Orphan gene finding – An exon assembly approach

Philippe Blayo [◇] Pierre Rouzé [†] Marie-France Sagot [◇‡]
[◇] Institut Gaspard-Monge,
Université de Marne-la-Vallée
5, bd Descartes
Champs-sur-Marne
77454 – Marne-la-Vallée cedex 2, France
[†] Laboratoire Associé de l'INRA
University of Ghent
Ledeganckstraat 35
9000 – Gent, Belgium
[‡] Inria Rhône-Alpes
Laboratoire de Biométrie et Biologie Évolutive
Université Claude Bernard (LyonI)
43, Bd du 11 Novembre 1918
69622 – Villeurbanne cedex, France
Corresponding author: sagot@biomserv.univ-lyon1.fr

Running head: Orphan gene finding

Author to whom proofs should be sent:
Marie-France Sagot
Laboratoire de Biométrie et Biologie Évolutive
Université Claude Bernard (LyonI)
43, Bd du 11 Novembre 1918
69622 – Villeurbanne cedex, France

Abstract

This paper introduces an algorithm for finding eukaryotic genes. It particularly addresses the problem of orphan genes, that is of genes that can not, based on homology alone, be connected to any known gene family and to which it is therefore not possible to apply traditional gene finding methods. To the best of our knowledge, this is also the first algorithm that attempts to compare in an exact way two DNA sequences that contain both coding (*i.e.* exonic) and non coding (*i.e.* intronic and, possibly, intergenic) parts. The comparison is performed following an algorithmical model of a gene that is as close as possible to the biological one (we consider in this paper the "one ORF, one gene" problem only). A gene is seen as a set of exons that are pieces of an assembly and are not independent. The algorithm is efficient enough: although the constants are higher than for usual sequence comparison, its time complexity is proportional to the product of the sequences lengths while its space complexity scales linearly with the length of the smallest sequence.

key words: orphan gene, gene finding, exon assembly, DNA/DNA and DNA/protein comparison, coding DNA comparison models, dynamic programming

1 Introduction

Gene finding is an important but difficult problem both in terms of biology and algorithmics. In eukaryotes, in contrast to prokaryotes, the problem is compounded by the presence of introns that interrupt the coding regions of genes, the exons. Finding a gene in the case of eukaryotes, as will be our concern in this paper, implies therefore not only finding the beginning and end of the gene in a genomic sequence, but also determining the internal structure of the gene, that is, the beginning and end of each exon.

Many algorithms and software for gene finding in eukaryotes have been elaborated, particularly as sequencing projects for whole genomes have proliferated in the last few years. One may consult the web page whose URL is http://linkage.rockfeller.edu/wli/gene/right.html for pointers to an extensive bibliography and references on the subject. Currently published algorithms call a gene what represents to the biologist the "protein coding region of the genes". What is denoted by an exon corresponds therefore to a "coding exon" as opposed to a "non-coding exon". This last may be found in a gene due to the possible presence of introns in the 5' and 3' non-coding parts of a gene. This is a simplification of reality that we also adopt in this paper for ease of exposition. For the same reason, we henceforward use the terms gene and exon to denote what are in fact, respectively, the protein coding region of genes and a coding exon.

Current algorithms commonly tackle the problem of gene finding by using one or, more generally, a combination of three basic approaches. The first two may be called intrinsic, they use information contained in the sequence only, while the last is extrinsic, it is based on information coming from outside the sequence as well as inside.

The first approach consists in identifying genes and their exon/intron structure based on an observed difference of composition between intergenic, interexonic and exonic regions ([13] [31] [32] were the pioneer work in this area). This is most often not a fine enough criterium to accurately establish the exact frontiers of genes, and of exons inside genes, at least in eukaryotes. The method may also be strongly organism-dependent.

The second approach tries to improve the localization of the gene and exons frontiers by identifying some signals that occur at, or near the limits between gene/intergene on the one hand, and exon/intron on the other [5] [6] [26] [34] [36]. The signals are in general not exact; indeed, they may often be quite degenerate in relation to a consensus. Those typically considered are splicing sites and poly-A signals. The splicing sites taken into account are mostly donors and acceptors. Branch points may also be sought as they may help to find acceptors. These are all signals occurring in introns, donors at the beginning and acceptors at the end whereas branch points are located near the end of introns. Promoters have been suggested as important signals to help identify gene starts but they are currently considered too difficult to model [5] [6]. Chris Burge is the only one at the present time who incorporates a simplified promoter model (for the TATA-box) in his program (GenScan [5] [6]). Like the first approach, this second may also be organism-dependent. It is in a likewise manner based on information about the organism that has to be learned before it can be used. Both approaches are statistical in nature.

The third and, historically, the last approach (in terms of algorithmics because in terms of manually performed analyses, this represents the first approach) to be used for gene finding has been homology with cDNAs (sometimes ESTs) or with previously determined genes, often the protein products of the genes that one may find annotated (not always accurately) and stored in databases. Algorithms using such an approach proceed in general in two successive steps: in a first step they find candidate exons in a DNA sequence believed to contain a gene and then, in a second step, they select the best among all subsets of candidates that may be assembled to form a potential gene. Selection of the best is done by comparison of the potential gene, as a DNA sequence [14] or after translation to a protein [15] [16] [17], to known cDNAs or database proteins. This second step is called exon assembly [20] or spliced alignment [16] [17]. Candidate exons may be found by any method, including either predictive like the two mentioned above (based on a difference of composition and/or the presence of sometimes weak signals) or exhaustive. The possibility of exhaustive enumeration comes from the known fact that (i) eukaryotic genes start with a particular codon (most often, but not exclusively, this is ATG) and end with one of three possible stop codons (TAA, TAG or TGA) and (ii) donors start with one among a few specific dinucleotides, usually this is GT (for introns of U2 type), or, much less frequently, GC, and acceptors also end with a specific dinucleotide which in general is AG. An exon is therefore always something framed by two known dinucleotides, or starting/ending with a given triplet. Gelfand *et al.* [16] [17] call "site mode" the exhaustive method for producing a best spliced alignment, and "exon mode" or "exon/intron mode" methods based on a preliminary filtering of potential exons by using a score that combines the strengh of predicted splicing sites with the coding potential of the exon. Depending on the method used, a large number of candidate exons may be produced.

It appears [7] [8] [12] [28] that the homology approaches, alone or in conjunction with the other two (prediction based on content or presence of signals), produce the best results for gene finding. However, as this was usually formulated until recently, homology considerations could only be used when an homolog of the gene one was trying to predict was already present in the database and, incidentally, when the database gene had itself been accurately annotated. Both assumptions are often not true. In the case of *Saccharomyces cerevisiae* (yeast) for instance, approximately 40% of predicted genes have no known homologs in the databases [10]. This rate varies between organisms but is often as high as for yeast. Some statistics exist for the number of inaccurately annotated genes (see [33] for eukaryotes – the case of prokaryotes is a little less dramatic). They point to the poor status of current sequence databases and to the great number of annotation errors they contain. This number is bound to grow as whole genome sequences are spurt out and often summarily analysed at ever increasing speeds. Very few algorithms take into account the possibility of such errors, that may happen anywhere, including within exons. The result is that what is often detected (by programs such as tblastx [1], Procrustes [16] [17] or GeneWise (http://www.sanger.ac.uk/Software/Wise2/) for instance) are incomplete pieces of exons or exons that do not contain any frameshifts.

Genes without known homologs (and one may include here genes with very weak homologs or with homologs possibly inaccurately annotated) have been called orphan genes in the literature [10]. They are orphan in the sense that they can not, based on sequence homology alone, be connected to any known gene family. One may still use homology to predict in a more reliable way their internal structure if one considers that an orphan may have brothers that are orphans like itself. Orphan brothers may correspond to ortholog or to paralog genes, that is, genes that are the result of a duplication event in a same organism. We shall hereafter call such genes, *parent genes*. Although the gene structure of parent genes is often similar, *i.e.* they have the same number of exons in the same order, this structure may neverthess differ (*e.g.* by addition or loss of introns and/or of domains). Comparing parent orphan genes to better determine their internal structure is the purpose of this paper.

The objects we shall consider are therefore two DNA sequences, each potentially containing one complete gene, plus possibly some intergenic parts both upstream and downstream from the gene. Moreover, it will be assumed that the gene in one sequence is parent with the gene in the other. Although we aim at finding a gene structure using two parent genomic sequences, we can treat as a particular case the problem of gene finding using a cDNA as one of the parent genes [14]. The case of alternative genes (*i.e.* of one DNA sequence being read in various ways, differing in the exon structure due to the use of alternative splice sites, or alternative sites of transcription initiation or termination) is more complex to handle and will therefore be considered elsewhere. In both biological and algorithmical terms, one could approach the problem we are contemplating in two basic ways. One way corresponds to an exclusively exonic model of a gene, that is to a model that considers a gene as a set of independent coding exons, each exon corresponding to a "well-conserved" fragment of two homologous genes. The second approach sees a gene as a set of coding exons that are pieces of an assembly and are, therefore, not independent. Each internal piece is thus related to the exons coming just before and after, as well as to the neighbouring introns. This paper is concerned with introducing an algorithm for the second model, that is, for a coding exon assembly approach to orphan gene finding. If the statistical properties of exons are still poorly understood in the case of one organism as mentioned by Gelfand *et al.* [16] [17], the use of statistics becomes even trickier when potentially two organisms are concerned, as will often be the case for our orphans. The approach adopted rests therefore on a model by "pure" homology. This means in particular that all potential left and right exon frames will be considered. In the case of this paper, these frames will be AG and GT respectively, however other possible frames could easily be considered without modifying the algorithm's asymptotic complexity. Although not using other assessable biological information such as content and presence of signals may weaken the predictive power of our algorithm, this last is completely organism-independent (except for the choice of a substitution matrix). This is important for our purposes. Our aim is not to replace or even compete with currently developed gene finding packages, but to complement them where these packages behave poorly or fail to provide a solution (among others, finding the structure of orphan genes).

Algorithms for comparing DNA sequences, either globally or locally, have existed for a long time; algorithms for comparing coding DNA sequences are newer and we briefly discuss the models on which they may be based in section 2.2; to the best of our knowledge, no exact algorithm has ever been elaborated for comparing DNA sequences that contain both coding (the exons) and non coding parts (mainly the introns but also the intergenic regions) which is the problem we have to treat here. Other approaches which compare two DNA sequences in order to infer the structure of the gene they may possess in common have appeared since this paper was first written [4] [3][35] but all three are heuristics besides presenting sometimes other limitations ([4] for instance can not consider genes with a different gene structure, that is, with a different number of exons). Providing a first algorithm that attempts to exactly compare two genomic DNA sequences is therefore the main theoretical and algorithmical contribution of this paper. The technique used is dynamic programming as in Procrustes [16] [17]. Procrustes has a complexity in $O(nm)$ and a space complexity in $O(\min\{m, n\})$ (using the "site mode" approach) for comparing two DNA sequences of lengths n and m at the DNA level only, one of them being of known structure and without allowing for frameshifts (the complexity for comparing one error-free DNA sequence and a known protein is not given by the authors). The challenge was to obtain the same time and space complexities although treating the considerably more complex problem of comparing two DNA sequences presenting an unknown, potentially different structure, and possibly containing sequencing errors. The algorithm introduced in this paper has therefore a time complexity in $O(mn)$ and a space complexity in $O(\min\{m, n\})$. Both complexities are asymptotically independent of the number of potential exon candidates one wishes to consider. The constant factors in both cases, although bigger than for usual sequence comparison algorithms, are reasonable enough to make the algorithms efficient in practice. This is demonstrated in an application section at the end of the paper.

2 Models and Statement of the Problem

2.1 Coding DNA Comparison Model

Up to now, there have been essentially three models for comparing DNA coding sequences. All are based on codon (*i.e.* triplet) alignments in the sense that, for each triplet of nucleotides in both sequences that are matched as amino acids:

- the number of gaps that may be present at codon boundaries is a multiple of three;

- the number of gaps internal to the codon that are allowed is either zero or may be positive (in this case, the nucleotides in the triplet considered will not appear consecutive in the sequence) but is a multiple of three.

The models vary therefore according to the number and type of gaps permitted, and also whether frameshifts are considered, and how.

Hein suggested in [21] [22] that there are 11 possible types of codon alignments (see Figure 1) but in fact, more may be considered [29].

The most sophisticated model established so far [21] [22] [25] [30] for comparing two coding DNA sequences allows for all the 11 codon alignment types suggested by Hein. Observe that in this case, the order of the events that lead to a particular codon alignment (nucleotide substitution or gap first) may not be indifferent and has to be taken into account. For complexity reasons, these algorithms do not consider the possibility of sequencing errors, that is, they do not allow for frameshifts, although incorporating this within the model is not impossible as is indicated at the end of both Pedersen's and Hua's papers.

The simplest coding DNA comparison model allows for codon alignments of types 1, 2 and 3 only (see Figure 1). They therefore perform the comparison entirely at the protein level.

The third and last currently established class of coding DNA models also treat codon alignments of types 1, 2 and 3 only, as for the protein/protein model. However, they further consider the possibility of frameshifts [2] [19]. An extended alphabet of codons $\Sigma = N^0 \cup N^1 \cup \ldots \cup N^c = N^{\leq c}$ may thus be used for such purpose, where c is a positive integer, N = {A, C, G, T} and $N^0 = \{\epsilon\}$ (ϵ denoting the empty word). In Arvestad [2], $\Sigma = (N^{\leq 5} - N^0)$. Frameshifts due to sequencing errors may therefore correspond to the insertion or deletion of one or two nucleotides anywhere in a codon. The score of a pair of codons from the extended alphabet $(\tilde{x}, \tilde{y})$ is then the maximum of the scores of all pairs of triplets (x, y) with $x, y \in \overline{N}^3$ that may be obtained from $\tilde{x}$ and $\tilde{y}$ respectively where $\overline{N} = N \cup \{-\}$ ($-$ denoting an indel).

We adopt a slightly different version of Arvestad's model in this paper. In our case, the extended alphabet of codons Σ will be equal to $N^{\leq 3}$. Our aim is to recover as gracefully as possible from a frameshift event due to a sequencing error, not to try to make a guess (based on an optimisation criterium that may seem arbitrary) of what that error was. For this reason also, we use a simpler scoring system for codons from the extended alphabet that is as follows. Given $\tilde{x}, \tilde{y} \in \Sigma = N^{\leq 3}$, we have:

$$w(\tilde{x}, \tilde{y}) = \begin{cases} \mathcal{M}(\tilde{x}, \tilde{y}) & \text{if } |\tilde{x}|, |\tilde{y}| = 3 \\ \gamma \times \min\{(|\tilde{x}| + |\tilde{y}|), (6 - |\tilde{x}| - |\tilde{y}|)\} & \text{in all other cases} \\ & \text{except if } |\tilde{x}| = 0 \text{ and } |\tilde{y}| = 3 \\ & \text{or if } |\tilde{x}| = 3 \text{ and } |\tilde{y}| = 0 \\ & \text{both of which correspond to a gap} \end{cases} \tag{1}$$

where

- w is the weighting function for the extended alphabet of codons; it is defined from $\Sigma \times \Sigma$ to $\mathbb{Z}$;
- $\mathcal{M}$ is a protein scoring matrix such as PAM [9] or Blosum [23];
- $\gamma \in \mathbb{Z}$ is a penalty for frameshifts.

Frameshifts have therefore a constant cost instead of an affine cost as will be the case for gaps. This is motivated by algorithmical considerations (both treatment and exposition become easier) but also by the fact that we do not know whether the same reasoning as for gaps could be applied to frameshifts, that is, whether an error committed in "reading" a letter at a given position in a sequence makes it more probable that a second error will be committed just afterwards during sequencing.

2.2 Gene Model

We distinguish the gene model that is considered from the model that we use for finding a gene. The second is an algorithmical concept while the first is a biological one. To simplify exposition, we assume from now on that the first codon of a gene is ATG, a donor always starts with a GT and an acceptor always ends with an AG. Considering other codon starts or donors (resp. acceptors) starts (resp. ends) does not increase the asymptotic behaviour of the algorithm and can easily be implemented as a parameter of the algorithm.

The biological model of gene we adopt is thus the following:

- a gene begins with an ATG;
- a gene ends with a stop codon (TAA, TAG or TGA);
- a gene contains any positive number of compatible exons, possibly only one, where:
  - an exon starts with an ATG (first exon) or is framed upstream by an AG (internal exon);
  - an exon ends with a stop codon (last exon) or is framed downstream by a GT (internal exon);
  - an exon codes for part of a protein;
  - an exon is compatible with the exon that comes just before or after (if any) if it does not overlap it and the two are in phase (assuming there is no frameshift).

Two exons, $exon_1$ and $exon_2$ (considered in that order), are in phase if:

- either both end, respectively start, with a codon;
- or, in the case where a codon is interrupted by an intron, the number of nucleotides (1 or 2) at the end of $exon_1$ and the number of nucleotides (1 or 2) at the beginning of $exon_2$ add up to three (see Figure 2).

We call *dangling* end (or start) of an exon the nucleotides in the last (or first) of its codon (possibly the codon is cut by an intron), and the nucleotides themselves, *dangling nucleotides*.

The algorithmical model we use to predict genes sees a gene as an ordered set of exons that correspond to the pieces of an assembly. Since exons code for proteins, they are scored as such. The scoring function used is the one given in equation 1 above. The score of an ordered set of exons assembled into a gene is the score of the assembly where gaps and frameshifts are allowed and exons frontiers are taken into account (see comment at the end of section 3.4.2). The cost of a gap is an affine function of its length l. It is thus $g_{open} + lg_{extend}$ where g_{open} is the cost for opening a gap and g_{extend} is the cost for extending the gap.

Finally, it is implicit in both the biological and algorithmical models that two parent genes may have a different number of exons. In particular, one of the genes could have only one exon. It could even correspond to a cDNA. The cDNA/DNA comparison problem is thus a special case of the one we treat. In the case of different gene structures, possibly corresponding to the deletion of an intron in one or the other sequence, a penalty for such mutational event may be considered and easily incorporated into the algorithm.

2.3 Statement of the General "Gene Prediction as Exon Assembly" Problem

Given two sequences $a \in N^m$ and $b \in N^n$, each one containing a gene that is parent with the gene present in the other, a protein matrix function, an affine gap cost function and a scoring function for an extended alphabet of codons (that incorporates a frameshift penalty), the algorithm gives as result the set of exons that may be assembled into a gene verifying the biological model defined above, and has highest score among all sets.

3 Algorithm

3.1 Basic Definitions

Let $s \in N^+$ be a generic DNA sequence. We note $|s|$ its length and $s[1..i]$ its prefix of length i. We use the symbols x and y to denote codons, that is, triplets of letters from N, $\tilde{x}$ and $\tilde{y}$ to denote a codon from the extended alphabet, that is, elements of $\Sigma = N^{\leq 3}$.

3.2 Main Ideas

The algorithm we propose for solving the "Gene Prediction as Exon Assembly" problem consists in:

- comparing the sequences taking into account the fact that they are made up of an alternation of coding and non coding regions and using for this purpose a dynamic programming matrix (in fact three because we are using an affine gap cost function [18]);
- locating at the same time a best scoring set of exons verifying the biological gene model.

Before giving the recurrence relations that are at the core of the algorithm, we present a sketch of the algorithm's main ideas. In all that follows, we suppose, without loss of generality, that sequence a is the longest one, that is $n \leq m$.

As we go comparing sequences a and b by filling up the dynamic programming matrices almost the usual way (from left to right, top to bottom but in our case, considering an extended alphabet of codons instead of nucleotides), every time an ATG is met in both a and b, we make the hypothesis that we are inside the first exon of the parent genes in the two sequences.

In the same way, when an AG is encountered later on in any of the two sequences (possibly both), we could be entering a new exon of the parent genes. All possible joins with the best assemblies that are compatible and end before it in both sequences have to be contemplated. We therefore need to store all information relative to these best assemblies.

Finally, when a GT is crossed, again in any of the two sequences (possibly both), we have to consider that, just before it, we reached the potential end of an exon, not necessarily the first one, in the sequence(s) where the GT was met. Let i, j denote the positions we had reached in a and b respectively when the GT was encountered. Since we have been keeping somewhere the information concerning the best assemblies ending before line i and column j (see above), it is easy to check whether our exon could be compatible with some of them. If it is, we need to update the information concerning these best assemblies.

The information about the best assemblies reached so far in our traversal of the dynamic programming matrices can be grouped into three main types: two concerning best assemblies interrupted by an intron in one sequence only, and one concerning the best assemblies interrupted by an intron in both sequences. Each piece of information is then subdivided for both biological and algorithmical purposes into as many parts as there are possible last events for an exon (substitution, insertion or deletion) and, in the case of substitution, quantity of information (corresponding to the number of nucleotides) that has to be memorized. Depending on that number, we choose to recall known nucleotides or anticipated ones (*i.e.*, the combination of nucleotides that may come at the start of a next exon). Indeed, when an intron is situated between two codons, or splits a codon in two parts, we may:

- complete the trunkated codons at the beginning of the potential exons coming after the intron only when these are met later in the algorithm. For this, we need to store the value of the dangling nucleotides at the end of the currently considered exon – that is, the codons split by the intron – and, for each such value, the scores of the best assemblies up to, but not including these dangling nucleotides;

- anticipate the completion of the codons split by an intron by considering the values that would be reached by each possible codon end. In this case, what is stored are the values of ends of split codons which were anticipated – that is, the possible nucleotides which will hang at the beginning of potential next exons – and the scores of the best assemblies obtained up to, and this time including the codon whose split parts border the intron.

We can substantially decrease the constant of the algorithm by choosing to sometimes recall known dangling nucleotides (nucleotides that are at the end of a currently considered exon), and sometimes anticipate such dangling nucleotides (nucleotides that will be at the start of a next potential exon).

The information just described is all we need to decide at the end what is the score of a best assembly. We shall see that, so far, we have a time complexity that is in $O(mn)$.

The last problem we have to face consists in recovering a best assembly. We assume for now that we keep the whole matrix until the end of the algorithm. In that case, the problem is trivial but requires quadratic space. We later show how this can be done in space linear in the length of the smallest sequence as for a "normal" pairwise sequence comparison (section 3.5). This will imply adding a few more fields to the data structure used and modifying the algorithm. These changes do not substantially alter the spirit of the algorithm, we therefore postpone discussing them for didactic purposes.

3.3 Data Structures

The algorithm uses three main types of data structures. As describing these structures is essential for understanding the recurrence relations that are at the heart of the algorithm as well as the algorithm time and space complexity, we describe them below in some detail. These data structures are:

- Dynamic programming matrices: since an affine gap cost is used, three matrices are needed [18], that we denote $\mathcal{S}$, $\mathcal{D}$ and $\mathcal{I}$ (for Substitution, Deletion and Insertion). These matrices are defined from $[0, m] \times [0, n]$ to $Cell$ where $Cell$ is a structure with the following fields (all positions are identified by a pair of integers, one for each sequence):

$beginAssembly$	start positions of the first exon of the assembly leading to the cell;
$score$	score of the assembly leading to the cell (this is an integer).

- Arrays for storing the value of the best assembly built so far that ends at a given column j for all $j \in [1, n]$ (end of exon in a only), before a given column j for all $j \in [1, n]$ (end of exon in both sequences) and at a given line i for the three lines preceeding a currently treated one and before a given column j for all $j \in [1, n]$ (exon present in b only – since we fill the matrices one line at a time, we do not need to preserve information about a best assembly ending in lines considered earlier than that). As discussed in the previous section, for space considerations, we keep for each type two arrays, one indexed on currently considered, and therefore known dangling ends of exons and one indexed on anticipated dangling starts of exons. We call these arrays, respectively, e_a^{known}, $e_a^{anticipated}$ (end of exon in a), e_{ab}^{known}, $e_{ab}^{anticipated}$ (end of exon in a and b), e_b^{known} and $e_b^{anticipated}$ (end of exon in b). We have:
  - e_a^{known}, $e_a^{anticipated}$, e_b^{known}, $e_b^{anticipated}$, e_{ab}^{known} and $e_{ab}^{anticipated}$ defined from $[1, p] \times (N \cup \{\epsilon\})^2 \times (N \cup \{\epsilon\})^2$ to $Cell$ where p is n for e_a and e_{ab} and $3n$ for e_b;

    (for arrays e_s^{known}, we keep the best for each possible combination of dangling nucleotide(s) at the end of the current exon (0 on both sequences, or 1 and 0, or 1 on both, or 2 and 0, or 2 and 1); for arrays $e_s^{anticipated}$, we consider each possible combination of anticipated dangling nucleotide(s) at the start of the next exon (1 on both, or 1 and 0, or 2 and 0));

- $e_a^{I\varphi}$, $e_a^{D\varphi}$, $e_b^{I\varphi}$, $e_b^{D\varphi}$, $e_{ab}^{I\varphi}$ and $e_{ab}^{D\varphi}$ defined for all possible phases φ (0, 1 or 2) from $[1, p]$ to $Cell$ where p is as before.

- A variable $Best$ of type $Cell_{Best}$ for storing information concerning a best assembly where $Cell_{Best}$ is the same as $Cell$ except that there is an additional field, noted $endAssembly$, that stores the end positions of the assembly.

3.4 Recurrence Relations

We have two classes of recurrence relations. One is concerned with filling up or updating the values of the cells in arrays e_a, e_b and e_{ab}. This is done each time we meet a GT in either one, or both sequences. These relations are shown in section 3.4.1. The second class of recurrence relations concerns matrices $\mathcal{S}$, $\mathcal{I}$, and $\mathcal{D}$. The relations are presented in section 3.4.2. To avoid overloading the formulas, we use $\mathcal{S}(i,j)$ to denote the score of matrix $\mathcal{S}$ at positions i and j instead of $\mathcal{S}(i,j)_{score}$. The same applies to matrices $\mathcal{I}$ and $\mathcal{D}$.

3.4.1 Potential End of Exon Relations

Observe that:

- e_a, e_b and e_{ab} are updated if $a_{i+1}a_{i+2} =$ GT and $b_{j+1}b_{j+2} =$ GT;
- e_a is updated if $a_{i+1}a_{i+2} =$ GT;
- e_b is updated if $b_{j+1}b_{j+2} =$ GT.

The recurrence relations for arrays e_a, e_b and e_{ab} are the following (α_i denotes here anticipated nucleotides):

the max in the first three cases is taken over $\tilde{x} \in \{\epsilon, a_i, a_{i-1}a_i\}, \tilde{y} \in \{\epsilon, b_j, b_{j-1}b_j\}$, $|\tilde{x}| + |\tilde{y}| \leq 3$

$$\begin{array}{lcl} e_a^{known}(j,\tilde{x},\tilde{y}) & = & \max\{e_a^{known}(j,\tilde{x},\tilde{y}), \mathcal{S}(i-|\tilde{x}|, j-|\tilde{y}|)\} \\ e_b^{known}(i,j,\tilde{x},\tilde{y}) & = & \max\{e_b^{known}(i,j,\tilde{x},\tilde{y}), \mathcal{S}(i-|\tilde{x}|, j-|\tilde{y}|)\} \\ e_{ab}^{known}(j,\tilde{x},\tilde{y}) & = & \max\{e_{ab}^{known}(j,\tilde{x},\tilde{y}), e_{ab}^{known}(j-1,\tilde{x},\tilde{y}), \mathcal{S}(i-|\tilde{x}|, j-|\tilde{y}|)\} \end{array} \tag{2}$$

the max in the next three cases is taken over $0 < \alpha_1, \alpha_2 \in N^{\leq 2}, 0 < |\alpha_1| + |\alpha_2| \leq 2, \tilde{x}\alpha_1, \tilde{y}\alpha_2 \in N^{\leq 3}$

$$\begin{array}{lcl} e_a^{anticipated}(j,\alpha_1,\alpha_2) & = & \max\{e_a^{anticipated}(j,\alpha_1,\alpha_2), \mathcal{S}(i-|\tilde{x}|, j-|\tilde{y}|) + w(\tilde{x}\alpha_1, \tilde{y}\alpha_2)\} \\ e_b^{anticipated}(i,j,\alpha_1,\alpha_2) & = & \max\{e_b^{anticipated}(i,j,\alpha_1,\alpha_2), \mathcal{S}(i-|\tilde{x}|, j-|\tilde{y}|) + w(\tilde{x}\alpha_1, \tilde{y}\alpha_2)\} \\ e_{ab}^{anticipated}(j,\alpha_1,\alpha_2) & = & \max\{e_{ab}^{anticipated}(j,\alpha_1,\alpha_2), e_{ab}^{anticipated}(j-1,\alpha_1,\alpha_2), \mathcal{S}(i-|\tilde{x}|, \\ & & \hfill j-|\tilde{y}|) + w(\tilde{x}\alpha_1, \tilde{y}\alpha_2)\} \end{array} \tag{3}$$

for all $\varphi \in \{0, 1, 2\}$

$$e_a^{I\varphi}(j) = \max \begin{cases} e_a^{I\varphi}(j) \\ \mathcal{I}(i, j-\varphi) \\ \mathcal{S}(i, j-\varphi) + g_{open} \end{cases}$$
$$e_b^{I\varphi}(i,j) = \max \begin{cases} e_b^{I\varphi}(i,j) \\ \mathcal{I}(i, j-\varphi) \\ \mathcal{S}(i, j-\varphi) + g_{open} \end{cases}$$
$$e_{ab}^{I\varphi}(j) = \max \begin{cases} e_{ab}^{I\varphi}(j) \\ e_{ab}^{I\varphi}(j-1) \\ \mathcal{I}(i, j-\varphi) \\ \mathcal{S}(i, j-\varphi) + g_{open} \end{cases} \tag{4}$$

$$e_a^{D\varphi}(j) = \max \begin{cases} e_a^{D\varphi}(j) \\ \mathcal{D}(i-\varphi, j) \\ \mathcal{S}(i-\varphi, j) + g_{open} \end{cases}$$
$$e_b^{D\varphi}(i,j) = \max \begin{cases} e_b^{D\varphi}(i,j) \\ \mathcal{D}(i-\varphi, j) \\ \mathcal{S}(i-\varphi, j) + g_{open} \end{cases}$$
$$e_{ab}^{D\varphi}(j) = \max \begin{cases} e_{ab}^{D\varphi}(j) \\ e_{ab}^{D\varphi}(j-1) \\ \mathcal{D}(i-\varphi, j) \\ \mathcal{S}(i-\varphi, j) + g_{open} \end{cases} \tag{5}$$

Variables and arrays e_a, e_b and e_{ab} are initialized with $-\infty$ (meaning $(-\infty, -\infty)$ for pairs).

3.4.2 General Relations

The general relations given below indicate how to fill the cells in matrices $\mathcal{S}, \mathcal{I}$ and $\mathcal{D}$. We have to distinguish whether the positions in the sequences we are considering represent the potential start of an exon in either (or both) sequences or not.

Not Potential Start of Exon Recurrence relations for positions in the sequences that are not potential start of exons in either sequence are similar to those given in Arvestad's paper [2] except that the extended alphabet of codons is not the same (see section 2.1).

$$\mathcal{S}(i,j) = \max \begin{cases} \mathcal{I}(i,j) \\ \mathcal{D}(i,j) \\ \max\limits_{\substack{\tilde{x}, \tilde{y} \in N^{\leq 3}, |\tilde{x}| + |\tilde{y}| > 0 \\ |\tilde{x}| \neq 0 \text{ if } |\tilde{y}| = 3 \text{ and} \\ |\tilde{x}| \neq 3 \text{ if } |\tilde{y}| = 0 \\ a[1..(i-|\tilde{x}|)]\tilde{x} = a[1..i] \\ b[1..(j-|\tilde{y}|)]\tilde{y} = b[1..j]}} \left\{ \mathcal{S}(i-|\tilde{x}|, j-|\tilde{y}|) + w(\tilde{x}, \tilde{y}) \right. \end{cases} \tag{6}$$

$$\mathcal{I}(i,j) = \max \begin{cases} \mathcal{I}(i, j-3) + g_{extend} \\ \mathcal{S}(i, j-3) + g_{extend} + g_{open} \end{cases} \tag{7}$$

$$\mathcal{D}(i,j) = \max \begin{cases} \mathcal{D}(i-3, j) + g_{extend} \\ \mathcal{S}(i-3, j) + g_{extend} + g_{open} \end{cases} \tag{8}$$

The three matrices are initialized with $-\infty$ in lines $0, 1, 2$ and columns $0, 1, 2$ (meaning for pairs $(-\infty, -\infty)$).

Potential Start of Exon When a potential exon start is met in either or both sequences, the relations to be used are instead the following (α_i denotes here nucleotides at the end of an exon in an assembly built so far):

$$
\mathcal{S}(i,j) = \max \begin{cases}
\mathcal{I}(i,j) \\
\mathcal{D}(i,j) \\
\text{/* the end of the previous exon is a complete codon */} \\
\max\limits_{\substack{\tilde{x} \in \{\epsilon, a_i, a_{i-1}a_i, a_{i-2}a_{i-1}a_i\} \\ \tilde{y} \in \{\epsilon, b_j, b_{j-1}b_j, b_{j-2}b_{j-1}b_j\} \\ |\tilde{x}| + |\tilde{y}| > 0 \\ |\tilde{x}| \neq 0 \text{ if } |\tilde{y}| = 3 \text{ and} \\ |\tilde{x}| \neq 3 \text{ if } |\tilde{y}| = 0}} \{\mathcal{S}(i - |\tilde{x}|, j - |\tilde{y}|) + w(\tilde{x}, \tilde{y}) \\
\text{/* the end of the previous exon is a split codon */} \\
\text{/* and we use stored information about what was then known dangling nucleotides */} \\
\max\limits_{\substack{\alpha_1\tilde{x}, \alpha_2\tilde{y} \in N^{\leq 3} \\ \tilde{x} \in \{a_i, a_{i-1}a_i, a_{i-2}a_{i-1}a_i\} \\ \tilde{y} \in \{b_j, b_{j-1}b_j, b_{j-2}b_{j-1}b_j\} \\ |\tilde{x}| + |\tilde{y}| \geq 3}} w(\alpha_1\tilde{x}, \alpha_2\tilde{y}) + \begin{cases} e_a^{known}(j - |\tilde{y}|, \alpha_1, \alpha_2) \\ e_b^{known}(i - |\tilde{x}|, j - |\tilde{y}|, \alpha_1, \alpha_2) \\ e_{ab}^{known}(j - |\tilde{y}|, \alpha_1, \alpha_2) \end{cases} \\
\text{/* the end of the previous exon is a split codon */} \\
\text{/* and we had anticipated dangling nucleotides at the start of the current exon */} \\
\max\limits_{\substack{\alpha_1\tilde{x}, \alpha_2\tilde{y} \in N^{\leq 3} \\ \tilde{x} \in \{\epsilon, a_i, a_{i-1}a_i\} \\ \tilde{y} \in \{\epsilon, b_j, b_{j-1}b_j\} \\ 0 < |\tilde{x}| + |\tilde{y}| \leq 2}} \begin{cases} e_a^{anticipated}(j - |\tilde{y}|, \tilde{x}, \tilde{y}) \\ e_b^{anticipated}(i - |\tilde{x}|, j - |\tilde{y}|, \tilde{x}, \tilde{y}) \\ e_{ab}^{anticipated}(j - |\tilde{y}|, \tilde{x}, \tilde{y}) \end{cases}
\end{cases} \tag{9}
$$

$$
\mathcal{I}(i,j) = g_{extend} + \max \begin{cases}
\text{/* continuing an insertion */} \\
\mathcal{I}(i, j-3) \\
\text{/* opening an insertion after a match or substitution */} \\
\mathcal{S}(i, j-3) + g_{open} \\
\text{/* continuing an insertion which straddles an intron in at least one sequence */} \\
g_{open} + \max\limits_{\substack{\tilde{x} = \epsilon \\ \tilde{y} \in (N^{\leq 3} - N^0)}} \begin{cases} e_a^{I^{3-|\tilde{y}|}}(j - |\tilde{y}|) \\ e_b^{I^{3-|\tilde{y}|}}(i, j - |\tilde{y}|) \\ e_{ab}^{I^{3-|\tilde{y}|}}(j - |\tilde{y}|) \end{cases}
\end{cases} \tag{10}
$$

$$
\mathcal{D}(i,j) = g_{extend} + \max \begin{cases}
\text{/* continuing a deletion */} \\
\mathcal{D}(i, j-3) \\
\text{/* opening a deletion after a match or substitution */} \\
\mathcal{S}(i, j-3) + g_{open} \\
\text{/* continuing a deletion which straddles an intron in at least one sequence */} \\
g_{open} + \max\limits_{\substack{\tilde{x} \in (N^{\leq 3} - N^0) \\ \tilde{y} = \epsilon}} \begin{cases} e_a^{D^{3-|\tilde{x}|}}(j) \\ e_b^{D^{3-|\tilde{x}|}}(i - |\tilde{x}|, j) \\ e_{ab}^{D^{3-|\tilde{x}|}}(j) \end{cases}
\end{cases} \tag{11}
$$

Since frameshifts are allowed, a potential exon start is met whenever $a_{i-|\tilde{x}|-1}a_{i-|\tilde{x}|}$ = AG or $b_{j-|\tilde{y}|-1}$ $b_{j-|\tilde{y}|}$ = AG, or both, where $\tilde{x}, \tilde{y} \in N^{\leq 3}$.

Observe that, whenever a gap crosses an intron, it is considered that two mutational events have taken place. The reason for this choice is that, although the gap appears continuous at the protein level, these events take place at different, often distant locations in at least one of the DNA sequences.

3.4.3 Finding a Best Assembly

A pseudo-code for finding a best assembly using the recurrence relations given in the previous sections is presented below.

function findBest($s_1, s_2, dangling$)
1. initialize all fields of the matrices's cells at lines and columns $0, 1, 2$ with $-\infty$;
2. initialize e_a, e_b and e_{ab} with value $dangling$;
3. **for** $i \in [0..|s_1|]$
4.   **for** $j \in [0..|s_2|]$
5.   compute $\mathcal{S}(i,j), \mathcal{I}(i,j)$ and $\mathcal{D}(i,j)$ with relation 7, 8 and 6 (potential start of exon in neither sequence) or 10, 11 and 9 (potential start of exon in either sequence or both);
6.   **if** ($a_{i-2}a_{i-1}a_i$ = ATG and $b_{j-2}b_{j-1}b_j$ = ATG)
7.   **if** ($w(a_{i-2}a_{i-1}a_i, b_{j-2}b_{j-1}b_j) > \mathcal{S}(i,j)$)
            /* potential exon beginning */
8. $\mathcal{S}(i,j) \leftarrow w(a_{i-2}a_{i-1}a_i, b_{j-2}b_{j-1}b_j)$;
9. $\mathcal{S}(i,j)_{beginAssembly} \leftarrow (i-2, j-2)$;
10.   **if** ($\mathcal{S}(i,j) > -\infty$ **and** $a_{i-2}a_{i-1}a_i$ = stop codon **and** $b_{j-2}b_{j-1}b_j$ = stop codon **and** $\mathcal{S}(i,j) > Best$)
         /* potential assembly end */
11.   all fields $Best \leftarrow$ all fields $\mathcal{S}(i,j)$;
12. $Best_{endAssembly} \leftarrow (i,j)$;
13.   **if** ($a_{i+1}a_{i+2} =$ GT)
14.   compute e_a using relations 2, 3, 4 and 5;
15.   **if** ($b_{j+1}b_{j+2} =$ GT)
16.   compute e_b using relations 2, 3, 4 and 5;
17.   **if** ($a_{i+1}a_{i+2}$ **and** $b_{j+1}b_{j+2} =$ GT)
18.   compute e_{ab} using relations 2, 3, 4 and 5;
19. **return** $Best$;

3.4.4 Taking Minimal Intron Length into Account

In the relations given so far, we have been considering that introns can be of any length. Although extreme cases have been observed, introns have in general a minimum length which may vary slightly between organisms, and which is often considered to be 40 nucleotides. It is not difficult to add this as a constraint to the algorithm. This requires however more space. The reason is that we must memorize up to $\frac{ilength_{min}}{2}$ arrays e_a, e_b and e_{ab} where $ilength_{min}$ is the minimum length of an intron one wishes to allow. The maximum of $\frac{ilength_{min}}{2}$ would be reached for an intron composed of a succession of GTs. We therefore need three arrays of dimension up to $\frac{ilength_{min}}{2}$ whose cells are e_a^{known}, $e_a^{anticipated}$, e_b^{known}, $e_b^{anticipated}$, e_{ab}^{known} and $e_{ab}^{anticipated}$ respectively. Let us note these new arrays E_a^{known}, $E_a^{anticipated}$, E_b^{known}, $E_b^{anticipated}$, E_{ab}^{known} and $E_{ab}^{anticipated}$. They are the functions now used in the recurrence relations for matrices $\mathcal{S}, \mathcal{I}$ and

$\mathcal{D}$ in place of e_a, e_b and e_{ab}. It is not difficult to see that one needs to simply change the value returned by these functions for the algorithm to take a minimum intron length into account. This value will be that of the cell at position $\frac{ilength_{min}}{2}$ in the arrays E_a, E_b and E_{ab}.

3.5 Recovering the Exons in a Best Assembly in Linear Space – The Whole Algorithm

We face now the problem of recovering a best assembly, and of doing so in linear space. The way we choose to go about this allows us to leave unchanged the asymptotic behaviour of the time complexity of the algorithm. The idea is reminiscent of the method developed by Hirschberg [24] for calculating the alignment (as opposed to just the score) of two sequences but resembles more closely the formulation given by Durbin [11]. Simply put, the core of this method uses the principle of divide and conquer to find the row (or column) where an optimal alignment crosses the middle column (resp. row) in the dynamic programming matrix, and then to iterate on the two separate submatrices thus obtained. In our case, we can not proceed exactly like this because we are not aligning sequences but potential gene structures, we do not know whether any exon passes through the middle row (or column) of the matrices and, finally, all exons are not equal (they end with different reading phases, with or without gaps or frameshift errors). However, we can adapt the idea to our purposes.

This adaptation is based on the idea that, if, for each best assembly so far met during the progress of the algorithm (including the assembly that will be a final best), we keep information on the first exon belonging to it that starts just after or at the middle column (let us denote it $exon_{afterDivision}$) as well as the exon that immediately precedes it in the assembly (denoted $exon_{beforeDivision}$), we have already the start and end positions of two of the exons we wish to recover. We then just need to iterate on the submatrices from the assembly first exon to $exon_{beforeDivision}$ on the one hand, and from $exon_{afterDivision}$ to the last exon on the other. As observed, the divide operation may not cut the assembly itself in the middle (see Figure 3). In fact, it could even fail to divide the assembly at all if the last exon (or first) in the genes occur before (resp. after) or overlap the middle column. In this case, it suffices to consider the division that leaves the last (resp. first) exon on one side and all remaining exons on the other side.

We may observe also that, when we iterate on the submatrices, we can not apply the algorithm in exactly the same way as for the first step. The reason is that the exon assemblies in the first and second submatrices must end, respectively start, with a particular exon (*i.e.*, $exon_{beforeDivision}$ which does not end with a stop codon, respectively $exon_{afterDivision}$ which does not start with an ATG). Concerning the second submatrix, the algorithm must in particular start with the knowledge of the scores preserved in arrays e_a, e_b and e_{ab}. This is reminiscent of what Myers and Miller [27] had to consider when they extended Hirschberg's method to compare biological sequences allowing for an affine gap scoring function.

Since all exons are not equal and we must remember how ended those coming just before and after a division, we need to add a few more fields to the structure $Cell$ as described in section 3.3. These fields are:

$afterDivision$	start positions of first exon after position $\frac{n}{2}$ or, if the middle column has not yet been reached, of previous exon in the assembly leading to the cell;
$beforeDivision$	end positions of exon coming just before the one registered in the previous field;
$danglingDivision$	number of dangling nucleotides of first exon after position $\frac{n}{2}$ or, if the middle column has not yet been reached, of previous exon in the assembly leading to the cell (this is a pair of integers (k, l) where $0 \leq k, l \leq 3$ and $(k + l) > 0$);

The algorithm given in section 3.4.3 must also be modified in consequence. The modification implies in line 2 initializing also the $danglingDivision$ field of $Best$ with the value $dangling$ and adding the following between lines 18 and 19 of the pseudo-code (with line 19 now becoming line 26):

19. **if** (($j \geq \frac{n}{2}$) **and** first exon beginning after $\frac{n}{2}$ has just been found)
20. **if** (first exon beginning after $\frac{n}{2}$ is not first exon assembly)
21. update fields $afterDivision$, $beforeDivision$, $danglingDivision$ in $\mathcal{S}$, $\mathcal{I}$, $\mathcal{D}$, e_a, e_b, e_{ab};
22. **else if** (first exon beginning after $\frac{n}{2}$ is first exon assembly)
23. update the above fields at next exon (this may be done using a flag, details not shown);
24. **if** ($Best_{endAssembly}$ ends before column $\frac{n}{2}$)
25. update fields $afterDivision$, $beforeDivision$, $danglingDivision$ in $Best$;

Finally, a pseudo-code for the whole algorithm is given below. It is called with the command $findAssembly$ $(a[1..n], b[1..m], -\infty)$.

function findAssembly($s_1, s_2, dangling$)
1. $Best \leftarrow$ findBest($s_1, s_2, dangling$)
2. **if** ($Best_{afterDivision} = Best_{beginAssembly}$ **and** $Best_{beforeDivision} = Best_{endAssembly}$)
/* base of recurrence: only one exon left */
3. **return** ($Best_{beginAssembly}, Best_{endAssembly}$);
4. **else**
/* divide and conquer */
5. **return** the concatenation of
6. findAssembly($a[Best_{beginAssembly}..Best_{beforeDivision}], b[Best_{beginAssembly}..Best_{beforeDivision}]$, $dangling$) and
7. findAssembly($a[Best_{afterDivision}..Best_{endAssembly}], b[Best_{afterDivision}..Best_{endAssembly}]$, $Best_{danglingDivision}$);

3.6 Complexity of the Algorithm

3.6.1 Time complexity

We count only comparison operations involving the data structures described in section 3.3. Let us consider first the pseudo-code for calculating the score of a best assembly given at the end of section 3.4.3. The two **for** loops are performed, respectively, $|s_1|$ and $|s_2|$ times. The operations inside the loops require constant time. More precisely, line 5 requires 16 (potential start of exon in neither sequence) or $(99p_{start} + 99p_{start} + 288p^2_{start} + 5p_{start} + 5p_{start} + 5p^2_{start}) + (2 \times (3p_{start} + 3p_{start} + 3p^2_{start}))$ operations (potential start of exon in either sequence or both) where p_{start} is the frequency of occurrence of AG (or any allowed set of acceptor end dinucleotide). The values $99p_{start}$ (twice) and $288p^2_{start}$ correspond to all possible combinations for $|\alpha_1|$ and $|\alpha_2|$ (where α_1, α_2 denote nucleotides at the end of an exon in an assembly built so far) given the lengths of $\tilde{x}$ and $\tilde{y}$ (currently considered nucleotides). The combinations for valid values of $|\tilde{x}|$ and $|\tilde{y}|$ are indicated in Table 1. The values $5p_{start}$ (twice) and $5p^2_{start}$ correspond to all 5 possible combinations of anticipated nucleotides (see section 3.3) and, finally, the two times $3p_{start}$ (twice) and $3p^2_{start}$ come from equations 10 and 11. Lines 14, 16 and 18 require $(8p_{end} + 252p_{end} + 12p_{end})$, $(8p_{end} + 252p_{end} + 12p_{end})$ and $(16p^2_{end} + (528 + 56)p^2_{end} + 18p^2_{end})$ operations respectively where p_{end} is the frequency of occurrence of GT (or any allowed set of donor start dinucleotide). The values $8p_{end}$, $8p_{end}$ and $16p^2_{end}$ correspond to the update of arrays e^{known}_a, e^{known}_b and e^{known}_{ab} respectively and come from the fact that we consider 8 possible combinations of dangling nucleotide(s) at the end of an exon (see section 3.3). The values $252p_{end}$, $252p_{end}$ and $(528 + 56)p^2_{end}$ correspond to the update of $e^{anticipated}_a$, $e^{anticipated}_b$ and $e^{anticipated}_{ab}$ and reflect the number of possible combinations for $|\tilde{x}|$ and $|\tilde{y}|$ given the lengths of α_1 and α_2 that had been anticipated (the $+56p_{end}^2$ in the last case comes from having to consider $e_{ab}^{anticipated}(j-1, \alpha_1, \alpha_2)$ in equation 3). The combinations are enumerated in Table 2. Finally, the values $12p_{end}$, $12p_{end}$ and $18p_{end}^2$ correspond to the updates of $e_a^{I_\varphi, D_\varphi}$, $e_b^{I_\varphi, D_\varphi}$ and $e_{ab}^{I_\varphi, D_\varphi}$. If one considers an uniform distribution of letters, the algorithm's time complexity is proportional to $(16\ (S, D, I)\ + 224p_{start} + 299p_{start}^2$ (start of exons) $+ 544p_{end} + 618p_{end}^2$ (end of exons) $)mn$, that is to $(16 + (224 + 544)\frac{1}{16} + (299 + 618)\frac{1}{16^2})mn = (16 + 51.6)mn$. The algorithm is therefore in $O(mn)$.

We have now to consider the number of times function $findBest$ is called and with what parameters. The pseudo-code for the whole algorithm given at the end of section 3.5 shows that at each step the dimensions of the submatrices we have to calculate to find a best assembly are cut by at least half until one exon only is found. A best assembly may thus be recovered in $O(mn)$ time with the same constant.

An idea of the execution time of the algorithm in practice may be seen in Figure 4. Distinct sequences of same length n were randomly generated for the test. The length indicated in the X-axis is the common length n of the two sequences. The curve in dashed line corresponds to the execution time of the algorithm, the one in full line is the curve of the function $n^2/49000$.

3.6.2 Space complexity

The data structures used were fully described in section 3.3. We need $4 \times 3 \times n$, that is $12 \times n$ cells for filling up matrices $\mathcal{S}, \mathcal{I}$ and $\mathcal{D}$. For arrays e_a, e_b, e_{ab} ($known$ or $anticipated$) on the other hand, a compact, but possibly not yet optimal representation of the information is possible that considers the exact amount of space needed for each possible combination of codon pairs from the extended alphabet, whether known or anticipated, and distinguishes whether an exon is cut by an intron in one sequence only or in both. This, plus arrays e_s^I and e_s^D for s equal to a, b and ab, requires $(79 \times n) + (79 \times 3 \times n) + (247p_{end} \times n) + 1$ cells respectively as indicated in Table 3. The space complexity of the algorithm is therefore $O(\min\{m, n\})$.

4 An Example of Application

As an example of preliminary application, the algorithm was used to pairwise compare a genomic sequence from a plant, barley (EMBL ac number: X12733), to each element in a set of sequences from various other plants, both dicotyledons (*Arabidopsis thaliana*, coton, petunia, *Vitis vinifera*, *Trifolium repens*; EMBL ac numbers: D84240, U53701, X54106, U36586, Z24755, U63931) and monocotyledons (maize, rice, pearl millet, *Washingtonia robusta*; EMBL ac numbers: X04050, X02915, M59082, M36469, U65972). All these sequences contain in their middle a gene coding for alcool deshydrogenase (ADH). The number of exons for the gene varies between 7 and 10 among the 12 sequences. The length of the ADH gene present in each sequence is also variable from one plant to another. It ranges from 982 to 1143 nucleotides. The sequences themselves have a length ranging from 2413 to 5249 nucleotides. The scoring system used was the following:

- substitution matrix: PAM 120;
- affine cost function for gaps: penalty for opening a gap of 10 and penalty for extending a gap of 9;
- penalty for a frameshift: 40;
- penalty for introducing an intron in one sequence but not the other: 20;

- minimum intron length: 55 nucleotides.

The result obtained is presented for all sequences in Figure 5. Since the gene does not start at the same place in the various sequences, the Figure allows us to simultaneously compare the sets of exons predicted to form part of the gene in each sequence on the one hand, and in the target sequence on the other.

The sensitivity/specificity at the nucleotidic and exonic levels for all the pairwise comparisons are, respectively, of 97%/98% and 80%/73%. We remind that sensitivity is calculated by dividing the number of true positives by the number of actual true elements (true positives plus false negatives) while specificity is the number of true positives divided by the number of predicted elements (true plus false positives). The sensitivity/specificity at the level of the borders of the exons (how many are missed or wrongly predicted) are better than the values obtained at the exonic level: 90%/82%. The frontier of some exons, and, specially, of the gene itself (its beginning and end) are the hardest to identify. In particular, the small speckles completely at the left-hand side of the Figure represent for the most part false positives: the gene start is often not correctly identified.

These were relatively "clean" datasets in the sense that the sequences contained little more than the genes whose structure we had to determine. The genes themselves are relatively well-conserved (better than the intronic parts) although far less than the degree of conservation programs such as ROSETTA [4] seem able to handle. In the same vein, the genes structures are quite variable, something other recent approaches which also pairwise compare genomic sequences are currently not able to deal with [4].

Other applications, to more difficult cases, will be discussed in a forthcoming paper (Aubourg, Blayo, Rouzé, Sagot, in preparation).

5 Perspectives

We have treated in this paper a case of gene finding particularly, but not solely, adapted to identifying the exon/intron structure of orphan genes, that is of genes apparently having no documented homologs in the databases. In so doing, we have described an algorithm for comparing DNA sequences containing both coding and non coding regions. The approach followed considered the coding regions as non independent from one another. A gene was thus seen as an ordered set of exons composing the pieces of an assembly. The algorithm proposed considers the possibility of sequencing errors, and therefore of frameshifts that change the reading phase of an exon. It is also completely organism-independent (except for the choice of substitution matrix).

We worked within a "one ORF, one gene" context. The "more than one ORF" context, or the more complex "one ORF, more than one gene" problem (which corresponds to the cases of alternative splicing) were not addressed. Good algorithmical (and, sometimes, better defined biological) models are needed for both. Any conceivable such model will require the capacity to recover not just a best assembly, but all best assemblies, or even, all assemblies within a neighbourhood of the optimal. This is a question we shall treat in the future.

The exclusively exonic model of a gene, that considers a gene as a set of independent exons, each exon corresponding to a "well-conserved" fragment of two homologous genes, presents some interesting aspects in relation to the "gene finding as exon assembly" approach followed in this paper. Preliminary results using such an alternative model have shown that, although it will more often predict an intron inside two exons to be of exonic nature (in particular when the intron is short or well-conserved), it has the advantage that an incorrect exon prediction stands a lesser chance of messing up with the final gene prediction. In the "gene finding as exon assembly" approach, if one exon, or simply the end of one exon, is wrongly or badly determined, everything that comes afterwards in the assembly may become wrong also.

This second model has, however, the advantage of better representing the biological problem at hand. In particular, it takes into account a greater number of biological constraints, among them probably the most important is phase compatibility, while still remaining organism-independent. We can introduce the phase constraint into the exclusively exonic model of a gene but algorithmical complexities will probably not be as encouraging as those for the algorithm presented in this paper. These issues will nevertheless be explored in the future.

The algorithm has been implemented (in C) and is called Utopia. Further applications to real data will be more thoroughly discussed elsewhere (paper in preparation).

Table 1: Enumeration of all possible combinations for $|\alpha_1|$ and $|\alpha_2|$ (where α_1, α_2 denote nucleotides at the end of an exon in an assembly built so far) given the lengths of $\tilde{x}$ and $\tilde{y}$ (currently considered nucleotides).

for $\lvert\tilde{x}\rvert \times \lvert\tilde{y}\rvert$	possible combinations of $\lvert\alpha_1\rvert \times \lvert\alpha_2\rvert$	number ab	number a, b
3x1	0x0 0x1 0x2	21	3
3x2	0x0 0x1	5	2
3x3	0x0	1	1
2x1	0x0 0x1 0x2 1x0 1x1 1x2	105	15
2x2	0x0 0x1 1x0 1x1	25	10
2x3	0x0 1x0	5	5
1x2	0x0 0x1 1x0 1x1 2x0 2x1	105	42
1x3	0x0 1x0 2x0	21	21
	total	288	99

Table 2: Enumeration of all possible combinations for $|\tilde{x}|$ and $|\tilde{y}|$ given the lengths of α_1 and α_2 that had been anticipated.

for	number of entries		possible combinations of		total	
$\|\alpha_1\| \times \|\alpha_2\|$	ab	a, b	$\|\tilde{x}\| \times \|\tilde{y}\|$	total	ab	a, b
1x1	16	4	0x0 0x1 0x2 1x0 1x1 1x2 2x0 2x1 2x2	9	144	36
0x1 1x0	4 4	1 4	0x0 0x1 0x2 1x0 1x1 1x2 2x0 2x1 2x2 3x0 3x1 3x2	16	128	80
0x2 2x0	16 16	1 16	0x0 0x1 1x0 1x1 2x0 2x1 3x0 3x1	8	256	136
total	56			total	528 + 56	252

Table 3: Detail of number of cells required for arrays e_s^{known}, $e_s^{anticipated}$, e_s^I and e_s^D where s is a, b or ab.

e_s^{known}			$e_s^{anticipated}$		
$\|\tilde{x}\| \times \|\tilde{y}\|$	s		$\|\alpha_1\| \times \|\alpha_2\|$	s	
	ab	a, b		ab	a, b
0x0	1	1			
1x0	4	4	1x0	4	4
0x1	4	1	0x1	4	1
1x1	16	4	1x1	16	4
2x0	16	16	2x0	16	16
0x2	16	1	0x2	16	1
2x1	64	16			
1x2	64	4			
Total	185	47	Total	56	26

e_s^I	e_s^D
3	3

Figure legends

Figure 1 Codon alignment [21].

Figure 2 An example of codon in phase.

Figure 3 An illustration of the divide and conquer approach to exon assembly.

Figure 4 Execution time of the algorithm (in seconds) for various lengths of sequences. Distinct sequences of same length were randomly generated for each test. The length indicated in the X-axis is the common length of the two sequences. The curve in dashed line corresponds to the execution time of the algorithm, the one in full line is the curve of the function $n^2/49000$.

Figure 5 Best assembly for each of the sequences of a gene coding for alcool deshydrogenase in various plants (EMBL ac numbers: D84240, U53701, X54106, U36586, Z24755, U63931, X04050, X02915, M59082, M36469, U65972) against a sequence coding for the same gene in another plant, barley (EMBL ac number: X12733). The matrix $\mathcal{M}$ chosen is PAM120, the frameshift penalty is 40, the cost for opening a gap is 10 and the cost for extending it also 9. The penalty for introducing an intron in one sequence but not the other is 40. The minimum intro length is 55 nucleotides.

Acknowledgements

Philippe Blayo and Marie-France Sagot were partly supported by a CAPES/COFECUB project (of type II, number 272/99) between the universities of Marne-la-Vallée and Rouen in France and of São Paulo and Campinas in Brazil, as well as by the REMAG project with the INRIA, France. Concerning the first project, they particularly wish to thank Said Abdeddaim and Thierry Lecrocq for very stimulating discussions on coding DNA comparison models and gene finding in general. The authors wish to thank Maxime Crochemore for a very careful reading of the manuscript and for support. Last but not least, we gratefully thank the anonymous referees for comments and suggestions which helped improve this paper.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410, 1990.

[2] L. Arvestad. Aligning coding DNA in the presence of frameshift errors. In Farach-Colton M, editor, *Combinatorial Pattern Matching, 1997*, volume 1264 of *Lecture Notes in Computer Science*, pages 180–190. Springer-Verlag, 1997.

[3] V. Bafna and D. H. Huson. The conserved exon method for gene finding. In P. Bourne, M. Gribskov, R. Altman, N. Jensen, D. Hope, T. Lengauer, J. Mitchell, E. Scheeff, C. Smith, S. Strande, and H. Weissig, editors, *Eighth International Conference on Intelligent Systems for Molecular Biology*, San Diego, California, USA, 2000. AAAI Press.

[4] S. Batzoglou, L. Pachter, J. P. Mesirov, B. Berger, and E. S. Lander. Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome Research*, 10:950–958, 2000.

[5] C. Burge. *Identification of genes in human genomic DNA*. Ph.D. thesis, Stanford University, Stanford, California, USA, 1997.

[6] C. Burge and S. Karlin. Prediction of complete gene structures in human genomic DNA. *J. Mol. Biol.*, 268:78–94, 1997.

[7] M. Burset and R. Guigó. Evaluation of gene structure prediction programs. *Genomics*, 34:353–367, 1996.

[8] J.-M. Claverie. Computational methods for the identification of genes in vertebrate genomic sequence. *Human Molecular Genetics*, 6:1735–1744, 1997.

[9] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*, volume 5 suppl.3, pages 345–352. Natl. Biomed. Res. Found., 1978.

[10] B. Dujon. The yeast genome project: what did we learn? *Trends Genet.*, 12:263–270, 1996.

[11] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis. Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[12] J. W. Ficket. Finding genes by computer: the state of the art. *Trends in Genetics*, 12:316–320, 1996.

[13] J. W. Fickett. Recognition of protein coding regions in DNA sequences. *Nucleic Acids Res.*, 10:5503–5518, 1982.

[14] L. Florea, G. Hartzell, Z. Zhang Z, G. M. Rubin, and W. M. Miller. A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome Res.*, 8:967–974, 1998.

[15] M. S. Gelfand. Computer prediction of the exon-intron structure of mammalian pre-mRNAs. *Nucleic Acids Res*, 18:5865–5869, 1990.

[16] M. S. Gelfand, A. Mironov, and P. A. Pevzner. Spliced alignment problem: A new approach to gene recognition. In D S. Hirschberg and E. W. Myers, editors, *Combinatorial Pattern Matching, 1996*, volume 1075 of *Lecture Notes in Computer Science*, pages 141–158. Springer-Verlag, 1996.

[17] M. S. Gelfand, A. A. Mironov, and P. A. Pevzner. Gene recognition via spliced sequence alignment. *Proc. Natl. Acad. Sci. (USA)*, 93:9061–9066, 1996.

[18] O. Gotoh. An improved algorithm for matching biological sequences. *J. Mol. Biol.*, 162:705–708, 1982.

[19] X. Guan and E. C. Uberbacher. Alignments of DNA and protein sequences containing frameshift errors. *Comp. Appl. Bio. Sci.*, 12(1):31–40, 1996.

[20] R. Guigó. Assembling genes from predicted exons in linear time with dynamic programming. *Journal of Computational Biology*, 5(4):681–702, 1998.

[21] J. Hein. An algorithm combining DNA and protein alignment. *Journal of Theoretical Biology*, 167:169–174, 1994.

[22] J. Hein and J. Stovlbaek. Combined DNA and protein alignment. *Methods in Enzymology*, 266:402–418, 1996.

[23] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices for protein blocks. *Proc. Natl. Acad. Sci. USA*, 89:10915–10919, 1992.

[24] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *ACM*, 18:341–343, 1975.

[25] Y. Hua, T. Jiang, and B. Wu. Aligning DNA sequences to minimize change in protein. In M. Farach-Colton, editor, *Combinatorial Pattern Matching, 1998*, volume 1448 of *Lecture Notes in Computer Science*, pages 221–234. Springer-Verlag, 1998.

[26] G. B. Hutchinson and M. R. Hayden. The prediction of exons through an analysis of spliceable open reading frames. *Nucleic Acids Res.*, 20:3453–3462, 1992.

[27] E. W. Myers and W. Miller. Optimal alignments in linear space. *Comput. Appl. Biosci.*, 4:11–17, 1988.

[28] N. Pavy, S. Rombauts, P. Déhais, C. Mathé, D. V. V. Ramana, P. Leroy, and P. Rouzé. Evaluation of gene prediction software using a genomic data set: Application to A*rabidopsis thaliana* sequences. *BioInformatics*, 1999. in press.

[29] C. N. S. Pedersen. personal communication.

[30] C. N. S. Pedersen, R. Lyngso, and J. Hein. Comparison of coding DNA. In M. Farach-Colton, editor, *Combinatorial Pattern Matching, 1998*, volume 1448 of *Lecture Notes in Computer Science*, pages 153–173. Springer-Verlag, 1998.

[31] J. C. W. Shepherd. Method to determine the reading frame of a protein from the purine/pyrimidine genome sequence and its possible evolutionary justification. *Proc. Natl. Acad. Sci. USA*, 78:1576–1600, 1981.

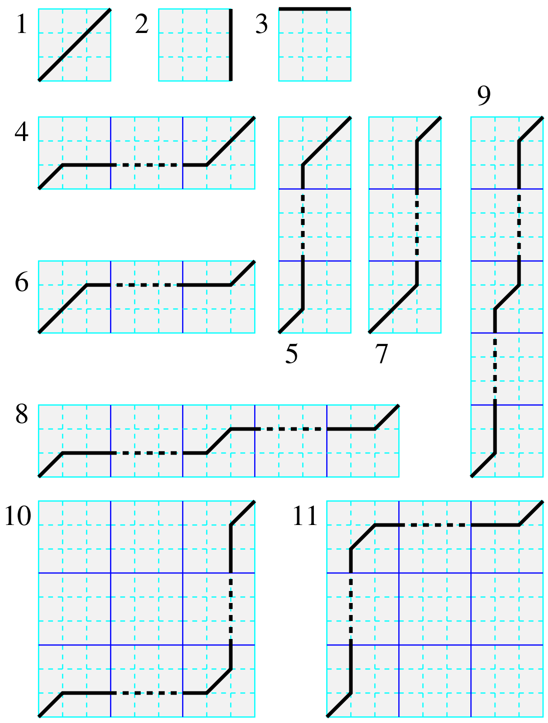
[32] R. Staden and A. D. McLachlan. Codon preference and its use in identifying protein coding regions in long DNA sequences. *Nucleic Acids Res.*, 10:5503–5518, 1982.

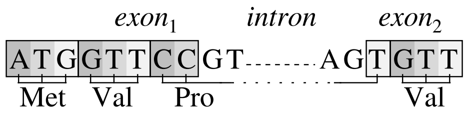
[33] N. Terryn, L. Heijnen, A. De Keyser, M. Van Asseldonck, R. De Clercq, H. Verbakel, J. Gielen, M. Zabeau, R. Villarroel, T. Jesse, P. Neyt, R. Hogers, H. Van Den Daele, W. Ardiles, C. Schueller, K. Mayer, P. Déhais, S. Rombauts, M. Van Montagu, P. Rouz, and P. Vos. Evidence for an ancient chromosomal duplication in A*rabidopsis thaliana* by sequencing and analyzing a 400-kb contig at the APETALA2 locus on chromosome 4. *FEBS Letters*, 445:237–245, 1999.

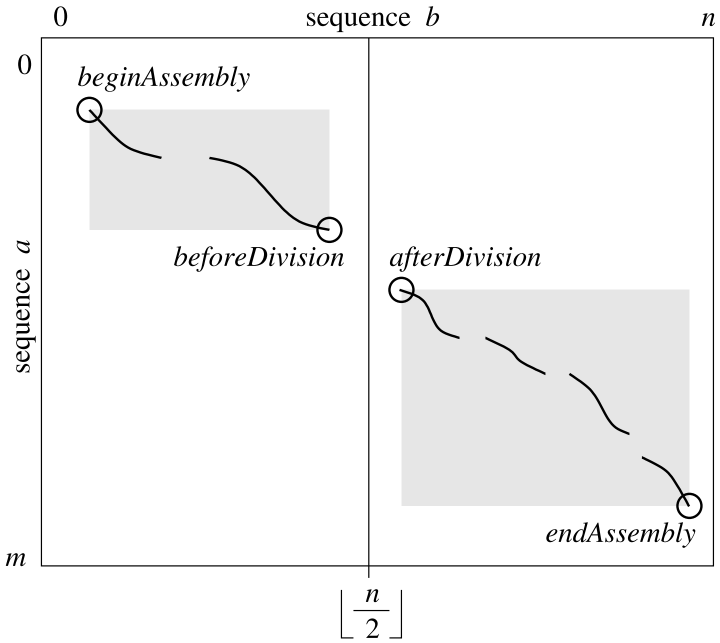
[34] A. Thomas and M. H. Skolnick. A probabilistic model for detecting coding regions in DNA sequences. *IMA J. Math. Appl. Med. Biol.*, 11:149–160, 1994.

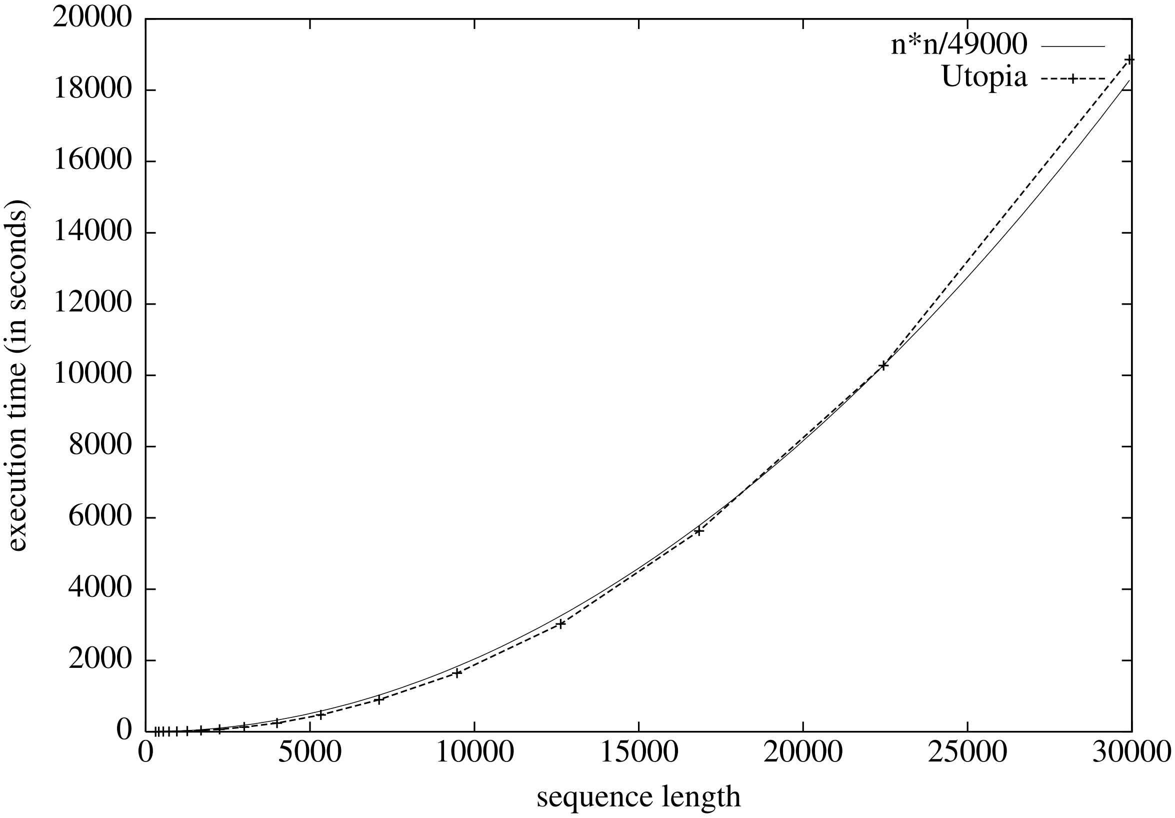
[35] T. Wiehe, S. Gebauer-Jung, T. Mitchell-Olds, and R. Guigo. SGP-1: prediction and validation of homologous genes based on sequence alignments. *Genome Research*, 11:1574–1583, 2001.

[36] Y. Xu, J. R. Einstein, R. J. Mural, M. Shah, and E. C. Uberbacher. An improved system for exon recognition and gene modeling in human DNA sequences. In *Second International Conference on Intelligent Systems for Molecular Biology*, pages 376–384. AAAAI Press, 1994.

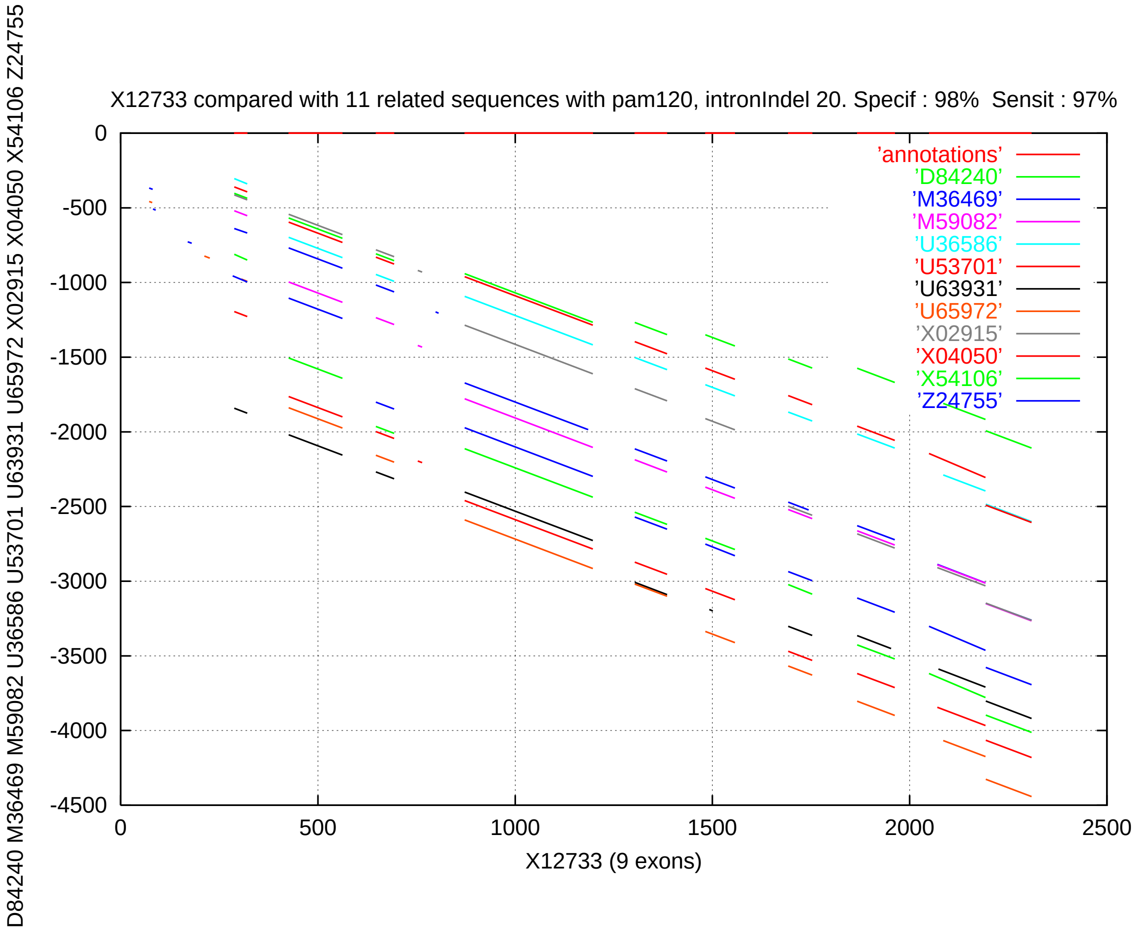

1
2
3
4
5
6
7
8
9
10
11


exon1
intron
exon2
A T G G T T C C G T . . . . A G T G T T
Met
Val
Pro
Val


0
sequence b
n
0
beginAssembly
beforeDivision
afterDivision
sequence a
endAssembly
m
⌊n/2⌋



n*n/49000
Utopia
execution time (in seconds)
sequence length
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000
5000
15000
20000
25000
30000


X12733 compared with 11 related sequences with pam120, intronIndel 20. Specif : 98% Sensit : 97%
D84240 M36469 M59082 U36586 U53701 U63931 U65972 X02915 X04050 X54106 Z24755
X12733 (9 exons)
'annotations'
'D84240'
'M36469'
'M59082'
'U36586'
'U53701'
'U63931'
'U65972'
'X02915'
'X04050'
'X54106'
'Z24755'
0
-500
-1000
-1500
-2000
-2500
-3000
-3500
-4000
-4500
0
500
1000
1500
2000
2500