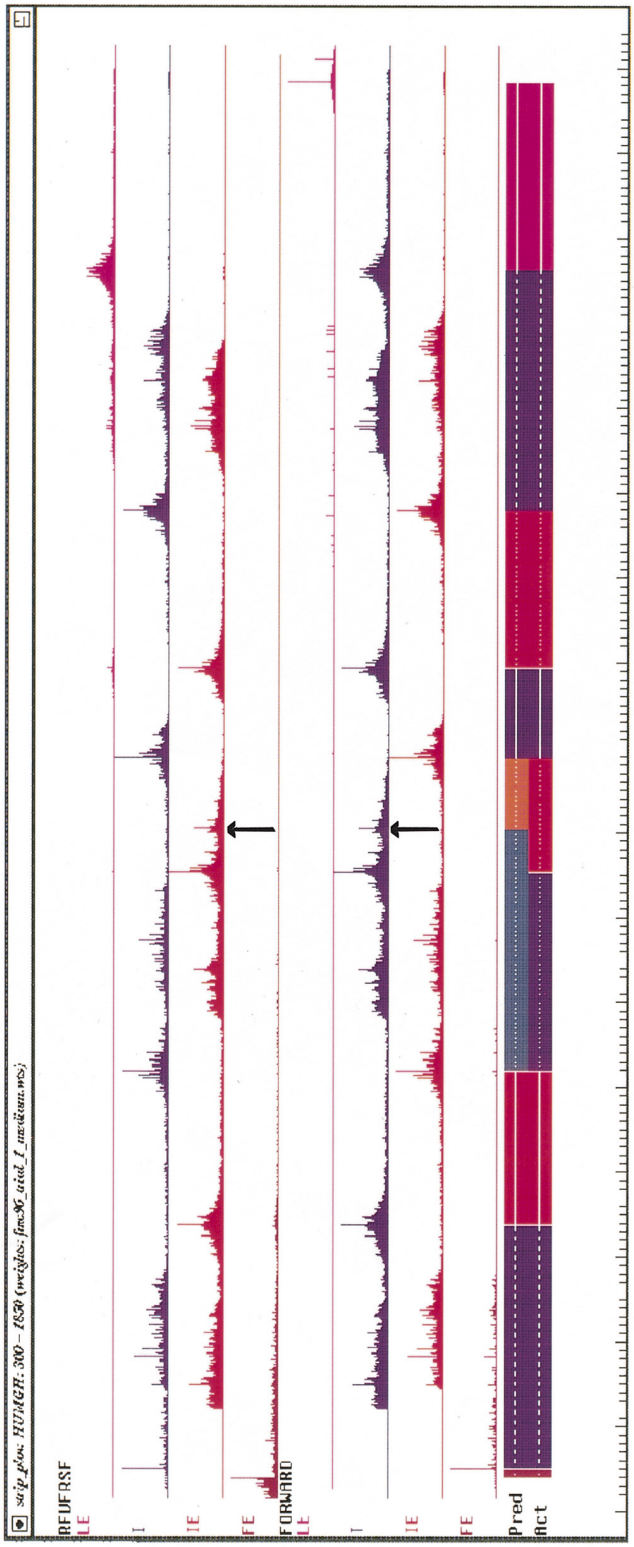

Figure 4. Suboptimal solutions displayed for human growth hormone gene using strip plot. The vertical bars represent *S*-vector values corresponding to the best solution containing the specified junction. The scores of all possible junctions are displayed. In the "forward" direction, the junction is the 3'-end of the sequence type shown on the left (e.g. FE = "first exon", I = "intron", etc.); in the "reverse" direction, the junction corresponds to the 5'-end. The bottom bars labeled Act show the actual solution (the predominant isoform as annotated in GenBank). Blue bars indicate introns, red bars exons. The bars labeled Pred show the optimum solution containing the junction indicated by the arrows. This solution represents the second major species of growth hormone found in the pituitary.

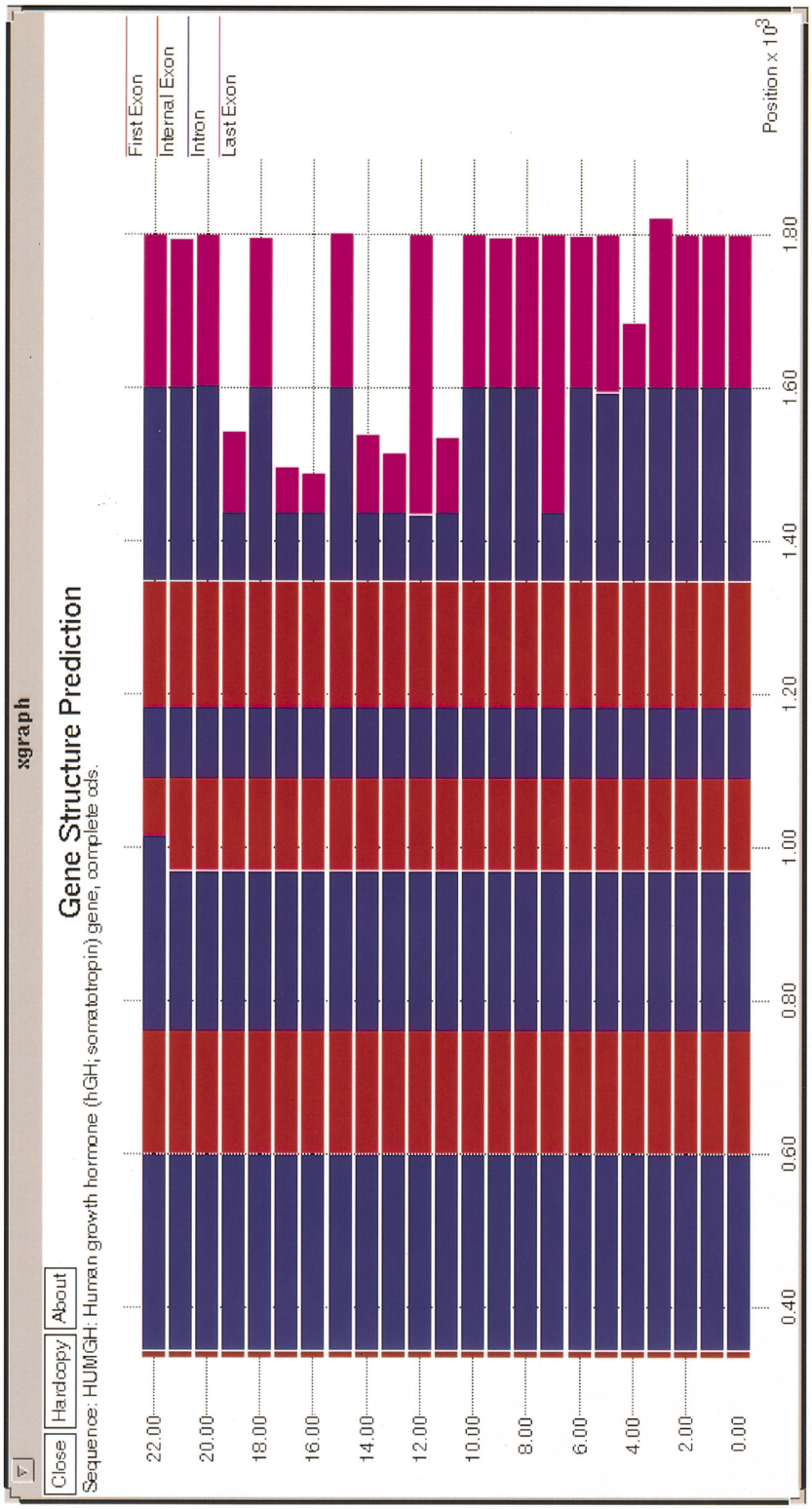

Figure 5. A list of suboptimal solutions for human growth hormone displayed using the tool Xgraph (©1989, David Harrison). The solution at 0.00 corresponds to the predominant spliced form *in vivo*. Suboptimal solutions 1 through 21 are displayed according to their rank. Note that exons 1 through 4 are the same in all solutions. Most of the variability is confined to the 3'-end of the gene, indicating that predictions in this reason are less reliable. The solution labeled 22 corresponds to the 592nd solution on the list and is the same as the alternatively spliced product shown in Figure 4.

every subinterval of the sequence is evaluated by the statistics and the DP finds the optimum from all of those possibilities. One could enforce the ''GT and AG'' rule, whereby only donor sites beginning with GT and acceptor sites beginning with AG would be considered and the DP would find the optimal solution, and suboptimals, from that more limited set of possibilities. This would increase the efficiency of the program considerably, but at the cost of possibly missing the correct solution, especially with error-containing sequences. In this work we have chosen the slower but more thorough approach, but in practical applications the efficiency gained by limiting the number of possible solutions may be more important. For example, a new GRAIL implementation utilizes dynamic programming to find optimal solutions from a limited number of candidate exons (Xu *et al*., 1994*a*).

The implementation of GenParser is flexible to allow additional statistics to be easily added to the system. Potential additional statistics include branch-point recognition and recognizers of intron-specific sequences such as *Alu* repetitive elements. The approach can be generalized easily to more sequence types. While the current program parses a genomic sequence into at most four sequence types (first exon, internal exon, intron and last exon), it is possible to consider untranslated exons as well as intergenic regions. By adding the methods for searching for additional signals, such as transcriptional start sites, polyadenylation signals and even promoters, the additional contextual constraints may lead to improved performance. Such extensions would also allow the program to better predict the gene structure of long genomic sequences, even those containing multiple genes.

Methods

Statistical tests to identify coding regions

All gene assembly programs available today make use of multiple lines of evidence from which to draw conclusions about the positions of coding exons in a genomic sequence. This information can take two forms: search by content and search by signal (Stormo, 1987). Content statistics measure bulk properties of a length of sequence while signal-based methods look for short, conserved sequence elements. Some of these statistics require reference to information derived from a sample of known sequences. A list of the sequences used to compile this information and how they were selected is given below.

Content statistics

In-frame hexamers. It has long been known that synonymous codons are not used with equal frequencies and that different organisms differ in their patterns of codon usage (Grantham *et al*., 1980; Bennetzen & Hall, 1982; McLachlan *et al*., 1984). These facts led to the *codon usage* statistic developed by Staden (Staden & McLachlan, 1982) and modified by Gribskov (Gribskov *et al*., 1984). These tests compare the frequencies of three-long words in a sequence with frequencies seen in coding regions. A contextual effect on codon usage has been observed that manifests itself as correlations between nucleotide positions in adjacent codons (Lipman & Wilbur, 1983; Yarus & Foley, 1985). Claverie was the first to show the utility of measuring the frequencies of words of length greater than three for intron–exon discrimination (Claverie & Bougueleret, 1986; Bougueleret *et al*., 1988). The *in-frame hexamer* or *dicodon* statistic measures both the effects of codon bias and correlations between neighboring positions. We have generalized Gribskov's approach and implemented the in-frame hexamer statistic of a log-likelihood ratio because, unlike Staden's method, it considers each reading frame independently and better reflects the statistical significance of long open reading frames of favorable hexamer usage. Here, the in-frame hexamer score for the interval starting at nucleotide i and ending at j, $IF_6(i, j)$, is calculated as follows:

$$IF_6(i,j) = \max \begin{cases} \sum_{k=0,3,6,\ldots,j-6} \ln\left(\frac{f_k}{F_k}\right) \\ \sum_{k=1,4,7,\ldots,j-6} \ln\left(\frac{f_k}{F_k}\right) \\ \sum_{k=2,5,8,\ldots,j-6} \ln\left(\frac{f_k}{F_k}\right) \end{cases} \quad (1)$$

where f_k is the frequency, in the table of in-frame hexamers in human coding sequences, of the hexamer starting at position k in the interval (i.e. position $i + k$ in the sequence). F_k is the frequency of same hexamer in a random population based on the base composition of the sequence:

$$F_k = \prod_{b=1}^{6} \nu_b \quad (2)$$

where ν_b is the frequency of nucleotide b in the locus under consideration. Hexamers whose occurrences equal that expected by composition have $IF_6 = 0$, those preferred have a positive score and those avoided, a negative score.

Local compositional complexity. Eukayrotic genomes contain large amounts of repetitive DNA sequences referred to as ''simple sequence'' DNA. These sequences are typically found in non-coding regions of the genome. In contrast, coding regions tend to be informationally rich. This property, quantified by the Shannon information (Shannon & Weaver, 1964), is a measure of the local redundancy of the sequence. Konopka & Owens (1990) have defined local compositional complexity averaged over the length of the sequence and have used this property to distinguish between coding and non-coding sequences. In GeneParser, this local entropy measure, H, using oligonucleotides of length $L = 8$, is calculated as follows:

$$H = -\sum_{k \in \{A,C,G,T\}} \left(\frac{N_k}{L}\right) \log_2\left(\frac{N_k}{L}\right) \quad (3)$$

where N_k in the number of times base k occurs in the oligonucleotide.

Figure 6 displays graphically the values of the In-frame Hexamer and Local Compositional Complexity statistics, in the upper-right and lower-left halves, respectively, for a region of the sequence HUMSAACT. Such information can be displayed for all of the statistics discussed in this section.

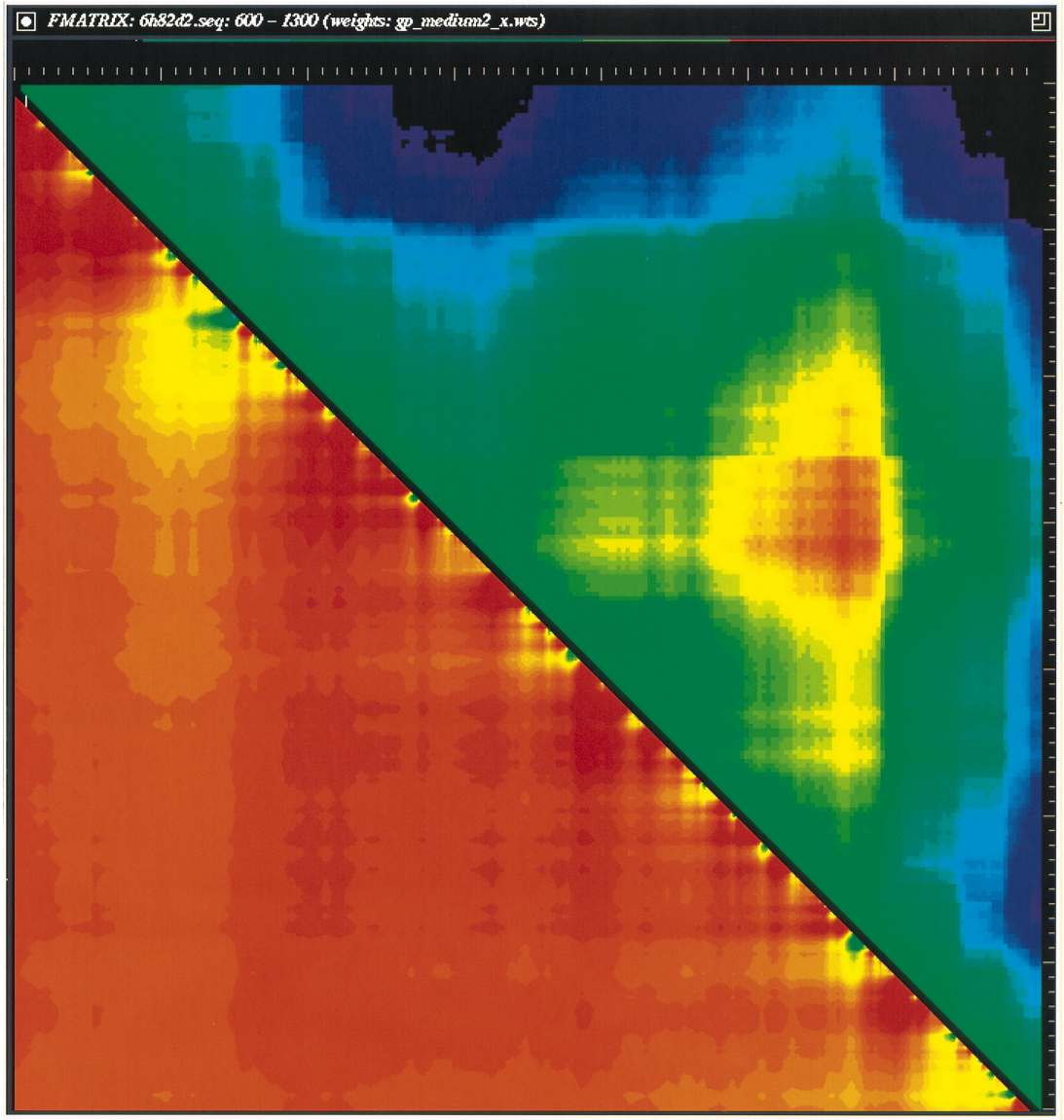

Figure 6. HUMSAACT, human skeletal alpha-actin gene, viewed as matrices for the in-frame hexamer statistic (upper right) and local compositional complexity (lower left). Colors correspond to the T-matrix score for the statistics (blue = low, yellow = intermediate, red = high). A region covering exons 2 and 3 is shown, along with adjacent introns. The bars at the top of the Figure shows the true positions of the exons (in red) and the introns (in blue) for this region of the sequence. Both exons are clearly visible as high or intermediate scoring regions in the upper (in-frame hexamer) half matrix. Similarly, both exons correspond to regions of high compositional complexity, and each intron has a relatively lower score by this measure.

Intron and exon length distributions. Introns and exons have very different extreme and average lengths (Hawkins, 1988). Even within the class of exons†, the length distributions of 3', 5' and interval exons all differ significantly from one another ($P < 0.001$ by χ^2 test). This information can be used as evidence that an interval is a member of a particular sequence type by looking up the frequency of the interval's length in a table. A low score, for example a score of zero for an intron of length less than 70 nucleotides, can be used as strong evidence that the interval is not part of the actual solution.

† The term exon to refers to the coding regions of those sequences. These length differences do not hold for the lengths of entire terminal exons that contain untranslated regions.

Bulk hexamers. In addition to considering in-frame hexamers in exons, it is useful to think of a nucleotide sequence as a series of overlapping hexamers without regard to reading frame. This approach is useful because the log-likelihood statistic applied to in-frame hexamers in exons, can be applied in this form to intron sequences as well. Claverie and others have shown that the frequencies of hexamers can differ greatly between functionally different classes of sequences and can be used to discriminate between them (Bouguelert *et al*., 1988; Claverie & Bougueleret, 1986; Claverie *et al*., 1990). The bulk hexamer score for the interval i to j belonging to sequence class X, $BH_X(i, j)$ is:

$$BH_X(i,j) = \sum_{k=i}^{j-6} \ln\left(\frac{f_k^X}{F_k}\right) \qquad (4)$$

where f_k^x is the frequency of hexamer k in sequence class X and F_k is the frequency of hexamer k in a random population based on the base composition of the locus in question. Thus, this statistic is similar to the *IF*-hexamer test except that the log-likelihood is summed over all reading frames and that each sequence class is compared with its own hexamer frequency table.

BLAST similarity scores. The program BLAST (Altschul *et al*., 1990) has become a very popular tool for identifying regions of similarity between a query sequence and sequences in the protein or nucleotide sequence databases. A BLASTX search takes a DNA sequence and translates all open reading frames for use in a search of peptide sequence databases (Gish & States, 1993). A genomic DNA sequence will generate a list of proteins to which the query sequence has similarity (assuming such sequences exist) along with a maximal-segment pair (MSP) score, which evaluates the significance of the alignment. This information can be used as evidence that an interval is an exon. Since the alignment of the query sequence and distant homologs are not likely to exactly coincide, the following heuristic is used to weight the extent of overlap between query interval and database alignment. Let (i, j) be the query interval and (x, y) be the segment with a database match with MSP score of μ. The exon BLAST score for the interval, $B_E(i, j)$ is then:

$$B_E(i,j) = \mu\left(\frac{\delta - |x-i|}{\delta}\right)\left(\frac{\delta - |y-j|}{\delta}\right) \qquad (5)$$

for all intervals within the exon BLAST decay, δ, value of the BLAST alignment interval (x, y). The score for $B_E(i, j)$ is set to zero when either boundary is more than δ nucleotides from the corresponding alignment boundary. This function provides a linearly weighted decay for the score $B_E(i, j)$ as the distance increases between the boundaries of the alignment and the query interval. A value of $\delta = 80$ was chosen as a compromise between requiring an alignment of similar length and allowing for variation in extent of overlap.

The existence of a significant alignment with sequences in the protein database can be used as evidence that a region is not an intron. In this case, the intron BLAST score, $B_I(i, j)$ is simply the score of the highest alignment contained entirely within the interval $(i + \beta, j - \beta)$, where β is the BLAST intron buffer. This buffer interval is provided to prevent alignments that extend slightly past actual exon boundaries from penalizing true intron sequences. A value of $\beta = 40$ was chosen since it is unlikely that an additional 13 amino acid residues would be added to extend an alignment by chance. The effect of this statistic is to lower the likelihood that a region with a high BLAST score (on the protein database) either overlaps or is contained within any intron.

Since the database will necessarily contain the exact training or test sequence, these exact matches need to be removed to provide a realistic simulation of what should be expected on novel sequences. To this end, test and training sequences were run using the BLAST email server running the BLASTX program (database freeze date: Feb. 1, 1993). The results were processed to filter alignments with the query sequence as well as with translated *Alu* sequences. The filters SEG (Wootton & Federhen, 1993) and XNU (Claverie & States, 1993) were applied to minimize the matches to sequences of low compositional complexity or short periodicity repeats, respectively.

Site statistics

In the present study, splice sites and translational initiation sites need to be discriminated from non-sites in the context of the genomic DNA sequence. To calculate a search matrix, the occurrence of each nucleotide in each position in the vicinity of a site needs to be known and compared with the distribution of nucleotides around non-sites. Occurrence matrices for these sites are present in the literature (Mount, 1982; Shapiro & Senapathy, 1987; Senapathy *et al*., 1990) and have been determined from analysis of GenBank sequences in this study as well. Each element in the occurrence matrix, $n_{b,i}$, represents the number of each nucleotide, b, found at each position, i, in the site. From this information, the elements of a frequency matrix, $f_{b,i} = n_{b,i} / N$ can be calculated, where N is the total number of sites in the sample. The search matrix (Stormo, 1990) is calculated as $M_{b,i} = \log_2(f_{b,i} / p_b)$ where p_b is the *a priori* frequency of base b. This frequency can be either the genomic base frequency or the local frequency, depending on the model of interest. The score for a query sequence using the search matrix represents the log-likelihood ratio that the query is a member of the matrix set compared with the random genomic population under the assumptions of additivity and independence between positions. The information content (Stormo, 1990) of the site is the dot product of frequency matrix and the search matrix and gives a measure of the importance or conservation of each position in the site. The score $S(i, j)$ for a query sequence, $s_i s_{i+1} s_{i+2} \ldots s_{j-1} s_j$, is calculated using the search matrix by the following formula:

$$S(i,j) = \sum_{k=i}^{j} M_{s_k,k} \qquad (6)$$

We have used the above methods to define and test the significance of matrices for searching for donor and acceptor splice sites. A region of about 20 nucleotides on either side of the splice site was examined. Based on the work of Shapiro & Senapathy (1987), a subset of each matrix was extracted and used for searching. A similar procedure was used to examine the context of translational initiation and termination sites. As reported previously, significant information was observed 5' of the initiating ATG; no additional information was observed in the vicinity of the stop codon†.

Dynamic programming

The basic syntactic constraints on gene structure are simple. Exons and introns alternate within a gene, they are adjacent to one another and never overlap. The first and last exons represent the 5' and 3' termini, respectively, of the gene in question. Using our working definition of an exon, these terminal exons differ in the signals that define their distal boundaries and may differ significantly in other statistical properties.

Given these constraints on gene structure, we define the optimal structure as the highest scoring combination

† Plots of the information surrounding each site and the search matrices used in GeneParser are available by anonymous ftp from beagle.colorado.edu in the file pub/GeneParser/JMB_sites.tar.Z.

of introns and exons subject to these constraints. Each interval (i, j) of sequence class k has score $L_k(i, j)$ that approximates the log-likelihood that the interval is a member of class k. This score is calculated as the weighted sum of a number of classification statistics discussed above. The score for a parse is defined as the sum of the L-matrix values for each of the component intervals. Using this definition, the DP score, D, of a valid parse might take the form:

$$D = L_f(i, j) + L_i(j + 1, k) + L_e(k + 1, l) + L_i(l + 1, m) + L_l(m + 1, n) \quad (7)$$

where L_f, L_i, L_e and L_l are the scores for first exon, introns, internal exon and last exon, respectively. Note that a valid solution can contain any or all of the sequence types as long as the syntactic constraints are followed (i.e. that $i < j < k < l < m < n$, that introns and exons alternate, and that terminal exons occur only at their respective ends in the solution).

These constraints can be used to write a DP recursion that will efficiently search the space of all potential solutions for the optimum. The recursion can be written in a general form for an N-state classifier when coupled with a table of allowed transitions. This general form is:

$$D_z(j) = \max \begin{cases} L_z(1, j) \\ \max\limits_{x \in N} \left[\max\limits_{k:2 \to j - m_z} [L_z(k, j) + D_x(k - 1)] \right] \\ 0 \end{cases} \quad (8)$$

where $D_z(j)$ is the score of the highest-scoring combination of intervals ending in an interval of type z that ends at position j. N is the set of possible transitions. The value m_z is the minimum length of sequence type z.

As the solution is built from left to right along the sequence for each position j, there are three cases to be evaluated. $D_z(j)$ is the maximum of these possibilities:

$$D_z(j) = L_z(1, j)$$

The interval from 1 to j exactly defines an interval of type z. $D_z(j)$ is set equal to this number since the solution up to j consists entirely of sequence type z.

$$D_z(j) = \max_{x \in N} \left[\max_{k:2 \to j - m_z} [L_z(k, j) + D_x(k - 1)] \right]$$

The segment from 1 to j ends in an interval of type z. Consider the example of adding an exon interval to the solution (i.e. where $z = E$ for exon and $x = I$ for intron). The score for the exon whose 5'-end is at position k is $L_E(k, j)$. $D_I(k - 1)$ contains the score of the best combination of intervals ending in an intron at position $k - 1$. $D_E(j)$ is then the score for the best combination of the exon $L_E(k, j)$ plus the best previous solution ending in an intron compatible with that exon (i.e. ending at position $k - 1$).

$$D_x(j) = 0$$

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).

Zero is included as a case for two reasons. First, it allows the recursion to predict no gene where no intervals exceed the threshold. It also relaxes the requirement for the solution to span the entire sequence. This is sensible since it is very unlikely that the input sequence will exactly span the gene found within. This is analogous to the use of zero to initialize the DP matrix in the local alignment algorithm of Smith & Waterman (1981).

Suboptimal solutions

Using this approach, there are two possible methods to generate suboptimal solutions. First, the score for each interval in the optimal solution (one at a time) can be set to $-\infty$ and the DP step re-run. This forces DP to find the next highest solution that does not include the deleted interval. This approach is desirable because it can be used to generate an exhaustive list of suboptimal solutions. However, to obtain an exhaustive list of ranked solutions, the combinatorics become prohibitive. Furthermore, to be computationally efficient, all L-matrices must be stored in computer memory simultaneously. For nucleic acid sequences of biologically interesting length, this is impractical.

A more efficient approach is to calculate a forward (as before) and reverse pass through the sequence with DP. Given the DP vectors for the forward direction D_f, D_i, D_e and D_l, let D'_f, D'_i, D'_e and D'_l be the DP vectors for the reverse direction. Just as $D_i(j)$ contains the best score for a solution ending in an intron at position j, so $D'_i(j)$ is the best score for a solution beginning with an intron at position j. A new set of vectors can be constructed to represent the score of the optimal solution that contains a given transition at a given position on the sequence. These vectors can be calculated as follows:

$$\begin{aligned}
S_{-f}(j) &= D'f(j) && \text{NULL – – first exon} \\
S_{fi}(j) &= D_f(j) + D'_i(j + 1) && \text{first exon – – intron} \\
S_{ie}(j) &= D_i(j) + D'_e(j + 1) && \text{intron – – internal exon} \\
S_{ei}(j) &= D_e(j) + D'_i(j + 1) && \text{internal exon – – intron} \\
S_{il}(j) &= D_i(j) + D'_l(j + 1) && \text{intron – – last exon} \\
S_{l-}(j) &= D_l(j) && \text{last exon – – NULL}
\end{aligned} \quad (9)$$

Suboptimal solutions can be generated by ranking the values in the S-vectors, descending through the list and generating solutions from the corresponding junctions. The use of a linked-list-like data structure greatly facilitates generation of these solutions.

Delta network for dynamic programming

Several methods used to evaluate a sequence's coding potential are described above. This information needs to be combined into a single number that can be used to classify the sequence interval as a member or non-member of a particular class. This information represents the log-likelihood information used by the DP algorithm. This section describes the application of neural networks to this problem and how the method has been adapted for use with DP.

The use of DP in sequence parsing presents an interesting challenge for the application of neural networks. To a first approximation, the desired target is to have all true exons score higher than non-exons (and likewise for introns). This could be accomplished by picking intervals at random along with true exons from many different genes. Using the raw classification statistics as input, the former could be trained to a target value of 0 and the latter to a target of 1. This would cause DP to pick only true exons (and introns) for the solution. This approach was tried with only marginally successful

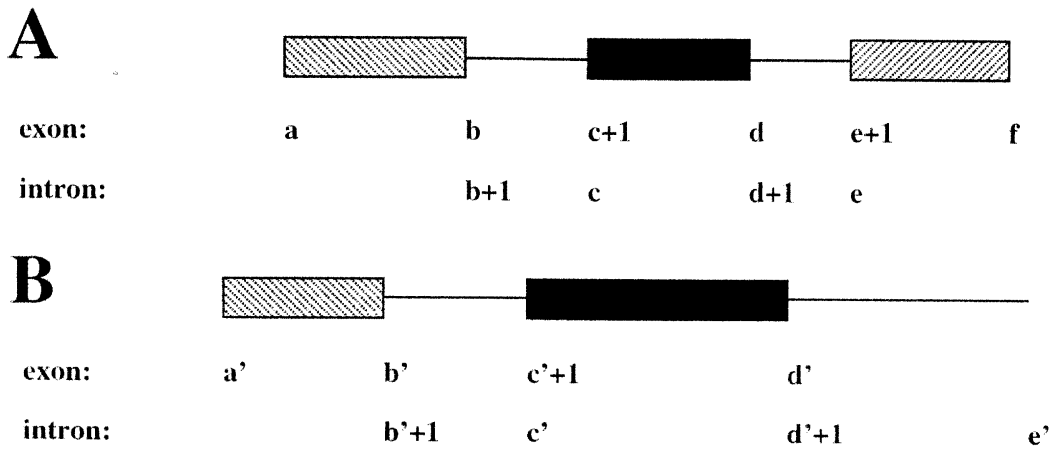

Figure 7. Derivation of DP training vectors. Given the actual solution in part A and an incorrect solution in part B, a training vector can be calculated by subtracting the incorrect solution from the correct solution. See equation (12).

results (data not shown). Clearly, all exons cannot be expected to have similar statistics and, more importantly, all genes cannot be expected to have similar statistical properties. An approach that alleviates these two major problems involves training the neural network on complete solutions instead of single intervals. That is, we expect that for each mRNA the correct solution should score higher than any incorrect solution on that mRNA, even though there may be higher-scoring incorrect solutions on other mRNAs.

Consider the DP score for a solution as calculated in equation (7). Let $D^{T,u}$ be the score of a correct (true) solution for sequence μ and $D^{F,u}$ be the score of an incorrect (false) solution on the same sequence. A perfect set of statistics and network weights would make

$$D^{T,u} > D^{F,u} \tag{10}$$

for all μ for all possible $D^{F,u}$. Subtracting $D^{F,u}$ from $D^{T,u}$ yields the inequality

$$D^{T,u} - D^{F,u} > 0. \tag{11}$$

Consider the pair of solutions in Figure 7. Writing down *L*

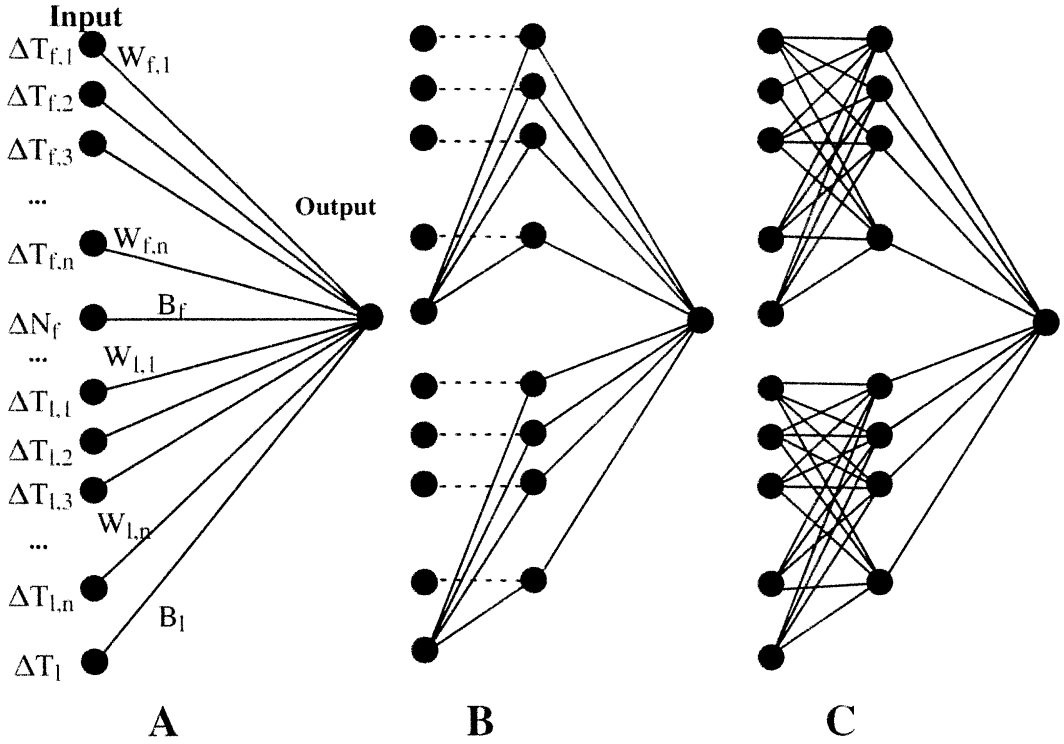

Figure 8. GeneParser network architectures. The 3 basic architectures evaluated are displayed. Network A is the simplest, consisting of a single layer of weights with 1 bias term per sequence type. The output unit has a sigmoidal activation function. Network B is functionally equivalent but with a layer of linear hidden units, which adds a bias term for each input statistic. Weights shown as broken lines are not modifiable and have the value 1. Network C has a hidden layer of units with sigmoidal activation functions.

Table 3

Training set

Locus name	Accession	Length (bp)	CDS (bp)	G + C (%)
High G + C				
HUMAPOAIT	X07496	2385	1680	62.055
HUMCG5B	X00265	2112	1164	61.979
HUMGLTH1	X06482	1020	722	70.490
HUMLYL1B	M22638	4569	1794	62.114
HUMMIS	K03474	3100	2725	71.065
HUMOTNPI	M11186	1338	859	68.909
HUMSAACT	M20543	3778	1710	62.255
HUMVPNP	M11166	2500	2135	67.880
Medium G + C				
HUMA1ATP	K02212	12,222	4886	51.383
HUMALPI	J03930	5291	3047	58.061
HUMANFA	K02043	2710	1771	52.878
HUMANT1	J04982	5768	3686	48.214
HUMAPOA4A	J02758	3613	2425	56.657
HUMAPOA4C	M14642	3604	2425	56.715
HUMAPOC2G	X05151	3840	871	53.177
HUMAPOCIA	M20902	5375	4450	53.042
HUMAPOCII	M10612	4340	876	53.433
HUMAPOE4	M10065	5515	2728	59.238
HUMATPSYB	M27132	10,186	7744	45.788
HUMBNPA	M31776	1922	1278	55.723
HUMCAPG	J04990	3734	2687	51.393
HUMCS3	M15894	2740	1584	54.380
HUMCSFGMA	M13207	3194	2128	56.481
HUMCYPIIE	J02843	14,776	11,324	50.196
HUMEF1A	J04617	4695	2314	47.370
HUMGOS19A	M23178	4102	1487	45.051
HUMGOS19B	M24110	4788	1488	45.343
HUMGHN	M13438	2657	1566	55.514
HUMHSP90B	J04988	8210	5067	49.026
HUMIL1B	M15840	7824	5959	45.105
HUMIL1BX	X04500	9721	5959	45.777
HUMKEREP	J00124	5339	4506	54.430
HUMLACTA	X05153	3310	2164	45.559
HUMMET2	J00271	1703	790	55.784
HUMMETIF1	M13003	2076	1203	56.455
HUMP45C17	M19489	8549	6582	50.684
HUMPAIA	J03764	17,509	9190	50.214
HUMPGAMMG	J05073	3771	2853	58.340
HUMPIM1A	M27903	6113	3614	55.815
HUMPLA	J00289	2967	1583	53.994
HUMPPPA	M11726	2775	829	58.378
HUMPSAA	M27274	7130	5280	54.572
HUMPSAP	M30838	4778	2386	53.851
HUMRPS17A	M18000	4029	3701	50.608
HUMSPBAA	M24461	10,476	6806	57.302
HUMTFPB	J02846	13,865	11,270	44.681
HUMTNFA	X02159	3633	1896	53.179
Low G + C				
HUMHMG14A	M21339	8882	5913	42.873
HUMIL2	J00264	5737	4790	32.317
HUMIL2A	X00695	6684	4805	32.570
HUMIL2B	K02056	5561	4805	32.386
HUMIL5	J03478	3230	1764	36.130
HUMIL5A	J02971	3241	1768	36.347
HUMPALC	M11518	7619	6872	41.383
HUMPCNA	J04718	6340	4599	43.170
HUMPRPH1	M13057	4946	2064	42.398
HUMPRPH2	M13058	4037	1939	43.151

Human genes from GeneID (Guigó *et al*., 1992) training set. These sequences were cropped for the purposes of training to encompass the first and last exons plus approximately 50 nucleotides 5' and 3', respectively. To speed training, the '' – m'' option of GeneParser was used to specify the longest interval to be considered. This values was taken as 100 plus the length of the longest intron or exon or 2000, which ever is larger. The loci names are taken from GenBank release 64 for consistency with the original paper (Guigó *et al*., 1992).

Table 4

Test set I

Locus name	Accession	Length (bp)	CDS (bp)	G + C (%)
High G + C				
HUMALPHA	J03252	4556	1599	62.116
HUMAPRT	Y00486	3016	543	65.782
HUMCYP2DG	M33189	5503	1503	61.566
HUMGAPDHG	J04038	5378	1008	60.785
HUMPNMTA	J03280	4174	849	61.811
HUMRASH	J00277	6453	570	68.185
HUMTRPY1B	M33494	2609	828	65.811
Medium G + C				
HUMEMBPA	M34462	3608	669	51.414
HUMFOS	K00650	6210	1143	51.369
HUMOBP3	M35878	10,884	876	48.833
HUMLD78A	D90144	3176	279	47.009
HUMLD78B	D90145	3112	282	49.968
HUMMETIA	K01383	2941	186	52.737
HUMP45C17	M19489	8549	1527	50.684
HUMPAIA	J03764	17,509	1209	50.214
HUMPDHBET	D90086	8872	1080	45.525
HUMPOMC	K02406	8658	804	51.444
HUMPP14B	M34046	8076	543	54.842
HUMPRCA	M11228	11,725	1386	56.913
HUMSAA	J03474	3460	369	49.653
HUMTBB5	X00734	8874	1335	56.198
HUMTCRAC	X02883	5089	426	50.147
HUMTHB	M17262	20,801	1869	50.589
HUMTKRA	M15205	13,500	705	53.274
Low G + C				
HUMHBBAAZ	M36640	2149	444	40.624
HUMHNRNPA	X12671	5368	963	43.256
HUMPRPH1	M13057	4946	501	42.398
HUMREGB	J05412	4251	501	42.249

Loci were taken from GeneID and GRAIL test sets and grouped according to G + C content. Loci HUMPLPSPC and HUMNMYCA were removed from the original set. HUMPLPSPC has alternative splicing annotated in the 3'-exon and HUMNMYCA contains a second open reading frame. Loci names are from GenBank 64 for consistency with the original papers.

values for each sequence type and grouping similar terms, the inequality becomes,

$$L_f(a, b) - L_f(a', b') + (L_i(b + 1, c) + L_i(d + 1, e)) - (L_i(b' + 1, c') + L_i(d' + 1, e')) + L_e(c + 1, d) - L_e(c' + 1, d') + L_l(e + 1, f) > 0 \quad (12)$$

At this point, it is useful to introduce a new notation that makes the classification statistics and their weights explicit:

$$D = \sum_{c \in \{f,e,i,l\}} \left[\sum_{j=1}^{N_c} \sum_{k=1}^{P_c} T_{c,j,k} w_{c,k} + N_c B_c \right] \quad (13)$$

where $T_{c,j,k}$ is the score for classification statistic k for the jth interval of type c. The term $w_{c,k}$ is the corresponding weight for that statistic and B_c is a bias term. P_c is the number of classification statistics used for sequence type c and N_c is the number of intervals of type c in the solution.

A neural network is used to find weights that satisfy the following inequality:

$$D^{T,u} - D^{F,u} = \sum_{c \in \{f,e,i,l\}} \left[\sum_{k=1}^{P_c} \left[w_{c,k} \underbrace{\left(\sum_{j=1}^{N_c^T} T_{c,j,k}^{T,\mu} - \sum_{j=1}^{N_c^F} T_{c,j,k}^{F,\mu} \right)}_{\Delta T} \right] + \overbrace{(N_c^{T,\mu} - N_c^{F,\mu})}^{\Delta N} B_c \right] \quad (14)$$

Table 5

Test set II

Locus name	Accession	Length (bp)	CDS (bp)	G + C (%)
High G + C				
HUMAZCDI	M96326	5002	756	60.756
HUMMKXX	M94250	3308	432	65.478
HUMTRPY1B	M33494	2609	828	65.811
Medium G + C				
HUMAGAL	M59199	13,662	1125	53.045
HUMAPEXN	D13370	3730	957	48.418
HUMCACY	J02763	3671	273	57.123
HUMCHYMB	M69137	3279	744	51.113
HUMCOLA	M95529	3401	333	53.661
HUMCOX5B	M59250	2593	390	49.826
HUMCRPGA	M11725	2480	675	48.145
HUMGOS24B	M92844	3135	981	60.223
HUMHAP	M92444	3046	957	48.194
HUMHEPGFB	M74179	6100	2136	59.852
HUMHLL4G	M57678	4428	408	56.843
HUMHPARS1	M10935	11,551	1221	46.152
HUMI309	M57506	3709	291	48.665
HUMKAL2	M18157	6139	786	56.524
HUMMHCP42	M12792	5141	1485	58.880
HUMNUCLEO	M60858	10,942	2124	45.403
HUMPEM	M61170	4243	1428	58.944
HUMPP14B	M34046	8076	543	54.842
HUMPROT1B	M60331	1306	156	51.991
HUMPROT2	M60332	1861	309	57.174
HUMRPS14	M13934	5985	456	48.805
HUMSHBGA	M31651	6087	1209	53.754
HUMTNFBA	M55913	2140	618	57.477
HUMTNFX	M26331	3103	702	53.529
HUMTNP2SS	L03378	1782	417	48.316
HUMTRHYAL	L09190	9551	5697	50.717
Low G + C				
HUMGFP40H	M30135	4379	435	41.676
HUMHIAPPA	M26650	7160	270	33.268
HUMIL8A	M28130	5191	300	33.288
HUMMGPA	M55270	7734	312	39.436
HUMPALD	M11844	7616	444	41.360

Selected complete genomic sequences from GenBank Release 77.

When written in this form, one can see a simple network implementation to solve this inequality. The inputs are simply $T^T - T^F$ for each statistic for each sequence type (ΔT) and the difference between the number of each sequence type in the actual and predicted solutions (ΔN). This network design is called the Delta Network because the net is trained on the difference between the correct solution and an incorrect solution for a particular sequence. If the right side of equation (14) is passed through a squashing function such as the symmetrical sigmoid:

$$g(x) = \frac{1}{1 + e^{-x}} - \frac{1}{2} \qquad (15)$$

then training to a target of 0.5 will maximize the difference between correct and incorrect solutions.

† The momentum term is the fraction of the previous weight change added to the newly calculated weight change. This increases the effective learning rate by increasing the progress made in the direction of the average gradient.

Network architectures

We evaluated several neural network architectures in this study. These networks are shown in Figure 8. Network A is the simplest architecture possible. The input statistics are subject to a single layer of weights with one bias term for each sequence type. This architecture has the advantage of being fast both in training and when incorporated into GeneParser. Previously, we had evaluated a network of type B in the context of the two-state (internal exon–intron) classification problem (Snyder & Stormo, 1993). In network B each input statistic is fed into an intermediate linear unit *via* an unmodifiable weight (clamped to unity). Each intermediate unit also receives a bias term from the same sequence class as the input statistic *via* a modifiable weight. Network B formally reduces to network A in which there is a single bias term.

Networks A and B are both linear networks in that they have either a single layer of weights or have multiple layers connected by hidden units with linear activation functions. Network C is a multilayered network with the intermediate layer having sigmoidal activation functions. The hidden layer has complete connectivity with the input units within each sequence type. This architecture was trained using standard backward error propagation methods (backprop; Hertz *et al*., 1991).

Neural network training procedure

GeneParser proved to be remarkably insensitive to the network training procedure used. In fact, the actual level of accuracy to which the network is trained at each training iteration (see below) influenced only the number of GeneParser training iterations required to reach plateau, not the final level of accuracy (data not shown). We settled on training for 5000 epochs using batch update, a learning rate, η, of 0.0001, and a momentum† of 0.9. A bail-out criterion of 1% was used to terminate training when greater than 99% of the training vectors had been learned. Finally, this process was repeated three to five times with randomly chosen initial weights in the range $-0.1 < w < +0.1$ and the best weights saved for use in the next pass of GeneParser through the data.

GeneParser training

The network is trained by allowing GeneParser to predict the structure of a number of genes (training sets are described below) starting with random weights. In the first pass through the training sequences, all of the GeneParser solutions will be incorrect. Training vectors are calculated for all incorrect solutions, as in equation (14). This pool of vectors is used to train the neural network used in GeneParser by minimizing the error on these vectors. Using the newly obtained weights, GeneParser is rerun on the training sequences to generate a new collection of potentially incorrect but different solutions. These solutions are used to make additional training vectors, which are added to the pool. Each cycle of GeneParser prediction and neural network training is referred to as a ''training iteration''. As the network weights begin to improve, GeneParser will generate correct solutions and sometimes duplicate incorrect ones. Training vectors derived from these solutions are filtered from the training pool.

Table 6

Calculation of performance measures

	Predicted Positives	Predicted Negatives
Actual Positives	True Positives	False Negatives
Actual Negatives	False Positives	True Negatives
Senstivity	$Sn = \frac{TP}{TP + FN}$	
Specificity	$Sp = \frac{TP}{TP + FP}$	
Correlation coefficient	$CC = \frac{(TP)(TN) - (FP)(FN)}{\sqrt{(PP)(PN)(AP)(AN)}}$	

Data sets

Content and site statistics

Starting from a list of 19,651 human loci from GenBank release 77, entries were selected that contained full-length genomic DNA sequences with CDS (protein coding sequence) annotated. Entries containing pseudogenes or annotation indicating alternative splicing or ''putative CDS'' were eliminated along with genes for histocompatibility antigens and immunoglobulins. This resulted in a collection of 333 loci (Snyder, 1994)†. Hexamer tables and weight matrices were calculated based on these sequences. Some authors have taken pains to remove or down-weight related sequences from their sequence pool (Guigó *et al*., 1992). We have chosen not to do this because such redundancies are likely to reflect the types of genes to be sequenced in the future.

Training set

The training set for the program GeneID (Guigó *et al*., 1992) serves as the starting point for the GeneParser training sets. The GeneID training set consists of 169 vertebrate genes for which both mRNA and genomic sequence are known and that possess at least one internal exon. Mutants, pseudogenes, genes that exhibit alternative splicing or have a length of greater than 15 kb were discarded. Immunoglobulins and histocompatibility antigens were also eliminated. For training GeneParser, the remaining 59 human genes were extracted and will be referred to as the Training Set. These genes are listed in Table 3.

We observe here and others have reported (Uberbacher & Mural, 1993) that the performance of many gene identification programs is sensitive to G + C content. To minimize the effect of the wide variations in base composition in the human genome, the genes in the Training Set were classified on the basis of G + C content by comparison with a distribution of all full-length human gene loci in GenBank release 77 (using selection criteria similar to above). Genes within one standard deviation of the mean were considered ''medium G + C content''. Genes with a G + C content greater than one standard deviation above or below the mean were classified as ''high'' and ''low'' G + C content, respectively.

† These loci names are available by anonymous ftp from beagle.colorado.edu in the file pub/GeneParser/JMB_loci.Z.

Test sets

A test set of human genes was compiled based on the human genes used in the testing of GRAIL and GeneID. These genes were not used in the development of these programs and were not present in the training set of GeneParser. These genes are listed in Table 4. It is important to use separate training and test sets because it is often possible to obtain arbitrarily high levels of performance when a program is trained on a small set of genes. However, such performance is not necessarily indicative of the likely performance when tested on novel data. Particularly in systems that use complex multilayered neural networks, it is often possible for the network to ''memorize'' the idiosyncrasies of training data while failing to internalize the salient properties of the population as a whole.

A second test set was developed to test the effect that specific genes may have on performance. This set was derived from every third locus from the collection of 333 genes used to group the Training Set by G + C content. Loci that were present in the Training Set or test sets of GeneID or GRAIL were detected, as were loci longer than 30 kb and those that contained less than two exons. This resulted in the 34 loci in Test Set II shown in Table 5. This test set should not be considered representative of human genes in general or even of human genes in GenBank. However, it does provide a test set largely independent of previous test and training data sets.

''Error-prone'' data sets

We have assumed up to this point that the GenBank loci used in this study are error-free. While this is most likely not the case in general, it is very likely true that the sequences are free from frame-shift errors in the coding region since in most cases some independent evidence has been used to establish that the CDS as annotated is the CDS of the gene in question. To give an indication of how GenParser performs on sequences with known error frequency, GeneParser was tested on sequences from Test Set I with a known frequency of substitution and frame–shift errors introduced randomly. To test if it is possible to train the program to cope with these errors, GeneParser was also trained on the high G + C content genes from the Training Set with errors introduced at known frequency.

Performance measures

To evaluate performance, we focus on the number of nucleotides predicted to be coding and the number of exons correctly predicted. To facilitate the comparison of methods, we use three measures of performance (Brunak *et al*., 1991): sensitivity (*Sn*), specificity (*Sp*), and the correlation coefficient (*CC*). Table 6 illustrates how each of these scores is calculated. We report two additional numbers to evaluate performance at the level of complete exons. ''Exons Correct'' refers to the fraction of exons predicted exactly (donor/start site and acceptor/stop site correct). ''Exons Overlapped'' refers to the fraction of actual exons which overlap predicted exons.

Acknowledgements

We thank Roderic Guigo and Richard Mural for discussions on the general problem of gene identification, and Alan Lapedes and Andrzej Ehrenfeucht for helpful

suggestions on the use of neural networks in this system. This work benefitted from discussions at the Recognizing Genes workshop at the Aspen Center for Physics. This work was supported by NIH grant HG00249 and DOE grant ER61606.

References

Altschul, S. F., Gish, W., Myers, E. W. & Lipman, D. J. (1990). Basic local alignment search tool. *J. Mol. Biol.* **215**, 403–410.

Bennetzen, J. L. & Hall, B. D. (1982). Codon selection in yeast. *J. Biol. Chem.* **257**, 3026–3031.

Bernardi, G. (1989). The isochore organization of the human genome. *Annu. Rev. Genet.* **23**, 637–661.

Bougueleret, L., Tekaia, F., Sauvaget, I. & Claverie, J.-M. (1988). Objective comparison of exon and intron sequences by the means of 2-dimensional data analysis methods. *Nucl. Acids Res.* **16**, 1729–1738.

Bridle, J. S. & Sedgwick, N. C. (1977). A method for segmenting acoustic patterns, with application to automatic speech recognition. *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, **27**, 656–659.

Brunak, S., Engelbrecht, J. & Knudsen, S. (1991). Prediction of human mRNA donor and acceptor sites from the DNA sequence. *J. Mol. Biol.* **220**, 49–65.

Claverie, J.-M. & Bougueleret, L. (1986). Heuristic informational analysis of sequences. *Nucl. Acids Res.* **14**, 179–196.

Claverie, J.-M. & States, D. J. (1993). Informational enhancement methods for large scale sequence analysis. *Comp. Chem.* **17**, 191–201.

Claverie, J.-M., Sauvaget, I. & Bougueleret, L. (1990). *k*-Tuple frequency analysis from intron / exon discrimination to T-cell epitope mapping. *Methods Enzymol.* **183**, 237–252.

Cohen, J. R. (1981). Segmenting speech using dynamic programming. *J. Acoust. Soc. Amer.* **69**, 1430–1438.

Dong, S. & Searls, D. B. (1994). Gene structure prediction by linguistic methods. *Genomics*, **23**, 540–551.

Fickett, J. W. & Tung, C.-S. (1992). Assessment of protein coding measures. *Nucl. Acids Res.* **20**, 6441–6450.

Fields, C. A. & Soderlund, C. A. (1990). Gm: a practical tool for automating DNA sequence analysis. *Comput. Appl. Biol. Sci.* **6**, 263–270.

Gelfand, M. D. & Roytberg, M. A. (1993). Prediction of the exon–intron structure by a dynamic programming approach. *BioSystems*, **30**, 173–182.

Gish, W. & States, D. J. (1993). Identification of protein coding regions by database similarity search. *Nature Genet.* **3**, 266–272.

Grantham, R., Gautier, C., Gouy, M., Mercier, R. & Pavé, A. (1980). Codon catalog usage and the genome hypothesis. *Nucl. Acids Res.* **8**, r49–r60.

Green, P., Lipman, D., Hillier, L., Waterston, R., States, D. & Claverie, J.-M. (1993). Ancient conserved regions in new gene sequences and the protein databases. *Sciences*, **259**, 1711–1716.

Gribskov, M., Devereux, J. & Burgess, R. R. (1984). The codon preference plot: graphic analysis of protein coding sequences and prediction of gene expression. *Nucl. Acids Res.* **12**, 529–549.

Guigó, R., Knudsen, S., Drake, N. & Smith, T. (1992). Prediction of gene structure. *J. Mol. Biol.* **226**, 141–157.

Hawkins, J. D. (1988). A survey on intron and exon lengths. *Nucl. Acids Res.* **16**, 9893–9908.

Hertz, J., Krogh, A. & Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, CA.

Knight, J. & Myers, E. (1995). Super pattern matching. *Algorithmica*, **13**, 211–243.

Konopka, A. K. & Owens, J. (1990). Complexity charts can be used to map functional domains in DNA. *Gene Anal. Techn. Appl.* **7**, 35–38.

Koonin, E., Bork, P. & Sander, C. (1994). Yeast chromosome III: new gene functions. *EMBO J.* **13**, 493–503.

Lipman, D. J. & Wilbur, W. J. (1983). Contextual constraints of synonymous codon choice. *J. Mol. Biol.* **163**, 377–394.

McLachlan, A. D., Staden, R. & Boswell, D. R. (1984). A method for measuring the non-random bias of codon usage. *Nucl. Acids Res.* **12**, 9567–9575.

Mouchiroud, D., D'Onofrio, G., Aissani, B., Macaya, G., Gautier, C. & Bernardi, G. (1991). The distribution of genes in the human genome. *Gene*, **110**, 181–187.

Mount, S. M. (1982). A catalog of splice junction sequences. *Nucl. Acids Res.* **10**, 459–472.

Roberts, L. (1991). Finding DNA sequencing errors. *Science*, **252**, 1256–1257.

Searls, D. B. (1993). The computational linguistics of biological sequences. In *Artificial Intelligence and Molecular Biology*, pp. 47–120. AAAI Press, Cambridge, MA.

Searls, D. B. & Dong, S. (1993). A syntactic pattern recognition system for DNA sequences. In *Proceedings of the Second International Conference on Bioinformatics, Supercomputing and Complex Genome Analysis* (Lim, H. A., Fickett, J., Cantor, C. R. & Robbins, R. J., eds), pp. 89–101, World Scientific, Teaneck, NJ.

Senapathy, P., Shapiro, M. B. & Harris, N. L. (1990). Splice junctions, branch point sites, and exons: sequence statistics, identification, and application to genome project. *Methods Enzymol.* **183**, 252–278.

Shannon, C. E. & Weaver, W. (1964). *The Mathematical Theory of Communication*, The University of Illinois Press, Urbana IL.

Shapiro, M. B. & Senapathy, P. (1987). RNA splice junctions of different classes of eukaryotes sequence statistics and functional implications in gene expression. *Nucl. Acids Res.* **15**, 7155–7174.

Smith, T. F. & Waterman, M. S. (1981). Identification of common molecular subsequences. *J. Mol. Biol.* **147**, 195–197.

Snyder, E. E. (1994). Identification of protein coding regions in genomic DNA. PhD thesis, University of Colorado, Boulder, CO.

Snyder, E. E. & Stormo, G. D. (1993). Identification of coding regions in genomic DNA sequences: an application of dynamic programming and neural networks. *Nucl. Acids Res.* **21**, 607–613.

Snyder, E. E. & Stormo, G. D. (1994). Identifying genes in genomic DNA sequences. In *Nucleic Acid and Protein Sequence Analysis: A Practical Approach*, 2nd edit., IRL Press, Oxford, in the press.

Soderlund, C., Schanmugam, P., White, O. & Fields, C. (1992). gm: a tool for exploratory analysis of DNA sequence data. In *Proceedings of the Twenty- fifth Hawaii International Conference on System Sciences, Biotechnology Computing Minitrack*, vol. 1, pp. 653–662, IEEE Computer Society Press, Los Alamitos, CA.

Staden, R. & McLachlan, A. D. (1982). Codon preference and its use in identifying protein coding regions in long DNA sequences. *Nucl. Acids Res.* **10**, 141–156.

Stormo, G. D. (1987). Identifying coding sequences. In *Nucleic Acid and Protein Sequence Analysis, A Practical Approach* (Bishop, M. J. & Rawlings, C. J., eds), pp. 231–258. IRL Press, Oxford.

Stormo, G. D. (1988). Computer methods for analyzing sequence recognition of nucleic acids. *Annu. Rev. Biophys. Chem.* **17**, 241–263.

Stormo, G. D. (1990). Consensus patterns in DNA. *Methods Enzymol.* **183**, 211–221.

Uberbacher, E. C. & Mural, R. J. (1991). Locating protein-coding region in human DNA sequences by a multiple sensor–neural network approach. *Proc. Nat. Acad. Sci., U.S.A.* **88**, 11261–11265.

Uberbacher, E. C. & Mural, R. J. (1993). *GRAIL documentation*.

Widrow, B. & Walach, E. (1984). On the statistical efficiency of the LMS algorithm with nonstationary inputs. *IEEE Trans. Inf. Theory,* **30**, 211–221.

Wilson, R., Ainscough, R., Anderson, K., Baynes, C., Berks, M., Bonfield, J., Burtoni, J., Connell, M., Copsey, T. & Cooper, J. (1995). 2.2Mb of continuous nucleotide sequence from chromosomeIII of *C. elegans. Nature (London),* **368**, 32–38.

Wonnacott, R. J. & Wonnacott, T. H. (1985). *Introductory Statistics*, pp. 486–496, John Wiley and Sons, New York.

Wootton, J. C. & Federhen, S. (1993). Statistics of local complexity in amino acid sequences and sequence databases. *Comp. Chem.* **17**, 149–163.

Xu, Y., Einstein, J. R., Mural, R. J., Shah, M. & Uberbacher, E. C. (1994*a*). An improved system for exon recognition and gene modeling in human DNA sequences. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology* (Altman, R., Brutlag, D., Karp, P., Lathrop, R. & Searls, D., eds), pp. 376–384, AAAI Press, Menlo Park, CA.

Xu, Y., Mural, R. J. & Uberbacker, E. C. (1994*b*) Constructing gene models from accurately predicted exons: an application of dynamic programming. *Comput. Appl. Biol. Sci.* **10**, 613–623.

Yarus, M. & Foley, L. S. (1985). Sense codons are found in specific contexts. *J. Mol. Biol.* **182**, 529–540.

Zucker, M. (1989). On finding all suboptimal foldings of an RNA molecule. *Science,* **244**, 48–52.

Edited by F. E. Cohen

(Received 14 September 1994; accepted 20 January 1995)