Appears in the *Proceedings of the 1991 DARPA Workshop on Case-Based Reasoning*, Morgan-Kaufmann.

Finding Genes by Case-Based Reasoning in the Presence of Noisy Case Boundaries *

Jude W. Shavlik
Computer Sciences Department
University of Wisconsin
Madison, WI 53706 USA
shavlik@cs.wisc.edu

Abstract

Effectively using previous cases requires that a reasoner first match, in some fashion, the current problem against a large library of stored cases. One largely unaddressed task in case-based reasoning is the process of parsing continuous input into discrete cases. If this parsing is not done accurately, the relevant previous cases may not be found and the advantages of case-based problem solving will be lost. Parsing the data into cases is further complicated when the input data is noisy. This paper presents an approach to applying the case-based paradigm in the presence of noisy case boundaries. The approach has been fully implemented and applied in the domain of molecular biology; specifically, a successful case-based approach to gene finding is described. An empirical study demonstrates that the method is robust even with high error rates. This system is being used in conjunction with a Human Genome project in the Wisconsin Department of Genetics that is sequencing the DNA of the bacterium *E. coli*.

1. Introduction

Given a new problem to solve, the initial step a case-based reasoner takes is to recall relevant cases from its memory. Since it is unlikely the current situation is completely identical to a previous one, some sort of *partial matching* is needed. Often this involves selecting some cues and using these to index into memory [.kolodner book.]. However, partial matching presupposes that the case-based reasoner is somehow given a well-defined current situation — one where the "boundaries" of the current case are cleanly defined. Unfortunately, the real-world is often "continuous", and the problem of accurately parsing experience into discrete cases is extremely challenging. This largely unaddressed task (one exception is [.redmond creating cases from examples.]) is made even more complicated when one's sensors for measuring the world are noisy. Even if one had a good idea of what constitutes the boundaries of a case, noise may easily blur these signals.

This paper presents a method for performing case-based reasoning in the presence of noisy case boundaries. The task domain is molecular biology, and we have successfully used our technique to find genes in noisy DNA sequences. The following sections provide a brief introduction to molecular biology and describe our algorithm. For now note that genes are subsequences in a lengthy string. Due to the nature of the genetic code, certain types of noise in this string cause current partial-matching algorithms for gene finding to fail.

---

* This research was partially supported by Office of Naval Research Grant N00014-90-J-1941, National Science Foundation Grant IRI-9002413, and Department of Energy Grant DE-FG02-91ER61129.

Gene finding is a task highly amenable to AI solutions: there are rapidly-growing computer databases (described below), most of the information is "discrete" (as opposed to involving, say, partial differential equations as do most other forms of computational science), and much of the domain-specific knowledge is heuristic. Several basic approaches are being investigated within the AI and related communities:

(1) Development of algorithms that perform similarity matches to known genes (i.e., case-based reasoning) [.blast{, this paper}, fasta .].

(2) Using machine learning to learn the general characteristics of genes [.shavlik aaai90, noordewier nips3, stormo perceptron, Lapedes Sirotkin.].

(3) Creating grammars that can be used to recognize genes [.Searls Investigating the Linguistics of DNA with Definite Clause Grammars Logic Programming, searls aaai88.].

This remainder of this article presents an approach that falls into the first category.

2. A Brief Introduction to Molecular Biology

Since this paper describes an AI application in molecular biology, a brief introduction to genes and proteins appears in this section. Sufficient detail is included so that the non-biologist can understand the rest of the paper; further information can be found in textbooks such as [.watson full book.].

DNA is a linear sequence from the alphabet {*A, G, T, C*}; each of these four letters is called a *nucleotide* (or *base*). Human DNA is estimated to contain 3×10^9 nucleotides, while the common intestinal bacterium *E. coli* contains about 5×10^6 bases. A DNA molecule usually involves two, complementary sequences, organized as a double helix. An *A* in one sequence is paired with a *T* on the other; *G* and *C* are similarly paired. This pairing forms the basis of cell *replication*, and its discovery by Watson and Crick in 1953 revolutionized biology [.watson crick.]. However, replication is not addressed in this paper, and the reader can think of DNA as a single, linear sequence (whose complement — or *reverse* strand — can easily be calculated when needed).

The most important aspect of DNA, for the purposes of this paper, is that subsequences in it encode *proteins*; these subsequences are called *genes*. Proteins are the "work horses" of the cell; for example, enzymes are proteins, as are cell membranes. Proteins, too, are linear sequences. They come from a 20-character alphabet; each of these letters is called an *amino acid*. In a process called *translation*, a gene is read and a protein produced.[1] Each consecutive three-letter string in a gene encodes one of the 20 amino acids — this mapping from nucleotide triplets to single amino acids is called the *genetic code*.[2] Each three-letter string is called a *codon*. Three are called STOP codons, because they tell the cell to stop translating the DNA. Table 1 contains a hypothetical gene and shows the protein it would encode; bars (|'s) are used to group the bases into codons. (This would be too short to be a real gene — most genes are several hundred nucleotides long.)

---

[1] There actually is an intermediate step called *transcription*, where DNA is copied into a similar molecule called RNA. This RNA is what is then translated into proteins. However, this level of detail is not necessary to understand this paper; the reader can assume DNA directly maps to protein.

[2] Since there are $4^3 = 64$ distinct three-letter DNA strings, the genetic code is redundant.

Since the translation process involves grouping nucleotides into threes, the *reading frame* is of extreme importance. There are three possible reading frames — one where nucleotide *i* is the first item in a codon, another where it is the second, and one more where it is the third. We will return to the topic of reading frames later — they play a major role in this paper because they determine case boundaries.

The Human Genome Project [.Alberts National Research Council.] is a massive, world-wide research project that has the goal of determining the sequence of human DNA and locating all the genes within it. The genomes of several other scientifically important species are also being sought. A genetics project at the University of Wisconsin is sequencing the DNA of the common bacterium *E. coli*, and the research reported in this paper is a result of collaboration with that project, headed by Prof. F. Blattner of the genetics department. Blattner's group has chopped up *E. coli* DNA into roughly 500 pieces. Each of these pieces is called a *contig* (for *contig*uous sequence), and we are computationally analyzing each contig as it is sequenced by Blattner's laboratory.

Unfortunately, there is no absolute START codon; this makes gene finding a non-trivial task. (See [.shavlik aaai90.] for a discussion on applying machine learning techniques to find a signal that indicates the start of a gene.) The remainder of this paper describes a case-based approach to gene finding, one that works in the presence of "noisy" DNA sequences. Sequencing is estimated to have an error rate of 1% [.Alberts National Research Council.]; the wrong nucleotide can be recorded or, more disastrously, an extra nucleotide can be inserted or an existing one can be missed. As further explained later, these insertions and deletions greatly affect the translation process, due to the triplet nature of the genetic code.

3. Finding Genes in Noisy Data

In any project to map and sequence an organism's genome, the interpretation of the final sequence is an undertaking of great magnitude. The inherent potential for errors in the recorded sequence further complicates such analyses. The Wisconsin *E. coli* sequencing project is producing large amounts of "anonymous" DNA; we are computationally analyzing this data. There are two closely-related, main goals of this research. First, we wish to correct sequencing errors by noting inconsistencies with other biological data. Second, we wish to locate and identify those regions of the sequence that encode proteins — both known and heretofore unknown.

We are undertaking two complementary approaches. One involves performing robust similarity matches with known protein sequences (this paper), while the second involves detecting DNA segments that have the general characteristics of genes (see [.shavlik aaai90.]). Our long-term aim is to assign a function to all regions of the "anonymous" DNA produced by the Wisconsin sequencing project.

Table 1. A hypothetical gene and its translation into a protein.

Gene:	|AGC|ATG|CAA|TAG|
Protein:	| S | M | Q | *STOP*

Our primary concern, with respect to error correction, is to locate *frameshift errors* (the mistaken insertion or deletion of a nucleotide, which can cause a gene to "shift" into an improper reading frame). Due to the triplet nature of the genetic code, such errors can be disastrous if they occur inside a putative gene. Once the translation process is "out of frame", the remainder of a predicted protein bears no resemblance to the correct protein and partial matching will fail. The computational methods this paper presents are designed to locate genes and be robust in the presence of frameshift errors.

It is important that the reader understand that frameshift errors greatly affect what protein is predicted by translating a gene. Perhaps this can best be seen by considering the process of translating from a string of bits to alphabetic characters, using the ASCII code. If a bit is dropped or inserted, the resulting translation will bear little resemblance to the correct text.

There are several international databases that store biological sequence data, most notably Genbank [.genbank main ref.] and Protein Information Resource (PIR) [.PIR main ref.]. These databases store "cases": complete genes (GenBank) and complete proteins (PIR). A number of researchers have developed "case-based" algorithms that partially match DNA subsequences (or the corresponding amino-acid sequences) to these databases (e.g., [.blast, fasta.]); however, their methods all suffer from being extremely sensitive to frameshift errors.

These previous approaches do an excellent job of matching in the presence of *substitution* errors — mistaking an *A* for a *C*, say. In fact, one of their primary strengths is that they can find *homologous* matches. A homologous protein is one from another species that is similar in terms of its amino-acid sequence (and biological functionality); due to the process of evolution, homologous proteins abound and locating them is of major importance. When genes are sequenced on a case-by-case basis, frameshift errors are much less frequent and the primary biological task is to find similar matches to known proteins. The previous algorithms were designed for this problem and have been rather successful. However, with the advent of massive sequencing projects, noisy DNA with unknown functionality and unknown codon boundaries is rapidly being produced. For these conditions, a new case-matching algorithm is needed.

3.1. A Case-Based Gene Finding Algorithm

We have developed a case-based algorithm for gene finding that is robust in the presence of frameshift errors. Our FIND-IT algorithm builds on the BLAST similarity-search program [.blast.]. BLAST efficiently produces approximate matches, but these matches do not extend across frameshift errors. The FIND-IT method described below coherently combines partial matches (to a given protein) in different reading frames, thereby overcoming missing and extra nucleotides in sequenced DNA.

Table 2 describes our algorithm. Given a sequence of DNA, the algorithm collects all open-reading frames (ORFs)[3] in the sequence and its complement. Next, the algorithm translates these ORFs to proteins and uses a variation of BLAST to match each protein against the PIR database. We convert to proteins, rather than directly matching DNA to GenBank entries, because only *partial* matches to amino acids are biologically well-defined; BLAST uses the PAM 120 matrix [.dayhoff 1978.] to define the similarity between two amino acids. Also, due to the genetic code's degeneracy and the different codon-usage patterns of various species, vastly

[3] An ORF is the DNA between two successive STOP codons; not all ORFs contain genes, but, if there are no sequencing errors, all genes lie within some ORF.

Table 2. The FIND-IT algorithm for matching known proteins to new DNA sequences in the presence of frameshift errors.

Given a sequence of DNA:

(1) Collect all of the open-reading frames (ORFs), in each of the three possible reading frames on the DNA and on its complementary strand (for a total of six frames). No minimal length is required to be considered an ORF; because sequencing errors may have introduced false STOP codons, no stretch of DNA is discarded.

(2) Translate each ORF into a amino-acid sequence and apply BLAST [.blast.] to produce partial matches (with gaps) to the Protein Information Resource (PIR) database. Note that *all* of the DNA in a sequence, in all six frames, is translated and matched.

(3) Collect all of the matches to each PIR protein encountered.

(4) By piecing together matches, look for consistent *coverings* of each protein (see text for details).

(5) Score the combined protein matches, sort, and report the best matches.

different DNA sequences may lead to quite similar protein sequences. Finally, matches to GenBank can be easily effected by translating the genes in that database to proteins, and then applying Table 2's algorithm. In summary, translating sequenced DNA to proteins and matching to the PIR protein database requires one to address the problem of frameshift errors, but the advantage is that partial matches are better defined biologically.

A useful feature of BLAST is that a protein need not match in its entirety; rather, it reports matching *sub*sequences. A match may terminate within a protein due to a frameshift error; in this case the remainder of the protein will match another ORF (examples of this follow).

A match returned by BLAST maps a portion of the DNA sequence to a segment of a protein. Figure 1 schematically shows three matches; the numbers following the matches indicate their reading frame. Note that matches A and C provide a consistent "covering" of the protein, while match B is inconsistent with the other two. Also note that by combining matches A and C, an extra nucleotide in the DNA can be identified (the one marked with an X). Below we define what it means to be a *consistent* collection of matches to a given protein (which we call a "covering").

Assume match i maps the DNA segment $[A, B]$ to the protein segment $[P, Q]$, while match j maps the DNA segment $[C, D]$ to the protein segment $[R, S]$. If matches i and j both belong to a consistent covering, then the following constraints hold:

(1) If $C > A$ then $R > P$. That is, the left-to-right order on the DNA is the same as that on the protein sequence. Also, they must be on the same DNA strand (i.e., forward or reverse).

(2) If matches i and j intersect on one sequence (DNA or protein), they also intersect in the corresponding locations on the other. (Since there may be extra or missing nucleotides in the DNA sequence, "small" discrepancies are ignored.)

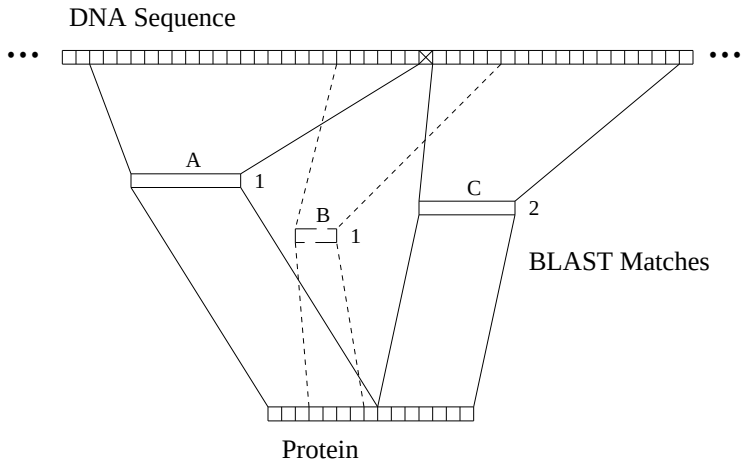

Figure 1. Combining BLAST matches to overcome frameshift errors.

(3) This distance between DNA locations *B* and *C* is approximately three times the distance between protein locations *Q* and *R*. That is, the amount of DNA between the end of one match and the beginning of the next roughly corresponds to the number of amino acids between the two matches. (This constraint can be relaxed if FIND-IT is applied to eukaryotic DNA from advanced species, where genes are not necessarily contiguous pieces of DNA. Hence, the approach elegantly extends to the recognition of "exons" among intervening sequences.)

In our current experience, FIND-IT generally looks for consistent coverings for a given protein from a set of about 100 matches (a given DNA contig partially matches several thousand different proteins); successful coverings usually contain from one to five matches. There are on the order of 2^{100} possible combinations when given 100 matches, but we are able to use the constraints on a consistent covering to greatly prune the number considered. In addition, we require that at least half the protein sequence be matched; the protein coverage of a collection of *N* matches can be determined in *O(N log N)* time, and many proteins can be discarded before performing the inefficient step of finding consistent coverings. However, our program still spends a considerable amount of time searching for consistent covers. One of the major open issues is to understand the computational complexity of this task and devise efficient algorithms for it; a later section further discusses this topic.

3.2. Sample Results from Two *E. coli* Contigs

We have applied our algorithm to two *E. coli* contigs already sequenced: EC17-115 and EC21-76. On the first of these, 14 matches to *E. coli* proteins were found, and 27 "strong" matches to proteins of other species were found. (A "strong" match is defined to be one where at least half the protein sequence is matched. When several homologous proteins match the same stretch of DNA, FIND-IT retains the strongest-matching one.) Overall, these matches accounted for 47.6% of the contig's DNA. On contig EC21-76, four *E. coli* proteins were encountered and 32 homologous matches were detected; 36.4% of this contig was covered. In our current research, we are trying to increase these numbers.

Figure 2 contains all those matches to contig EC17-115 that were combined into one of the 41 protein coverings; each arrow-headed segment represents one match produced by BLAST.

We have also tested FIND-IT on the DNA sequence of the completely-sequenced bacteriophage called λ; all of the λ proteins in PIR were found, as well as several matches to non-λ proteins. Some, but not all, of these homologies are noted in the PIR annotation; these previously-unknown homologies are of substantial biological interest.

3.3. Using Multiple Matches to Detect Sequencing Errors

This section presents two actual composite matches produced by FIND-IT. One (Figure 3) involves matching an *E. coli* protein, while the other (Figure 4) is a match to a human protein. These matches illustrate how sequencing errors can be detected and show that both known *E. coli* and homologous genes can be located.

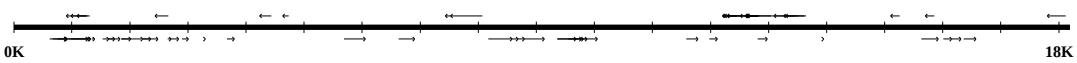

Figure 2. Consistent matches to contig EC17-115.

Protein: PAECS (Phosphoserine phosphatase - Escherichia coli #EC-number 3.1.3.3 | 286.0)
Protein Length=322; Protein Coverage=100.00%; Forward Strand

```
DNA[691]:   8176 [MPNI]TWCDLPEDVSLWPGLPLSLSGDEVMPLDYHAGRSGWLLYGRGLDKQRLTQYQSK  8349
 frame=1         [MPNI]TWCDLPEDVSLWPGLPLSLSGDEVMPLDYHAGRSGWLLYGRGLDKQRLTQYQSK
Protein:       1 [MPNI]TWCDLPEDVSLWPGLPLSLSGDEVMPLDYHAGRSGWLLYGRGLDKQRLTQYQSK    58

            8350 LGAAMVIVAAWCVEDYQVIRLAGSLTARATRLAHEAHL-MSPRWKIPHLRTPGLLVMDMD  8526
                 LGAAMVIVAAWCVEDYQVIRLAGSLTARATRLAHEA+L-++P  KIPHLRTPGLLVMDMD
              59 LGAAMVIVAAWCVEDYQVIRLAGSLTARATRLAHEAQLDVAPLGKIPHLRTPGLLVMDMD   118

            8527 STAIQIECIDEIAKLAGTA                                         8583
                 STAIQIECIDEIAKLAGT+
             119 STAIQIECIDEIAKLAGTG                                          137

DNA[163]:   8568 tgr[ngEM]VAEVTERAMRGELDFTASLRTRVATLKGADA-IF                  8687
 frame=3         +g-[.gEM]VAEVTERAMRGELDFTASLR+RVATLKGADA-I.
Protein:     134 ag-[tgEM]VAEVTERAMRGELDFTASLRSRVATLKGADANIL                  173

DNA[182]:   8681 [NILQ]QVRENLPLMPGLTQLVLKLETLGWKVAIAPGAL                     8791
 frame=2         [NILQ]QVRENLPLMPGLTQLVLKLETLGWKVAIA+G+.
Protein:     171 [NILQ]QVRENLPLMPGLTQLVLKLETLGWKVAIASGGF                      207

DNA[603]:   8779 S[AGFT]FFAEYLRDKVRLTAVVANELEIMDGKFTGNVIGDIVDAQYKAKTLTRLAQEYE  8952
 frame=1         S[+GFT]FFAEYLRDK+RLTAVVANELEIMDGKFTGNVIGDIVDAQYKAKTLTRLAQEYE
Protein:     204 S[GGFT]FFAEYLRDKLRLTAVVANELEIMDGKFTGNVIGDIVDAQYKAKTLTRLAQEYE   261

            8953 IPLAQTVAIGDGANDLPMIKAAGLGIAYHAKPKVNEKAEVTIRHADLMGVFCILSGSLNQK 9135
                 IPLAQTVAIGDGANDLPMIKAAGLGIAYHAKPKVNEKAEVTIRHADLMGVFCILSGSLNQK
             262 IPLAQTVAIGDGANDLPMIKAAGLGIAYHAKPKVNEKAEVTIRHADLMGVFCILSGSLNQK  322
```

Figure 3. Sample match to an *E. coli* gene (in contig EC17-115).

Figure 3's covering of the entire protein sequence of *phosphoserine phosphatase* involves four BLAST matches. Each match begins with `DNA[#]`, reports the frame it appears in, and contains three protein sequences; # is the matching score reported by BLAST and represents the sum of the PAM 120 matrix scores for the aligned amino acids. The top line in a match is the translated DNA, while the bottom is the protein sequence. Numbers at the end of lines represent nucleotide positions in the entire DNA contig and amino-acid positions in the protein, respectively. The middle line presents the *alignment*; letters represent identical matches, +'s represent positive-scoring partial matches, periods represent matches that score zero, and blanks represent negative-scoring matches. Dashes (-'s) represent gaps[4] introduced by BLAST during its matching process. The reason for lower-case letters appears below. Finally, braces ([]'s) indicate the "seed" match in the BLAST algorithm (see [.blast.] for details).

Note that in addition to characterizing a portion of a contig, a covering suggests frameshift errors. In Figure 3, there were three frame transitions; careful inspection of the boundaries between successive matches leads to the prediction of missing or extra nucleotides. Also, other sequencing errors and ambiguities can be located by noting the discrepancies between the protein sequence and the translated DNA. Finally, gaps indicate nucleotide insertions and deletions whose length is a multiple of three. These hypotheses can then be checked by reviewing the original sequencing gels in the genetics laboratory; sequencing errors or database (PIR or GenBank) errors can be corrected as appropriate.

To illustrate the error-correction process, consider the top two matches in Figure 3. The last four protein elements in the first match are repeated in the second. The matches at the end of the first match are stronger, so we can discard the first five matches (due to the insertion of the gap) in the second match. These deleted matches appear in lower case in the figure. Since five amino acids are dropped from the second match, its new beginning on the DNA sequence is $8568+15=8583$. Note that this is the last nucleotide in the first match. Hence, one nucleotide is used twice — as the last item in one codon and the first item in the next. Clearly, the sequencing process missed one nucleotide. Inspection of the genetic code (the map between codons and amino acids) shows that an *A* nucleotide needs to be inserted after position 8583, which will shift the second match to the first reading frame in accordance with the first match's reading frame. Similar analyses can be applied to the other matches in the figure.

[4] Allowing gaps permits, for example, the sequences SQLL and SQMLL to match; the alignment is SQ_LL. Some amino acids in a protein may have only an insignificant function and during evolution these amino acids may disappear without effect; matching with gaps accommodates this phenomenon.

Protein: S04092 (Acetyl-CoA acyltransferase precursor - Human #EC-number 2.3.1.16)
Protein Length=424; Protein Coverage=74.29%; Forward Strand

```
DNA[109]:   6199 SAPLDDIYWGCVQQTLEQGFN-I-ARNAALLAEVPHSVPAV[TVNR]LCGSSMQALHDAA  6366
 frame=1         +..L.DI---CV ..L+.G .-I-AR A .L+++P +VP  [TVNR] C+S++QA+ . A
Protein:      80 PEQLGDI---CVGNVLQPGAGAIMARIAQFLSDIPETVPLS[TVNR]QCSSGLQAVASIA   134

            6367 RMIMTG--D-AQACLVGGVEHMG                                      6426
                   I .G--D-+ AC---GVE M+
             135 GGIRNGSYDIGMAC---GVESMS                                       154

DNA[486]:   6498 [MGLT]AEMLARMHGISREMQDAFA-GAHARAWAATQSAA-FKNEIIP--TGGHDPDGVL  6659
 frame=3         [MG+T]+E +A   GISRE.QD+FA-+++ +A-A +QS +-F..EI+P--T  HD  G.
Protein:     181 [MGIT]SENVAERFGISREKQDTFALASQQKA-ARAQSKGCFQAEIVPVTTTVHDDKGTK   237

            6660 KQFNY--DEVIRPETTVEALATLRPAFDPVNGMVTAGTSSALSDGAAAMLVMSESRAHEL  6833
                 + .. --DE IRP TT+E+LA L+PAF  -+G .TAG.SS +SDGAAA+L+   S+A EL
             238 RSITVTQDEGIRPSTTMEGLAKLKPAFKK-DGSTTAGNSSQVSDGAAAILLARRSKAEEL   296

            6834 GLKPR-ARVRSMAVVGCDPSIMGYGPVPASKLALKKAGLSASDIGVFEMNEAFAAQILPC  7010
                 GL-P -+ +RS AVVG  P.IMG GP. A  +AL.KAGL+.SD+.+FE+NEAFA+Q   C
             297 GL-PILGVLRSYAVVGVPPDIMGIGPAYAIPVALQKAGLTVSDVDIFEINEAFASQAAYC   355

            7011 IKDLGLIEQIDEKIN-LNGGAIVGHPLGCSGARISTTLLNLMERKDVQ-FGLATMCIGLG  7184
                 +  L L- .--EK+N-L.G+. +GHPLGC+GAR  .TLLN + R+. +-+G+.+MCIG G
             356 VEKLRL-PP--EKVNPLGGAVALGHPLGCTGARQVITLLNELKRRGKRAYGVVSMCIGTG   412

            7185 QGIATVFE                                                     7208
                  G A+VFE
             413 MGAAAVFE                                                      420
```

Figure 4. Sample match to a non-*E. coli* gene (in contig EC21-76).

3.4. An Experiment: Evaluating *FIND-IT*'s Noise Sensitivity

This section contains an experimental evaluation of our gene-finding method; noise sensitivity of the algorithm is studied. The experimental method is as follows: a known gene (of length 999) was extracted from the GenBank database (the gene for *replication protein O* in bacteriophage λ) and various amounts of noise added to it in each of twenty-five experimental runs. Following this, we applied FIND-IT and counted the times it found the initial gene. We investigated three simple noise models and one composite model:

Replacement
With probability *p*, a given nucleotide is replaced with another one.

Deletion
With probability *p*, a given nucleotide is deleted.

Insertion
With probability *p*, a nucleotide is inserted after a given nucleotide.

Combination
With probability *p*, one of the above three changes occurs at a given nucleotide. All three possibilities are equally likely.

These noise models are somewhat simplistic — due to the nature of the sequencing process, insertions and deletions are most likely to occur within *runs* of the same nucleotide (e.g., $\cdots$ *AAAAA* $\cdots$). Nevertheless, these models are sufficient for our present purposes.

Figure 5 contains the results of this experiment. This experiment indicates that, under all four noise models, FIND-IT is unaffected until the noise rate exceeds 3%. Recall that the estimated error rate in sequencing is 1%; thus, the FIND-IT approach should robustly find genes in the sequences biological laboratories are producing.

4. Current Research Issues

We are improving and extending the FIND-IT method. Current activities include improving the algorithm's efficiency, tuning its parameters, and adding the ability to locate good, "dense" matches that do not cover a sizable portion of a protein. The last of these will prove useful for finding matches to protein "domains"[5]; unfortunately, in the present implementation, locating such matches is unacceptably expensive computationally.

We are studying the computational complexity of the "covering" problem defined in a previous section and devising efficient (possibly heuristic) algorithms for it. The problem of constructing consistent covers (defined above) can be modeled by a combinatorial optimization problem known as *maximum weight matching in an interval graph*. A restricted version of this problem, where all the weights are one, can be solved in O(N log N) steps [.Golumbic Hammer, Masuda Nakajima.], and preliminary work by Wisconsin computer science graduate student J. Meidanis has shown that an O(N log N) algorithm is also possible for the general case. This algorithm shows promise as a tool for rapidly finding consistent coverings; we plan to implement it, and to decide which edge-weighting scheme leads to the most useful overall covers.

When no similarity matches to known proteins are possible, we need alternative methods for recognizing genes. We are investigating some complementary approaches, some of which involve neural-network learning, that either (1) identify regions of DNA that have the same general, "global" statistics of known genes or (2) locate stretches of DNA that are known to

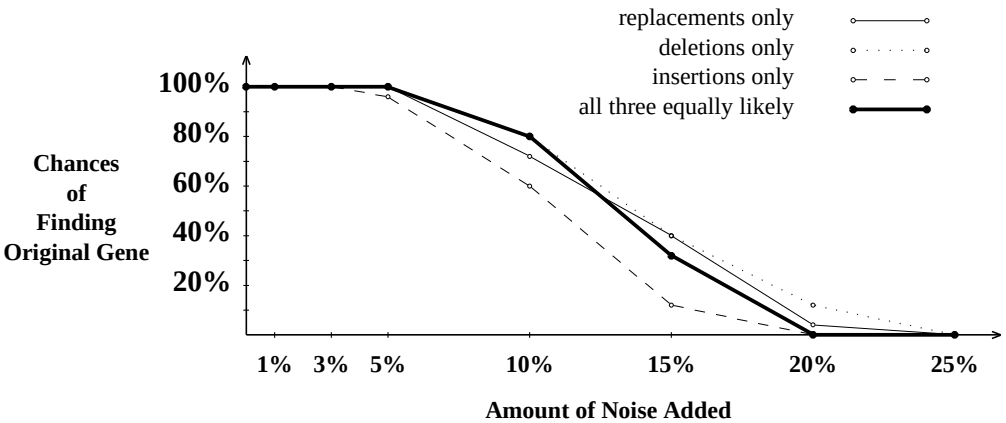

Figure 5. FIND-IT's chances of finding a gene as a function of sequence noise. (Results averaged over 25 experimental runs.)

[5] A *domain* is a portion of a protein that possesses some "stand-alone" function.

"signal" some biological activity (e.g. *promoter* regions bind the protein that initiates transcription, which is followed by translation [.shavlik aaai90.]). Research along the lines of the first approach includes [.staden doolittle, gribskov burgess.]. Approach two has also been investigated [.stormo perceptron, Lapedes Sirotkin, shavlik aaai90, Noordewier nips3.].

5. Conclusion

We have presented a case-based approach to gene finding that is robust in the presence of errors — both in the input data and in the case libraries. These errors, particularly frameshift errors, greatly complicate the task of determining the boundaries of cases, due to the triplet nature of the genetic code. This research addresses the important general question of what makes a case; more specifically, how do we parse the "noisy" world into discrete cases for matching against a case library? If the current case is improperly delimited, partial matching with previous cases will fail. Our algorithm addresses the problem by producing multiple, partial matches to many cases and then combining some subset of them into a consistent whole. This leads to error detection and correction. Our general idea for robust case matching promises to be applicable in other domains involving "continuous" data, such as speech recognition and vision. We are successfully applying the FIND-IT method in support of a Human Genome project in a Wisconsin genetics laboratory, and have already found several previously unknown *E. coli* genes.

ACKNOWLEDGMENTS

Discussions with Fred Blattner, Donna Daniels, Guy Plunkett, Eric Bach, Debby Joseph, Prasoon Tiwari, Mick Noordewier, and Joao Meidanis greatly improved this research. Special thanks go to Fred Blattner, director of the *E. coli* sequencing project, for providing the sample ‘‘contigs’’ and for pointing out the problem of matching DNA that contains frame-shift sequencing errors.

REFERENCES