Finding Genes in DNA using Decision Trees and Dynamic Programming

Steven Salzberg, Xin Chen, John Henderson, and Kenneth Fasman
Department of Computer Science and Division of Biomedical Information Sciences
Johns Hopkins University
Baltimore, MD 21218
{salzberg,xchen,jhndrsn}@cs.jhu.edu,ken@gdb.org

Abstract

This study demonstrates the use of decision tree classifiers as the basis for a general gene-finding system. The system uses a dynamic programming algorithm that finds the optimal segmentation of a DNA sequence into coding and noncoding regions (exons and introns). The optimality property is dependent on a separate scoring function that takes a subsequence and assigns to it a score reflecting the probability that the sequence is an exon. In this study, the scoring functions were sets of decision trees and rules that were combined to give the probability estimate. Experimental results on a newly collected database of human DNA sequences are encouraging, and some new observations about the structure of classifiers for the gene-finding problem have emerged from this study. We also provide descriptions of a new probability chain model that produces very accurate filters to find donor and acceptor sites.

Introduction

Finding genes in eukaryotic DNA is a problem of central importance to genomic research. The large quantities of raw DNA sequence being produced on a daily basis all over the world in a wide range of projects virtually demand new methods for automatic analysis. One of the most important steps in the analysis of new DNA is finding out whether or not it contains any genes, and if so, determining exactly where they are. A number of methods have been developed for this problem and turned into systems, some of which are used by biologists around the world on a daily basis. Some of the best known systems are GRAIL (Xu, Mural, & Uberbacher 1994), GeneID (Guigo *et al.* 1992), GeneParser (Snyder & Stormo 1993; 1995), and FGENEH (Solovyev, Salamov, & Lawrence 1994). Although none of these programs solve the gene finding problem, all of them perform well enough to greatly narrow down the search for genes, which is a valuable service. Reseach continues on ways to improve these systems and develop new approaches. This paper describes a new approach, based on a combination of decision tree classifiers, hand-coded rules, and dynamic programming.

Decision tree algorithms are an important, well established class of machine learning techniques. They have been used for a wide range of applications, especially for classification problems (Breiman *et al.* 1984; Quinlan 1993; Murthy, Kasif, & Salzberg 1994). In a recent study, we found that decision trees can produce accurate classifications of relatively short sequences of DNA (as little as 54bp) (Salzberg 1995). That study considered only a special case of gene-finding, where the task was to distinguish between subsequences that were either entirely coding or entirely noncoding. The full gene-finding problem is more complex, and has been posed variously as a parsing problem, a probabilistic reasoning problem, and as a classification problem. The coding recognition module of GRAIL (Xu, Mural, & Uberbacher 1994) is an example of a classifier (a neural net classifier,trained using the back propagation algorithm) that is currently used as part of a gene-finding system.

The main framework of our system is a dynamic programming algorithm that can efficiently consider the large number of alternative parses that are possible for any sequence of DNA. We modified the standard decision tree algorithm so that it could produce a probability score, which is needed by the dynamic programming algorithm. These modifications are described below. In addition, we found that some of the decision trees had uncovered very simple rules, which were easy to hand-code directly. In these cases (explained below) we simply replaced the decision tree classifiers with rules. The resulting combined system, which consists of a set of decision trees and rules embedded in a dynamic programming system, is the first complete gene-finding system based on the decision tree approach. This paper reports on the results of this newly developed algorithm as applied to a large set of human DNA sequences. Although the system is only in its first version and is relatively unrefined and untuned, it still performs quite well on a test set of previously unseen sequences.

In order to refine and test our methods, we extracted

a large collection of human genomic DNA sequences from the Genome Sequence Data Base (GSDB). We then carefully filtered this data to remove any obvious inconsistencies, and we only retained complete coding sequences for our experiments. This data is described more fully in Section , after which we describe our algorithms and our experimental results.

Assembling a collection of complete DNA sequences

The classification rules used in a gene finding system must be tuned to recognize features of coding and non-coding sequences. This tuning process involves the adjustment of various algorithm parameters, and the final accuracy of the system depends to a large extent on the quality and quantity of data used to train them. We therefore placed considerable emphasis on the development of a reliable training set. (We are using this same collection of data in a companion study on a gene-finding system based on Hidden Markov Models (Henderson, Salzberg, & Fasman 1996).)

Ad hoc query access to scientific databases via a standard query mechanism is essential for a study of this kind (Waterman *et al.* 1994). Large numbers of DNA sequences must be gathered which meet rigorous criteria. The Genome Sequence Data Base (Cinkosky, Fickett, & Keen 1995) was therefore the clear choice as our primary source of human DNA sequences. Among the major public nucleotide sequence databases (including Genbank, EMBL Data Library, and DDBJ), GSDB is the only one that supports publicly accessible SQL queries. This feature is of critical importance in obtaining a well-defined set of sequences from the database, and easily updating that set as new data becomes available.

The defining features of all sequences in our data set are as follows. Each sequence:

- is obtained from human DNA
- is greater than 500 nucleotides in length
- contains a complete coding sequence (CDS), including both 5′ and 3′ ends
- contains at least one exon

Because of GSDB's implementation in a relational database management system (Sybase), pointers to the basic data set can be obtained with a single query, which is impossible with any other public sequence database. The final version of our data, which was retrieved in early August 1995, will soon be made available on our Web site.

It is well known that the quality of the entries in the DNA sequence databases varies dramatically (e.g., (Kristensen, Lopez, & Prydz 1992; White *et al.* 1993)). Poor quality sequence data can be caused by everything from vector contamination to erroneous annotation of sequence features. Although each of the contributing databases has its own quality control checks, they are not uniformly applied at all sites.

Data set	Num. of sequences	Total length	Total exon length
Training	295	2147903	474241
Test	100	716226	146473

Table 1: Characterization of the human DNA data.

Once our initial data set was obtained, a number of quality controls had to be applied to remove entries which did not meet our standards. We applied filters to the raw data set to find and remove sequences with overlapping exon and intron ranges or with exons defined out of the range of the coding region. We also manually removed the entry for the human germline T-cell receptor beta chain gene (Hood *et al.* 1993) from our data set. Although this entry is the largest single human sequence currently in the database, and it is a model for careful, dense annotation, it is atypical because of the V, D, and J segment structure. This, combined with its very large size made it very difficult to process, and ultimately easier to leave out of our data set.

Even after this filtering, we expect that many errors remain in the data. For example, we have no easy way of detecting genes where alternative splicing occurs, and it is often impossible to detect insertion and deletion errors. A more fundamental problem is that annotations placed in the data base are inconsistent: we have encountered many cases where the first exon was apparently defined as beginning at the ATG site, but there are many others where the first exon begins upstream of the ATG. Likewise, many final exons are annotated as ending with a stop codon, while many others include the stop codon in the middle. The final data set is described in Table 1. We are also using a second, smaller data set that was part of a large comparative study of GRAIL, GeneParse, GeneID, FGENEH, GenLang, SORFIND, and XPound (Burset & Guigo 1996). This data set contained 570 vertebrate sequences, from which we selected all the human sequences, giving us 93 sequences. (The accession numbers and data can be obtained by ftp; send email to salzberg@cs.jhu.edu for instructions.) For this data, we used 63 sequences for training and reserved 30 for testing. This data is "cleaner" than ours in that many problems were filtered out, such as sequences with non-consensus splice sites, sequences whose first exon did not start with ATG, sequences whose last exon did not end with a stop codon, and others (see (Burset & Guigo 1996) for details). Although cleaner, this data set provides a basis for comparison with other systems and was included for this reason. All of the training and tuning of the gene-finding system was conducted exclusively on the training sets. After training, the system was run without further adjustments on the test data.

Decision trees for classifying coding regions

Our decision tree software is based on the OC1 decision tree system (Murthy, Kasif, & Salzberg 1994), which is available freely over the Internet[1]. The OC1 system was shown in a previous study to be very effective at labeling short homogeneous pieces of DNA as coding and noncoding (Salzberg 1995). OC1 is a randomized decision tree system, which because of this property produces a slightly different decision tree on each run. We constructed ten different decision trees from our training data, and combined their outputs to label a segment of DNA with the probability that it is a coding region. Details of the system can be found in Murthy et al. (Murthy, Kasif, & Salzberg 1994).

A decision tree system produces as output a simple tree data structure to be used as a classifier, as shown in Figure 1. This figure shows part of a tree that was generated for this study. The internal nodes of the tree contain *features* that are calculated for each subsequence as it is passed to the tree. Features used here included dicodon usage, Fickett's asymmetry measure, the sequence length, and the autocorrelation, run, and Fourier measures from Fickett and Tung (Fickett & Tung 1992). To classify a sequence, it is first converted into a set of feature values, and this is the representation that is given to the decision tree. The bottom of the tree (its leaf nodes) contains class labels indicating whether the subsequence is coding (exon) or noncoding. Unlike standard decision trees, the system here needed to convert these class labels to numbers.

We first ran a simple experiment to determine how well a decision tree could distinguish between whole exons and whole introns. Using our training data set, we ran a cross-validation experiment and found that the trees could obtain 94% accuracy on this problem. Details are given in Section . This experiment convinced us that decision trees could provide a good basis for a dynamic programming approach to gene finding. The dynamic program, though, needs a numeric score for each potential exon, not just a class label, so OC1 had to be modified.

Modifying OC1 (or any decision tree system, in fact) to produce numeric scores is fairly straightforward, and works as follows. At the leaf nodes of every tree, we store not just the label of the majority class, but the distributions of all classes found at that node. For example, if a leaf node contains 40 exons and 20 introns, instead of just storing the label "exon" one can store the distribution (i.e., 40/60 exons and 20/60 introns). As stated above, we create 10 trees from the training data, which serves to make the system less sensitive to peculiarities that may exist in a single tree. To classify a new sequence, we compute the same features that were computed for all the training sequences, and then use those features to classify the sequence in each of the 10 trees. The distributions returned are then averaged, giving a probability score for each of the classes.

[1]The system is available on the Web at http://www.cs.jhu.edu/labs/compbio/home.html. It is also available via ftp at ftp.cs.jhu.edu, in the directory pub/oc1.

A dynamic programming algorithm to segment DNA

Dynamic programming is the name of a large class of algorithmic methods that use a simple principle: in order to find the optimal solution to a problem, break the problem into smaller problems, compute their optimal solution, and then glue those solutions together. Fortunately, the gene-finding problem fits this framework perfectly.

Our dynamic programming (DP) algorithm finds a segmentation of a DNA sequence into alternating coding and noncoding regions that is guaranteed to be optimal. This is possible because DP conducts an exhaustive search through all possible optimal solutions. Note that there are an exponential number of ways to parse a sequence into exons and introns; however, the advantage of the DP algorithm is that it prunes off large numbers of these possibilities, considering only those alternatives that are candidates for the optimal solution. Dynamic programming has been used for gene-finding previously by Snyder and Stormo (Snyder & Stormo 1993) in combination with a feedforward neural network. It has also been used for partitioning amino acid sequences (Sankoff 1992) and, in a similar but more general context, in our own work on partitioning sequences of arbitrarily labeled objects (Fulton, Kasif, & Salzberg 1995). Wu (Wu 1995) proposed the DP formulation most similar to ours, though it has not been implemented.

The input to our dynamic programming algorithm is a complete DNA sequence. The output is a partitioning of that sequence into an initial noncoding region, followed by alternating exons and introns, following by a final noncoding region. This partitioning is optimal with respect to the scores produced by the scoring algorithm (which in our system is a set of decision trees and rules).

The basic formulation can be expressed straightforwardly with the help of a matrix $D(t, n)$. (This notation is similar to that used by Snyder and Stormo (1995).) $D(t, n)$ will keep the score of the optimal parse of a sequence S ending in a subsequence of type t at location n. The different sequence types are:

1. initial noncoding region
2. initial exon
3. internal exon
4. intron
5. final exon
6. final noncoding region

The scoring algorithm returns a score $P_t(i, j)$ which is the probability that the subsequence from i to j is a sequence of type t. The algorithm processes a sequence

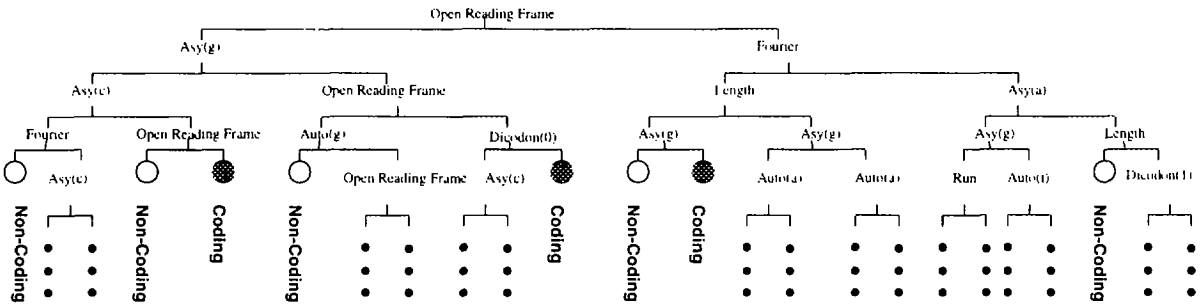

Figure 1: Sample decision tree for classifying human DNA

S from left to right, and at each location it computes the best parse by choosing:

$$D(t,n) = \max \begin{cases} \max_{1<i<n} D(t_{prev}, i) \\ +P_t(i+1..n)(n-i) \\ P_t(1..n) \end{cases}$$

That is, at each location n, to compute the score of the best parse that ends in each type t, we compute the score of the subsequence $[i+1, n]$ as a sequence of type t and add that to the score of the optimal parse on $[1, i]$ ending with type t_{prev}, which is the segment type compatible with t. These previous optimal parses are found in the appropriate D matrix. Note that the score P is normalized by the length of the sequence, so the algorithm will maximize the score per base for the whole sequence. (Snyder and Stormo (Snyder & Stormo 1993) used a different approach: they converted their probability scores to log likelihoods and then added them together. The computation here computes the average probability score per base for the sequence.)

Frame consistent dynamic programming

To this formulation we add one additional but crucial complication: in order to produce a "legal" parse, the coding regions must not contain any in-frame stop codons when concatenated together. If one simply checks for frame consistency and eliminates solutions that are not consistent, the algorithm can no longer guarantee an optimal parse (and, in fact, it will frequently produce suboptimal parses). The problem can be illustrated as follows:

```
noncoding | E1 |  I1       | E3  |
noncoding    | E2 |  I2    | E4     |
```

Suppose we have the two partial parses shown above - two different possible initial exons, in different frames, with introns that could end at the same location i. The D matrix will store only the higher-scoring parse at i. Thus if the first parse scores higher, the second parse is lost, even though the secon parse is in a different reading frame. But it is quite possible that exons 2 and 4 form a higher scoring parse than exons 1 and 3. Note that if the algorithm does not do any frame consistency checking, it *will* find the highest score, but it may not be reading frame consistent.

Therefore at every location n, we must keep a *separate* D matrix for each of the three reading frames. At each location, we keep the optimal parse ending in a sequence of type t in frame f, for $f \in (0, 1, 2)$. The system thus needs eighteen matrices.

Assumptions and constraints

There are a number of inter-region constraints and assumptions that can be made for the gene-finding problem, and many of these were incorporated into our system. These constraints are immensely valuable, for without them the algorithm will have to compute the optimal parse for every single location in every sequence, which is computationally very costly. In addition, by giving the system more possibilities to compare, it makes the gene-finding problem itself more difficult.

Fortunately, the biology community has uncovered many important facts about gene splicing and transcription, some of which can be used to help the system. The constraints we considered are as follows (although, as noted below, some were not used in the final version of the system):

1. The first exon of every gene begins with the start codon.
2. A gene can have only one stop codon in the same reading frame as the start codon, and that stop codon must appear as the last codon in the gene.
3. Each non-initial exon must be in phase with the previous exon.
4. Each piece of DNA presented for analysis will start and end with a noncoding region and contain a single gene.
5. There are a number of signals which denote boundaries between exons and introns(Solovyev, Salamov, & Lawrence 1994; Mount, Peng, & Meier 1995). We can find all boundary candidates (which is a superset of the true boundaries) by looking for these signals.

These assumptions, although helpful, have a number of shortcomings. There are many genes in which the stop codon appears in the middle of the last exon. The start codon can appear after the start of transcription,

although in our data this happened very rarely. However, our system is in reality finding the initial and the final coding region, *not* the initial and final exons, because it always uses the start and stop codons as signals of the beginnings and ends of these regions. We should note that every other major gene-finding system uses this same assumption, so in a sense, all these systems are really finding coding regions, not genes. Assumption 4 is used by our system and most other gene-finders; however, it is the one assumption that needs to be removed to create a truly general gene-finder. (After all, an uncharacterized DNA sequence may contain no genes at all, or it may contain several genes.)

Constraint 3 eliminates a large number of candidate solutions by requiring that the reading frames of all the exons in a parse be consistent; i.e., when the exons are concatenated, the resulting sequence has no in-frame stop codons.

Because our system assumes that all first exons start with ATG and all last exons end with stop, it would be more accurate to call it a "coding region" system. We use the phrase "gene-finding" because it seems to be the convention adopted by the field.

Splice site selection

The knowledge of splice junction signals (the boundaries between exons and introns) provides very important clues as to where the true gene might be. Guided by the splice site consensus information collected by Mount (Mount *et al.* 1992; Mount, Peng, & Meier 1995), we created a pre-processor that scanned every sequence and scored every location on how well it matched the 5' and 3' consensus sites. We used a 9-base region around the 5' (donor) site and a 15-base region around the 3' (acceptor) site. In order to compute these sites, we computed the probability of each base in each position in our training data. We then stored in a matrix the score $M_{b,i} = \ln(f_{b,i}/p_b)$, where $f_{b,i}$ is the frequency of base b in location i of the site and p_b is the prior probability of base b in the data. Using the same method as Snyder and Stormo (Snyder & Stormo 1995) we scored a site by adding these log ratios: a new site $s_i, s_{i+1}, \ldots, s_j$ is scored by:

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

To pre-process the data, we applied this scoring function to all the true sites and then to a large selection of random sites. We then selected a cutoff score that excluded very few of the true sites (about 3-4%) included only about 1% of the random (false positive) sites.

One drawback of this site statistic is that the addition of log ratios is valid only under the assumption that the bases in a site are independent of one another. This assumption is clearly false, and to improve the statistic we came up with the following new site statistic. For a site with d positions, we compute a $16xd$ matrix instead of a $4xd$ matrix. In the first position, as above, we compute just 4 values, $M_{b,i} = \ln(f_{b,i}/p_b)$. However, in each of the subsequent $d-1$ locations, we compute the conditional probability of base b in location i given base a in $i-1$:

$$M_{b,i|a,i-1} = \ln f_{ab,i-1}/f_{a,i-1}$$

This *conditional probability* matrix produced better discrimination thresholds, in our experiments, than the standard matrix that assumes independence between positions. (The actual matrices for our data will be made available on our Web site.)

GeneID Human Data			
Donor Site		Acceptor Site	
False Negatives	False Positives	False Negatives	False Positives
0/255	1033/63951	0/255	2072/63951
8/255	552/63951	7/255	864/63951
16/255	405/63951	16/255	582/63951
Our Human Data			
Donor Site		Acceptor Site	
False Negatives	False Positives	False Negatives	False Positives
38/1954	2528/58631	41/1954	2658/58631
47/1954	1316/58631	79/1954	1316/58631
87/1954	663/58631	178/1954	663/58631

Table 2: False negative and false positive rates using individual base site statistics.

The improvement that results from using this matrix is illustrated in Tables 2 and 3, which look at both donor sites and acceptor sites. The table compares the individual base scoring method to the conditional probability method on two data sets. Each line of the table uses a different fixed score to decide whether or not a site is genuine. The GeneID data (Burset & Guigo 1996) contains 63 of the 93 human sequences (out of 570 total) from (Burset & Guigo 1996), and this data had more stringent filters applied to it. Our data was less stringently filtered, and as a result contains more of the typical errors that one finds commonly in the sequence databases. Therefore any statistic for identifying splice junctions will have more trouble with this data, as shown in the tables. These tables show the number of false positives and false negatives as a fraction of the total number. The false positives are an estimate, generated by selecting a 10 random sequences from our data, removing all the true sites, and then applying the site statistic. For example, in the second line of Table 2, we set the cutoff score so that 8 out of 255 true donor sites would exceed the cutoff. Using this same score, 552 out of 63951 (0.86%) of the random sites would also exceed the cutoff. We set a different cutoff score for the acceptor site matrix, but at a similar level, where it missed 7 out of 255 true sites, it would include 864 (1.3%) of the false sites.

GeneID Human Data			
Donor Site		Acceptor Site	
False Negatives	False Positives	False Negatives	False Positives
0/255	854/63951	0/255	936/63951
8/255	414/63951	6/255	684/63951
9/255	310/63951	10/255	504/63951
Our Human Data			
Donor Site		Acceptor Site	
False Negatives	False Positives	False Negatives	False Positives
0/1954	1976/58631	0/1954	1355/58631
14/1954	1734/58631	12/1954	1120/58631
31/1954	1174/58631	97/1954	615/58631

Table 3: False negative and false positive rates using conditional probability statistics.

Table 3 shows that the use of a conditional probability score at each donor and acceptor site gives substantially better discrimination power. For example, in the GeneID data, a cutoff that misses 16 true donor sites will falsely include 405 random sites. Using the conditional probability scores, a cutoff that misses just 9 true sites will include only 310 false negatives. Similarly, with our data set, a cutoff using the independent statistic that misses 178 (9.1%) of the true acceptor sites will include 663 false positives, whereas the conditional matrix allows one to set a cutoff that misses just 97 (5%) of the true acceptor sites while including 615 false positives.

One source of the greater errors in our data is simple mis-labeling of the splice junctions themselves. For example, we found one sequence with 15 exons where every single splice site was offset by one, with the consequence that the site statistics, when applied to the putative true site, gave very poor scores. Offsets of one position appear to be common in the annotations, and we are in the process of hand-adjusting all of these for our dataset and notifying GSDB of the errors.

DP algorithm with constraints

Using the splice junction matrices, we re-formulated the DP algorithm so that it only computes new scores at every potential splice site, not at every location. Because the algorithm's running time is quadratic in the number of sites considered, this speeds up the algorithm by a factor of several hundred, allowing it to process in seconds what would have taken hours. We set the thresholds using the tables above so that the system would not miss any true sites (no false negatives) in the GeneID data. The same thresholds were used for our data, where they would result in missing about 5% of true sites. (Note that some of the "true" sites might be mis-labeled in the database, in which case missing them is not harmful.)

Computational complexity. For each extension of the partitioning from k to $k+1$, the DP algorithm considers $2k$ possible choices. If the total sequence length is n, and one call to the scoring function has cost c, then the DP algorithm does $O(cn^2)$ work. For our system, c was the cost of evaluating a sequence with rules or decision trees. In practice c was a very small portion of the overall run time. We can avoid even more computational cost by taking advantage of the following observation. The decision tree algorithm uses essentially constant time to label a sequence, because the depth of the trees is always quite small. The main cost comes from computing the feature values. During the DP algorithm, all these features will be computed for every possible subsequence of the input, i.e., for all $n(n-1)/2$ subsequences. Fortunately, every one of the features used by the decision tree algorithm can be computed incrementally in constant time. In other words, if we already know the feature for the sequence $S_{i,j}$, we can quickly compute the same feature for $S_{i,j+1}$. As a simple example, consider the open reading frame (ORF) feature. If we know the location and length of the longest ORF for $S_{i,j}$, then extending the sequence by one base can only extend the ORF by at most one base, which we can check in constant time. We therefore classify all the subsequences in a preprocessing step, which takes $O(n^2)$ time by doing incremental feature updates. This allows us to run DP in $O(n^2)$ time.

The modified algorithm runs in time proportional to s^2, where s is the number of signals. Since s is usually about 10-20 times smaller than n, the length of the sequence, the modified algorithm runs hundreds of times faster than an algorithm that computes a score at every location in the sequence.

Classification rules and trees

The first experiments described below show that decision tree classifiers can distinguish between exon and introns using content measures alone with high accuracy, over 94%.

Methods

Decision tree classifier accuracy

We first divided our set of 395 complete coding sequences into a training and test set. 100 sequences were chosen at random and set aside to be the test set; these were not used in any of the trials during which we tuned our algorithms. Based on the experience of previous gene-finders (Burset & Guigo 1996), we divided the 295 training sequences into those with high, medium, and low GC-content. We defined "medium" as within one standard deviation of the mean value for all the sequences. In our data, the mean GC content was 49%, with a standard deviation of 11%. We then computed 10 trees for each of these subsets.

Data set	Num exons	Num introns	Exon acc.	Intron acc.
High GC	634	706	96.9	94.2
Med GC	1553	1727	93.3	93.9
Low GC	377	415	94.2	97.1
Overall	2564	2848	94.3	94.5

Table 4: Percent accuracy of decision trees on the exon-intron classification problem.

As a preliminary experiment, we separated the sequences into a set of coding regions (whole exons) and a set of noncoding regions (whole introns and additional intergenic DNA from the 5' and 3' ends of some sequences). We used the GSDB annotations to define our exon and intron boundaries, although it is important to note here that many of these annotations artificially place the first exon beginning at the start (ATG) set, and end the last exon at a stop codon, even if the true exons ran past these boundaries. On this 2-class problem, we ran a 5-fold cross validation study[2] to check the accuracy of the method on isolated subsequences. For this simpler problem, the classification accuracy of the combined tree systems on the training data was very high, over **94%**, as shown in Table 4.

Note that this 94% figure was obtained without using any signal information: the decision trees classify by using a set of features computed for each subsequence, and the features used here intentionally excluded the presence of start codons and the 5' and 3' splice junction signals. (Roughly 99% of introns begin with *GT* and end with *AG*.) The reason we ignored this important signal information here was that this information is used by the dynamic programming algorithm to pick out candidate exons and introns. Therefore any time the decision tree system is called to label a subsequence, that sequence *already* has the appropriate signals. This test shows that the decision tree can distinguish these sequence types very accurately by content alone.

Dynamic programming system accuracy

The classification problem of the prevous section is actually not the problem that the dynamic program needs solved. At each site in the input sequence, the dynamic programming is not asking for a decision about exon vs. intron. Rather, it is asking questions such as this: given that a subsequence $s_{i,j}$ has a valid start site at location i and a donor site at location j, and that $s_{i,j}$ is an open reading frame (ORF), is $s_{i,j}$ an initial exon? So what is really happening is that the system is comparing the true exon to many alternatives, which we call pseudoexons. Pseudoexons have valid ORFs and strong signals at either end, but are not true exons. Likewise when the system is processing introns, it is comparing true introns to pseudointrons.

To address this problem, we crafted four *different* classifiers, to make the following comparisons:

1. initial exons vs. initial pseudoexons.
2. internal exons vs. pseudoexons.
3. introns vs. pseudointrons, and
4. final exons vs. pseudoexons.

Each of these classifiers was developed and trained using data output by a version of the dynamic programming algorithm itself: this version does not parse the input, but instead just outputs all the alternative subsequences as it considers them. In this way we built up four different training sets for these classifiers.

The decision tree classifiers were most effective for the initial and final exons, and were developed as described above. For the initial exons, the decision tree classifiers obtain an average accuracy of 83.3% on a sample extracted from the training data. For the final exons, the classifiers obtain 89.2% accuracy on average. Note that although these numbers are quite high, the dynamic program considers a very large number of pseudo-sites, so even higher accuracies would be desirable.

For the internal exons and introns, hand-crafted rules worked as well as the best decision trees. For the introns, rules were developed using the splice junction scores described above, and the decision tree could not beat their performance. The reason behind this is as follows: suppose the system is comparing a true intron $s_{i,j}$ to a pseudointron $s_{i,k}$, and suppose the two subsequences have similar lengths. Since they have the same starting location, almost any content statistics (such as hexamer usage) will be virtually identical; the only real way to distinguish them is using the sites at each end. The conditional probability matrix scores around those sites proved to be very effective for this task.

For internal exons, the content information was more useful, but here the best hand-crafted rule set used a combination of the site scores and the in-frame hexamer statistic of Snyder and Stormo (1995). These rules were discovered by the decision tree program, in fact, but they were simple enough to encode directly as rules.

Results of using the hybrid decision tree/dynamic programming system for the full gene-finding task are given in Tables 5 through 6. The system is called DynaGene, in reference to its underlying dynamic programming formulation. The tables follow the model for reporting used by Burset and Guigo (Burset & Guigo 1996). Note that these results should be considered preliminary, as the system is still in its initial development stages. We did not yet divide the data by GC content for these experiments, which should improve

[2] A 5-fold cross validation takes a training set and breaks it randomly into five roughly equal size subsets. For each subset, the algorithm is training on the remaining data and tested on the subset not used for training. The accuracy is then summed over all five partitions.

Data	Size	TrCDS	PrCDS	TP	TN	TEx	PEx	OvEx	TPE	1MEx
Train	2147903	474241	665805	305529	1310386	2230	4317	1888	665	1406
Test	716226	146473	118124	70086	521715	662	700	432	164	298

Table 5: Total base counts and exon prediction accuracies for finding genes using DynaGene.

the performance. Nonetheless, we consider the preliminary results promising. In Table 5, TrCDS is the sum

Data Set	Sn	Sp	CC	P(I)	P(All)
Train	0.64	0.45	0.38	0.78	0.75
Test	0.48	0.59	0.43	0.92	0.83

Table 6: Overall accuracy of gene parsing using DynaGene.

of the true CDS lengths, PrCDS is the sum of the predicted CDS lengths, TP is the total length (in base pairs) of correctly predicted coding regions, TN is the total length (in base pairs) of correctly predicted noncoding (intron or intergenic) regions, TEx is the total number of true exons, PEx is the total number of predicted exons, OvEx is the number of true exons that are overlapped by a predicted exon, TPE is the number of perfectly predicted exons (both boundaries are correct), and 1MEx is the number of exons for which at least one edge (either the 5' or 3' end) is predicted correctly. In Table 6, Sn is the sensitivity: the percentage of true exon bases that were correctly predicted. Sp is the specificity of the parse: the number of predicted exon bases that were truly exon, CC is the correlation coefficient. P(I) is the probability that if a given base is truly an intron we will mark it correctly, and P(All) is the probability that the system mark any base correctly.

The results given in the tables show accuracies for both the 295 training sequences and the 100 test sequences. Note that some re-adjustment of the system occurred before obtaining the final results on the training data, but the system was only run once on the test data. The accuracies for the training data vary depending on system parameters; for example, we can increase the CC from 0.38 to 0.42 by adjusting a parameter that will make the system predict fewer exons; it will then get only 567 (instead of 665) exons exactly right, but the increase in specificity will cause the CC to rise. All of these accuracies are based on a definition of "truth" defined by the GSDB entries; of course, some of those annotations and sequences have errors, and the real accuracies may as a result be somewhat higher. Another explanation for the low accuracy on some sequences is the presence of unusual start, donor, and acceptor sites. We ran DynaGene on the smaller GeneID data set as well, and there obtained much higher accuracies: a CC of 0.50 and exact exon prediction on 35% of the exons.

A detailed example

To illustrate the system's performance more clearly, we consider one fairly complex gene from the test set, HUMALPPD (human placental heat-stable alkaline phosphotase, accession M19159. This complex gene entry is 4268 nucleotides long, containing 1608 coding bases scattered across 11 exons. The DynaGene system parsed this sequence into 10 exons, of which 8 exactly matched the true exons, and one other overlapped a true exon This does not represent the system's best performance (on some smaller genes it found exactly the right parse) or its worst, but it illustrates the ability of the system to do well on relatively complex genes. The comparison between the system's answer and the true gene structure is:

True Exon	Predicted	Comment
54 129	54 129	correct
222 338	222 338	correct
451 566	451 566	correct
764 938	764 938	correct
1015 1187		false negative
1438 1572	1438 1572	correct
1672 1744	1672 1744	correct
1875 2009	1875 2009	correct
2092 2283	2092 2305	1 matching edge
2503 2619	2503 2619	correct
2749 3047		false negative
	3645 3840	false positive

Note that the only errors made by DynaGene here consist of missing some of the exon boundaries. The system's performance on the 14 known exons is very accurate; in all but one case, at least one end was located correctly. On the basecount level, this parse is 84% correct, and on the more difficult measure of percentage of exons found exactly, it gets 8/11. This is especially encouraging given the difficulty of correctly discovering the structure of such a complex multi-exon gene.

One other example is worth mentioning. Our data includes HUMCOL7A1X (accession L23982), the collagen type VII intergenic region and (COL7A1) gene, which has 118 exons. For this sequence, DynaGene predicted 122 exons, of which 38 were exactly correct, 91 had at least one edge correct, and 105 were overlapping. Note that the correlation coefficient for this one sequence was 0.43, identical to the average performance of the system on the test data. But the extraordinary complexity of this gene makes even this level of performance difficult to obtain.

Conclusions and next steps

The results of this new gene-finding system show encouraging performance on a difficult set of data. Overall, we obtained basepair accuracies of 83%, and the sensitivity and specificity for exons were only 48% and 59%. The hardest test of any system is how often it found exactly the right exon with both the 5' and 3' ends correct; no system currently does well at this test, and ours is no exception, finding only about 1 in 4 exons exactly. However, we find an exon overlapping the true exons 2/3 of the time. Thus our system seems to be accurate at picking out the exonic regions of an uncharacterized DNA sequence. It also finds at least one correct edge (splice junction) for half of the exons in our test data.

It should be emphasized again that these are preliminary results, and the system is still actively under development. In addition, these numbers by no means give the whole picture: the system showed a tendency to do very well on a large fraction of the data, but to completely miss the correct answer on a small percentage. These misses, which clearly need further analysis, substantially bring down the overall numbers.

The introduction of a new site statistic, which computes conditional probabilities between successive bases at donor and acceptor sites, is an important development that should help improve all gene finding systems. It is being used in our system both as a thresholding mechanism to improve efficiency, and as part of the scoring algorithm that classifies the different types of subsequences. Because similar statistics are used in many systems (and, as noted above, the independent site statistic is due to (Snyder & Stormo 1995) in the GeneParser system), this should help improve the accuracy of other systems as well.

An important advantage of using decision trees and rules (in contrast to neural networks, for example) is the ability that it provides to analyze the errors made by the system. This is an important topic for future work, and in particular we are currently examining the errors and separating them into different types: misidentification of initial exons, internal exons, introns, and final exons. Each of these sequence types has different properties, and an understanding of what makes it difficult to correctly identify any one of them should help improve all gene-finding systems.

This is the first test of the new system, and these results should be taken as quite preliminary and likely to improve substantially. It already produces good results on human DNA, and one of our immediate short-term plans is to run the system on Burset and Guigo's vertebrate DNA database (Burset & Guigo 1996), which contains 570 genes from a wide range of vertebrate organisms. We will then compare our results directly to many other standard gene-finding systems that have also been run on this data. We also plan to make our database available so others can use it for their own experiments. We also plan to incorporate database lookup information in the system, which is also known to produce substantial improvements (Snyder & Stormo 1995). Finally, we have begun design of an approach that uses decision trees as part of a preprocessing step to label likely exonic regions, which may lead to more robust performance overall.

Acknowledgements

Thanks to Simon Kasif for many helpful suggestions. This material is based upon work supported by the National Science foundation under Grant Nos. IRI-9116843 and IRI-9223591, and by a Young Faculty Research Initiative grant from the G.W.C. Whiting School of Engineering at Johns Hopkins University.

References

Breiman, L.; Friedman, J.; Olshen, R.; and Stone, C. 1984. *Classification and Regression Trees.* Wadsworth Internatl. Group.

Burset, M., and Guigo, R. 1996. Evaluation of gene structure prediction programs. *Genomics.* accepted.

Cinkosky, M.; Fickett, J.; and Keen, G. 1995. A new design for the genome sequence data base. *IEEE Engineering in Medicine and Biology.*

Fickett, J., and Tung, C.-S. 1992. Assessment of protein coding measures. *Nucleic Acids Research* 20(24):6441–6450.

Fulton, T.; Kasif, S.; and Salzberg, S. 1995. Efficient algorithms for finding multi-way splits for decision trees. In *Proc. of the Twelfth Internatl. Conf. on Machine Learning*, 21–29. San Mateo, CA: Morgan Kaufmann.

Guigo, R.; Knudsen, S.; Drake, N.; and Smith, T. 1992. Prediction of gene structure. *J. Mol. Biol.* 226:141–157.

Henderson, J.; Salzberg, S.; and Fasman, K. 1996. Hidden Markov Models for finding genes in human DNA. Technical Report 1996-02, Dept. of Computer Science, Johns Hopkins University.

Hood, L.; Koop, B.; Rowen, L.; and Wang, K. 1993. Human and mouse t-cell-receptor loci: the importance of comparative large-scale DNA sequence analyses. *Cold Spring Harb. Symp. Quant. Biol.* 58:339–348.

Kristensen, T.; Lopez, R.; and Prydz, H. 1992. An estimate of the sequencing error frequency in the DNA sequence databases. *DNA Seq.* 2(6):343–346.

Mount, S.; Burks, C.; Hertz, G.; Stormo, G.; White, O.; and Fields, C. 1992. Splicing signals in *drosophila*: intron size, information content, and consensus sequences. *Nucleic Acids Research* 20:4255–4262.

Mount, S.; Peng, X.; and Meier, E. 1995. Some nasty little facts to bear in mind when predicting splice sites. In *Gene-Finding and Gene Structure Prediction Workshop.*

Murthy, S. K.; Kasif, S.; and Salzberg, S. 1994. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research* 2:1–33.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers.

Salzberg, S. 1995. Locating protein coding regions in human dna using a decision tree algorithm. *Journal of Computational Biology* 2(3):473–485.

Sankoff, D. 1992. Efficient optimal decomposition of a sequence into disjoint regions, each matched to some template in an inventory. *Mathematical Biosciences* 111:279–293.

Snyder, E. E., and Stormo, G. D. 1993. Identification of coding regions in genomic DNA sequences: An application of dynamic programming and neural networks. *Nucleic Acids Research* 21(3):607–613.

Snyder, E. E., and Stormo, G. D. 1995. Identification of coding regions in genomic DNA. *Journal of Molecular Biology* 248:1–18.

Solovyev, V.; Salamov, A.; and Lawrence, C. 1994. Predicting internal exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames. *Nucleic Acids Res.* 22:5156–5163.

Waterman, M.; Uberbacher, E.; Spengler, S.; Smith, F.; Slezak, T.; Robbins, R.; Marr, T.; Kingsbury, D.; Gilna, P.; Fields, C.; Fasman, K.; Davison, D.; Cinkosky, M.; Cartwright, P.; Branscomb, E.; and Berman, H. 1994. Genome informatics I: Community databases, report of the invitational DOE workshop on genome informatics. *J. Computational Biology* 1:173–190.

White, O.; Dunning, T.; Sutton, G.; Adams, M.; Venter, J.; and Fields, C. 1993. A quality control algorithm for DNA sequencing projects. *Nucleic Acids Res* 21(16):3829–3838.

Wu, T. 1995. A phase-specific dynamic programming algorithm for parsing gene structure. Gene-Finding and Gene Structure Prediction Workshop.

Xu, Y.; Mural, R.; and Uberbacher, E. 1994. Constructing gene models from accurately predicted exons: an application of dynamic programming. *CABIOS* 10(6):613–623.