To detect and analyze sequence repeats whatever be their origin

Jacques Nicolas

Chapter 4

To Detect and to Analyze Sequence Repeats whatever their Origin

Jacques Nicolas

IRISA, INRIA centre de recherche Rennes-Bretagne Atlantique, Team Symbiose Campus univ. de Beaulieu 35042 Rennes Cedex – France.

*Corresponding author: Jacques Nicolas

Running title: To detect and to analyze sequence repeats

i. Summary

The development of numerous programs for the identification of mobile elements raises the issue of the founding concepts that are shared in their design. This is necessary for at least three reasons. First, the cost of designing, developing, debugging and maintaining software could present a danger of distracting biologists from their main bioanalysis tasks that require a lot of energy. Some key concepts on exact repeats are always underlying the search for genomic repeats and we recall the most important ones. All along the chapter, we try to select practical tools that may help the design of new identification pipelines. Second, the huge increase of sequence production capacities requires to use the most efficient data structures and algorithms to scale up tools in front of the data deluge. This paper provides an up-to-date glimpse on the art of string indexing and string matching. Third, there exists a growing knowledge on the architecture of mobile elements built from literature and the analysis of results generated by these pipelines. Besides data management which has led to the discovery of new families or new elements of a family, the community has an increasing need in knowledge management tools in order to compare, to validate or simply to keep trace of mobile elements models. We end the paper with first considerations on what could help the near future of such research on models.

ii. Key words: Repeats, Seeds, String index, Pattern matching, DNA parsing, Grammatical models.

1. Introduction

The many types of repeats that occur in genomic sequences have been largely described in the literature and new types are often discovered in newly sequenced species. The management of such a quantity of families rests on dedicated databases (1) or software pipelines like REPET (see (2) and the chapter inside this volume). Besides the study of these natural repeat families, biologists are routinely concerned with sequence comparison, a task that relies on the search of words common to a given set of sequences and is almost always based on the pre-computation of a repeat index on the sequences. This issue became even more critical with the advent of Next Generation Sequencing technologies, which is leading each laboratory to access or to produce an increasing quantity of sequence data. Considering that whole genome sequencing is becoming an ordinary practice on bacteria and the analysis of the genetic diversity of eukaryotic populations by means of large scale re-sequencing projects becomes a general trend. In this context, the presence of repeats causes major assembly issues and requires further algorithmic developments.

On the theoretical side, it is important to try describing all these repeats with a set of precise common concepts in order to better understand their structure and to rationalize the design of search algorithms. Simplified generic models are used to capture important formal properties of biological repeats. Most of them are issued from problems that arose in other domains such as data compression or web indexing. Among the corresponding studies, those addressing the fundamental problem of looking for exact repeats prevail. This paper proposes a quick review of concepts at the core of any repeat model in a

sequence, mostly focusing on exact repeats. It seems clear to us that any people interested in large scale study of genomic repeats should have a good understanding of these concepts and we have tried to point all along the chapter at efficient tools that could help turning theory into practice.

2. Working on exact repeats

Exact repeats are words with several identical occurrences that are possibly overlapping. The search for approximate repeats is always based on the search for exact repeats that reflect the presence of the repeated structure and serve as anchor points during the exploration. Exact repeats have been extensively studied, starting from simple *k-grams or k-mers*, which are just words of fixed size k. A fundamental issue is to limit the number of representatives that are necessary in order to describe all "interesting" repeats. The notion of *maximality* is quite natural in this respect but not so trivial to define properly. For instance, given the string *GTTCGTTTCTTA*, the single letter *T* is repeated seven times in the string, making it the exact repeat with the maximal number of occurrences. Frequency alone is however unlikely to be useful and one has to add a criterion on the length of interesting repeats. Most of people working on sequences are familiar with statistics that help to distinguish repeats with an unexpected number of occurrences with respect to their length (see e.g. (3), (6)). Counting unexpected words is not easy because strings have combinatorial properties that have to be taken into account (see for instance section 2.3), a problem that becomes even harder in strings like genomic sequences that are structured and contained many particular patterns. The first step in any analysis is thus to be able to

explore the set of repeats without being hampered by combinatorial effects. In other terms, in genomics like in other texts, *the context* of word appearance matters and repeats are in some way made of letters appearing in a common context. In the string instance before, *GT* is a repeat made of co-occurring repeats *G* and *T* that are mutual contexts of their occurrences. In this example in fact, every occurrence of *G* is followed by an occurrence of *T*. The fact that *G* appears with the right context *T* with probability 1 renders repeat *GT* strictly more interesting than repeat *G*. The next section investigates in more details such properties.

2.1. A bestiary of simple repeats

The simplest idea of maximality is to look for each position at longest repeated substrings ending at this position (longest repeated suffixes) and present elsewhere in the string. (very useful for plagiarism detection !).. For instance in the previous string at ending position 6, GT is the longest repeated suffix (it ends at 6 and has another occurrence ending at 2). Considering the occurrence of repeat T at this position is suboptimal in the sense that a longer repeat ends at the same position, conveying potentially more information. Some authors have designed very efficient algorithms for this task (4), and the visualization software FORRepeats has been developed on this basis (5). Note that every time GT occurs in the preceding string, GTT also occurs. So “longest” means “longest up to a certain position”. In order to select only the string GTT, the concept of maximal repeat is necessary. A *maximal repeat* is a word that cannot be extended to the left or to the right without decreasing its number of occurrences in the sequence. Maximal

repeats (MR) have very nice properties since they contain longest repeats, they are never more numerous than the number of sequence characters and they contain all other repeats. Moreover these repeats can be used as basic blocks to compute error-prone repeats and they have a well defined mathematical structure of inclusion: the intersection of two MR remains a MR. As a consequence of this last property, small words are generally maximal repeats: in string GTTCGTTTCTTA, since GTT and TTC are maximal repeats, then TT is for instance also a MR. Some authors select for this reason *supermaximal* repeats, the set of MR that are not included in another MR. Note however that this does not guarantee anymore to cover the whole set of repetitions, e.g. the last occurrence of TT in our example is lost if only supermaximal repeats are retained. Another more interesting variation with respect to applications consists to fix a lower bound for the number of occurrences of a repeat. A *multi-repeat* of multiplicity k is a repeat with at least k occurrences. Maximal multi-repeats are naturally defined has multi-repeats with at least two occurrences surrounded by two different pairs of characters. Multi repeats are particularly useful in the context of multiple sequences. In this case, it may be useful to define both a multiplicity per sequence and a multiplicity over the set of sequences (quorum). In case of multiple sequences, it is sometimes possible to distinguish query sequences and target sequences. New types of maximal repeats are then relevant. *Maximal substring matches* are words present in both the query and the target with at least one occurrence in the query and one in the target surrounded by different pairs of characters. If these two occurrences are the sole occurrences of the word in the target and

the query, it is called a maximal unique match and if the word is the whole query, it is called a complete match.

Repeats in general may occur everywhere in the string, possibly at overlapping positions. Genomic sequences offer sometimes more constrained distributions. The occurrences of tandem repeats have to be consecutive in genomic sequences contrary to interspersed repeats. Technically, the term used to denote consecutive copies is *repetition*. The study of repetitions in sequences constitutes a foundation of stringology, a field concerned with the combinatorial study of strings and initiated at the beginning of the previous century by A. Thue. Sequences have subsequences that form periodic signals in the same way than numerical functions and this serves as the basis for very efficient pattern matching algorithms. Thus the sequence ACAAACAAAC has period 4 (for every position P, the character at P is the same than the character at P+4) and is a repetition made of the word ACAA. Maximal repetitions or runs may be defined in the same way than maximal repeats and corresponds to an abstract notion of tandem repeat purely generated by amplification and with no mutation: it denotes repetitions that cannot be extended to the left or to the right without losing their periodicity –in the example, AAA is a run of A starting at positions 3 and 7 but AA is not since it has period 1 like AAA-. Once again, maximality does not guarantee to get the longest runs. For instance, the sequence ACAACACAACAG starts with the run ACAACA of period 3 but contains a longer run, namely ACAACACAACA of period 5. Maximal repetitions are representative of all repetitions and contain branching tandem repeats, a concept used in Vmatch defined as squares *ww* which are not followed by the first letter of word *w*.

2.2. Suffix array: the leading data structure for tools working on repeats in a sequence

Genomics has been one of the driving forces together with web searching in the development of new index structures on sequences. Conversely, virtually all new developments for genome scale sequence analysis should be concerned with progress made in sequence data structure elaboration. Since various tasks have to be fulfilled on genomic sequences that may reach several giga base pairs, it is of utmost importance to achieve fast sequence pre-treatments that render such tasks independent of the length of the sequence. In practice, this requires the development of indexing algorithms with a linear or sub-linear behaviour in time and space. A *suffix tree* is a sequence structure made of a hierarchical dictionary of the sub-sequences starting at each position, each level of the tree corresponding to different possibilities of subsequence beginning and each leaf to a subsequence position. For instance, the suffix tree of ATATAC is tree(A(T(1,3),C(5)), C(6), T(2,4)). Suffix trees have long been the most efficient way to index sequence and allows the search of any word in a sequence in a time proportional to its length. However, since there is a big performance gap between main memory and disk storage and current computers have only several gigabytes of main memory, even the strict linear bound on algorithms may be insufficient for some usages -the most efficient suffix tree implementation requires 12 bytes per nucleic acid-.

In the current state of the art, the best structure is the *suffix array*, a simple list of starting positions in a sequence sorted according to the ascending lexicographical order of each corresponding subsequence. For instance the suffix array of sequence GTTCGTTTCTTA is [12, 4, 9, 1, 5, 11, 3, 8, 10, 2, 7, 6], in accordance with the lexicographical order of suffixes A, CGTTTCTTA, CTTA, GTTCGTTTCTTA, GTTTCTTA, TA, TCGTTTCTTA, TCTTA, TTA, TTCGTTTCTTA, TTCTTA, TTTCTTA. Note how repeats are naturally clustered in contiguous positions in the array: for instance the last four elements of the array are positions of repeats TT. Proposed in 1990 by Manber and Myers (7), the structure has been made largely available via a number of studies and implementations (8) since the work of M. Abouelhoda, S. Kurtz and E. Ohlebusch (9) showing that suffix arrays can replace suffix trees in every aspect. In practice, some additional pre-computed tables are necessary towards this aim –based on explicit recording of prefixes common to two consecutive entries in the array-, a minor change with respect to the main structure. Suffix arrays are closely related to the so-called Burrows-Wheeler transform (BWt[1]), a reversible permutation of sequence characters that tends to generate runs in the transformed sequence. This nice property has been first used to design efficient lossless data compressors such as bzip2. It is now increasingly used in genomics. We mention here an efficient tool, BWtrs (10), able to look at all runs, that is, all maximal exact tandem repeats at eukaryotic genome-wide scale. This tool improves on previous ones either by its memory consumption or the fact that it is not limited in small length repeat motif like microsatellites, nor to specific alphabet. BWtrs accepts a

[1] If SA denotes the suffix array of sequence s then BWt[i], the i^{th} letter of the BWt, is the s[SA[i]-1 mod|s|]. In our example, it corresponds to string *TTTACTTTCGTG*.

Genbank or user-defined sequence in the FASTA format and four input parameters: the minimum and maximum motif size of tandem repeats, their minimum total length and the minimum number of units (repeat ratio).

As for suffix trees, suffix array can be built with a linear time and space complexity. The best implementation available to date is probably SAIS, which has been conceived by G. Nong (SA-IS, (11)) and further enhanced by Y. Mori. It is based on a clear divide-and-conquer recursive approach and uses only 5 bytes per character. In practical applications, it may be useful to get a more complete package, including the management of auxiliary tables, allowing multiple sequences and able to make profit of multiple CPU computers via multithreading. The Bielefeld University proposes an open source program, mkESA, which answers all these constraints (12). Its output is compatible with mkvtree, another widely used implementation included in the package Vmatch ((9), see section 3).

The story of sequence data structures does not end here. Research studies are now focusing on compressing the BWT while keeping interesting average computing time. Given the current increase of main memory capacity on computers, further compression is only useful for very large strings and it is not completely clear which impact it will have an on the analysis of genomic sequences. We anticipate that the added value will come from structures taking into account basic operations needed in this context: dynamic structure adjustment for genome re-sequencing projects, management of internal mutation or insertion/deletion (indel) in repeats, management of palindromic secondary structures. For instance, T. Schnattinger et al. (13) have recently presented a promising technique tested on the search for miRNA genes, using a variation on wavelet trees allowing

bidirectional search and thus adapted to palindromic repeats. A wavelet tree is a natural companion of BWt originating from the field of data compression, coding sequences without loss through a tree of bit strings.

2.3. Introducing don't care positions in exact repeats

A striking characteristic of genomic sequences is that possible variations occur in a non uniform way along the positions. This reflects the existence of various biases such as the GC content or of many repeats, and more generally this is a consequence of the fact that genomic sequences are highly constrained sequences. Since all comparison methods are based on the search of exact repeats that are then extended (the seeds), this has important consequences on the sensitivity/specificity that can be achieved with respect to the length of the detected repeats. To get high sensitivity requires to limit the size of seeds as much as possible, while being specific requires both a lower bound size for these seeds (in practice Blast is used with a default seed length k=11 on DNA sequences) and to filter low complexity repeats from sequences before any search in order to get meaningful statistics of scores. A program like Blast is particularly confused by the existence of interspersed repeats. A number of tools have been developed using k-mer seeds as Blast does for the identification of genomic repeats (e.g. Repeatscout (14) or ReAS (15)). For large genomes, authors recommend to use k=12 and more generally, for a genome of size n, $k \cong \log_4(n)$ gives the best results.

Since a few years, a number of improvements have been carried out on Blast-like algorithms in order to choose better seeds with increased sensitivity for a same computational cost. The idea is to introduce don't care positions –jokers- along the seed (*spaced seeds*) and more generally some constraints on the allowed matches at each position (*subset seeds*). Considering simple combinatorial effects on words helps to understand the interest of spaced seeds in the reduction of dependencies. Given for instance an occurrence of word TTTT at some position, the probability to get at least another occurrence of the same word at the next three positions is 0.25, whereas it is 0.0625 for ATAT and 0 for the word ATTT. Spaced seeds and multiple spaced seeds (optimal set of seeds instead of a single one) have been used in the tool PatternHunter (16) and recent versions of Blast. The YASS software designed by L. Noe and G. Kucherov (17) is going a step further extending Blast with the use of subset seeds. YASS has proved to be useful for the pairwise aligment of mobile elements (18, 19). Seeds may be specified using a seed motif built over a three-letter alphabet #, @ and –, where # stands for a nucleotide match, – for a don't care symbol and @ for a match or a transition –A/G or C/T-, transitions taking precedence over transversions, in both coding and non-coding regions. For instance, using the pattern ##@-##, the sequence CGCCCG will be a hit on sequence ACGTACGT on position 2 since they can be aligned with a transition T/C and a substitution A/C. Note that no score is used here. The weight of a pattern, defined as the number of # plus half the number of @, is the main characteristic of seed selectivity/sensitivity, together with the seed model itself. For instance, the pattern ######### is the Blast motif used with parameter k=9, meaning that nine consecutive

characters have to be present in order to start an alignment. Its weight is 9 and its sensitivity is 0.453, estimated with respect to alignments of size 50 generated with a simple Bernouilli sequence model. If one is interested in other patterns of weight 9, of size at most 14 and containing 2 @ and 8 #, that is, allowing 2 transitions and requiring 8 matching positions, the pattern ##@-#@#--#-### can been shown to have far greater sensitivity (0.625) and to be optimal with respect to the seed model. Designing an optimal set of seeds depends on the application at hand and programs exist to generate them. Iedera (20) can be used as a pre-processing step to YASS for this purpose. It allows a number of parameters to be taken into account for seed selection and can be applied to DNA or protein sequences (21). At first sight, it seems hard to apply seed models on proteins since seeds have to be very short (w=3 for Blast) and hits have to consider the similarity of aligned sequences instead of just a match. However, subset seeds provide a good way to manage similarity without the need for score and in practice, for sufficiently long matches it is possible to improve the search sensitivity through the design of appropriate seeds. Idera has been used for instance in the software Plast, a parallel alignment search tool dedicated to the comparison of large protein banks that runs 3 to 5 times faster than the NCBI-BLAST software on this task (22).

A recent approach has further demonstrated the interest of studying seeds, introducing a new idea of adaptive seed. Among other experiments, S. Kielbasa et al. (23) show an impressive comparison of chimpanzee and human Y chromosomes. This genomic region is known to be hard to compare due to a very low information content correlated to its

repeat richness and a number of rearrangements. In 2010, J. Hughes et al. succeed in sequencing the male-specific region of the chimpanzee Y chromosome and published a surprising paper in Nature (24) claiming that >30% of its content share no alignment with the human corresponding region. This corresponds to an unexpectedly high level of divergence since this level reduces to less than 2% for the rest of the genome. Authors used ClustalW with default parameters in their analysis. S. Kielbasa et al. show by using adaptive seed based comparison that the level of divergence is in fact less than 14%. It remains a high level but this implies that their method recover more than 15% possible alignments. *Adaptive seeds* are for any target sequence and starting position in a query the shortest seeds at this position such that the matching sequence occurs at most *f* times in the target, *f* being a fixed parameter threshold. The idea is to explore a large set of seeds (the seed length is not fixed a priori) and to quickly select the most promising ones on the basis of their level of specificity. Adaptive seeds can be computed very directly using a suffix array data structure and moreover, they are compatible with spaced and fixed seeds, leading to adaptive spaced seeds and adaptive subset seeds. In the Human/Chimpanzee Y chromosome study, repeats are made of microsatellites that have to be masked (detection of tandem repeats) and other classes of repeats such has LINEs and SINEs that have to be kept in order to increase the sensitivity of the search. S. Kielbasa et al.have developed downloadable software, Last (23), and a restricted version that can be run on a web server.

We end this section with some remarks on the way to extend exact seeds to get approximate matches taking into account possible variations. Consider first the case where only mutations occur inside repeat copies. Seeds are extended to the left while the number of mutations remains acceptable and no other seed is reached. The resulting left-extended seeds are then extended to the right while the total number of mutations remains acceptable, possibly integrating other seeds. This way, all maximal matches are reported. If insertions and deletions are allowed, it is necessary to maintain a set of extension possibilities instead of a single value at each step. Note that the algorithm remains linear with the respect of sequence length or number of seeds but is cubic with respect to the number of errors. Under natural constraints on the cost of indel with respect to match/mismatch[2] and with the assumption that the total number of differences remains small, it is better to use it rather than the cost for constraining the search and the corresponding approach by a greedy algorithm becomes quadratic instead of cubic.

3. Displaying repeats

3.1. Dotplots

Various kinds of tools displaying repeats at genome level have been proposed in the past, one of the oldest and still in use being dotplots. Dotplot is a quadratic representation of repeats, crossing in a 2D array a sequence either with itself or another sequence. The possibilities of this type of representation have been greatly enhanced since a few years by integrating suffix array indexing possibilities. Gepard (25) can switch from a classical

[2] Namely Cost(Indel)= Cost(Mismatch)- Cost(Match)/2

window-based calculation for small dotplots to a suffix-array based computation looking at all repeats of a given length for large-scale dotplots, something that was out of reach of programs like Dotter. Gepard is compatible with the Vmatch package (9) and offers a command-line mode or a Java Webstart run mode. Annotated sequences can also be uploaded from the PEDANT database. Figure 1 displays a dotplot of a CRISPR (Clustered Regularly Interspaced Short Palindromic Repeats) region in the bacterial genome of *H. ochraceum*, a repeat-rich region where repeats form a skeleton of regularly spaced words flanking foreign genomic material.

3.2. Landscapes and Pygrams

Dotplots present the disadvantages of a quadratic representation: apart from the computational complexity, it entails an interpretation complexity that may hinder the understanding of structures made of complex repeat arrangements. We have proposed to take maximal repeat as a basis for the visualization of all repeats. Pygram (26) introduces the pyramid diagram, an abstract representation of the organization of repeated structures in genomic sequences. A pyramid diagram is a hierarchical representation of repeats along the sequence that makes use of the fact that intersections of MR are MR. Choosing a different color for each MR and displaying the smallest on top of the largest repeats, this property ensures that no maximal repeat will be masked by others. Technically, a Pygram for a genome sequence S is a bi-dimensional plot where S and all its maximal

repeats are mapped along the x-axis. Given an x-axis magnifying factor zx and a y-axis magnifying factor zy, mapping is defined as follows: the ith nucleotide of S is located at position (i/zx,0), and the MR of size m located at position i within S corresponds to an isosceles triangle (a pyramid) of height d m/ zy and base [i/zx, (i+ m)/zx] on the x-axis -d =1 for a MR on the normal strand or -1 for a MR on the reverse strand-. The pygrams may be considered as a rational reconstruction of landscapes -defined by B. Clift, G. Stormo et al. in 1986 and extended in 1998-, fully characterizing the structure that is displayed without requiring the computation of intermediate repeats. Pygram introduces various practical improvements over landscapes (two-strand display, zoom lenses) and offers several additional features, including frequency visualization and multigenome repeat analysis. Most important, Pygram visualization is closely associated with a query system designed to locate repeats that share specific properties. When combined, the query system and visual interface provide an efficient repeat browser that is useful for discovering unexpected structures in genomes.

Figure 2, provides the pygram of the CRISPR structure of figure1. The zoomed region not only shows the regular spacing of the repeat unit skeleton but it also points to the absence of repeat in the spacer region, a clear signature of the presence of extra genomic material.

Pygrams have been the basis of the visualization software TandemGraph that has been used for tandem repeat display along the human genome and is proposed in the database TRedD (27). Pygrams are also used in the CRISPR database Crispi (28).

3.3. The Modulome/Mobilome

The complex repeat architecture characteristic of every mobile element family is an interesting source of knowledge on the way they have evolved but is hardly depicted by the previous display modes. Two converging facts contribute to this difficulty. First, the variations inside a given family lead to a complex multiple alignment since starting from identical copies of an ancestor sequence, mutation, deletions or large insertions cumulate over time to produce very divergent copies. This is particularly observable in the case of non autonomous elements. These copies exhibit generally a modular structure where modules may appear in various orderings. Second, graphical tools on multiple alignments often consider them as multidimensional data that have to be projected into a 2D space like dotplots. For tools like VISTA (29) or GATA (30), a reference sequence has to be chosen so as to ensure a linear number of projections with respect to the number of sequences in the family. A generic software allowing beautiful and sophisticated graphics for all genomic comparison tasks has been designed by M. Krzywinski: Circos (31) uses a concentric circular layout, a practical way to display multiple information together with relationships between pairs of positions with connecting ribbons, which encode the

position, size, and orientation of related genomic elements. Circos can produce either bitmap images or high quality vector images. It offers many options and graphical representations (line, histogram and scatter plots, heat maps, connectors...) with a major concern of producing accurate, discernable and flexible display. For these reasons, it is a valuable asset in the development of new comparison tools with high quality outputs. Unfortunately, this tool is lacking a true dedicated programming language and a complete user manual butmany tutorials are available on the web and show in details the wide range of Circos display modes. Figure 4 (see also the book's cover) highlights some of the possibilities of Circos on LTR retrotransposon data. For casual users, it is highly recommended to use predefined configuration files and Perl script utilities such as tableviewer -dedicated to the graphical display of data tables-. To our knowledge, there exists to date no specific Circos environment for the study of mobile element repeats, apart from a small tool, Circoletto (32), combining Blast with Circos.

We have proposed a tool dedicated to the visualization of families of mobile elements, ModuleOrganizer (34), an extension of DomainOrganizer (33) that represents a given family of TE sequences as an assembly of elementary blocks called modules. A module is a flexible motif present in at least two sequences of a family of transposable elements and built on a succession of maximal repeats. Flexibility is founded on two simple criteria that delimit the possible spacers between consecutive repeats: the length of the spacer should not be greater than the linked part and the distance between spacers should not exceed a certain threshold. We have proposed to create such modules by a recursive

assembly method working on the set of maximal repeats present in the family sequences and added the possibility to detect palindromic modules or truncated modules. Each sequence may then be summarized by a vector of counters associated to module occurrences. The method results in a hierarchical graphical view of sequences segmented into modules, a representation that allows the exploration of transformations that have occurred between them (see Figure 3 on the analysis of a non autonomous family).

4. Perspective: modelling repeats

The common approach for the analysis of particular families is the development of dedicated pipelines assembling on-the shelf tools and tuning their parameters to get the best cover of known repeats while excluding repeats associated to other families. Among recent workflows characteristic of the variety of available programs, one can cite REPET ((2) and chapter inside this volume) and REPCLASS (35), some general pipelines for the annotation of transposable elements; DAGGPAWS (36), a workflow established to annotate transposable elements in plants or MITE-Hunter (37), a pipeline that focuses on small class II non autonomous transposable elements such as MITE. These are valuable resources that reflect an increasing knowledge on the way mobile elements are generated and evolve over time.

4.1. What is the next step?

From one side, pipelines are costly to develop and even more to maintain. Most of them use a (possibly large) parameter sets that may be hard to tune for a new data set because it is not direct to link input parameters with the desired result. Once empirical identification methods have been designed that produce good results, a high added value can be produced by people trying to rationalize them and to formalize the type of repeats that have been characterized. This can also help to solve a major issue on the fact that user have to test and compare the results of an increasing number of programs. This can also help to solve a minor issue on the fact that elements in few copy number (e.g. up to 3) are generally ignored since automatic methods have to rely on multiple occurrences of repeats for their detection.

From the other size, there exist expert centres that have acquired through large scale scans of genomes and observation of multiple instances of repeat families a good grasp of some of their major features. How to transform such expertise into scientific hypotheses that can be tested and validated on an increasing volume of genomic sequences?

We are convinced that major improvements on these questions can only be achieved by combining two types of developments: the development of generic, efficient solvers for a variety of well defined string matching problems; and the development of generic languages for modelling the complex architecture of repeats naturally occurring in genomic sequences. The rest of this section elaborates further on these two key ideas.

4.2. Vmatch, a general framework for the search of similarities in genomes

The field of word algorithmics and stringology has clearly matured since a few years in the sense that practical programs have been developed for a broader audience. It is now possible to develop efficient pipelines with reduced code acting on top of generic softwares computing well defined string selections. Vmatch (9) is a package resulting from continued efforts for years in the field of indexing and pattern matching for genomic sequences (a previous version was called Reputer) and is maintained since 2003 by S. Kurtz. It offers a flexible framework where one can proceed to a very large variety of queries. Vmatch is free for academic research and can be obtained after downloading a license agreement form. It proposes a command language whose complete possibilities are described through associated user manuals (38).

The first step in using Vmatch is to build an efficient index of the sequences to be analysed. This is the role of command mkvtree, which accepts all common sequence input formats –it even accepts gzipped files- and can be further specified with many options for the construction of auxiliary search tables or the specification of the alphabet used for sequences. The alphabet may be any user-defined alphabet not larger than 250 printable symbols. It is specified by a file storing a series of lines of equivalent letters – useful for protein sequences- and a last line of wild card characters. Another command, mkdna6idx, generates a 6 frame translation index of DNA sequences.

Once the index is built, the command vmatch allows a number of different matching tasks, including Blast-like operations. It is possible to extract almost all types of fundamental repeats that have been described in section 1: maximal repeats, supermaximal repeats, branching tandem repeats, maximal substring matches, reverse complemented, matches of length k and approximate matches with errors… Furthermore, there exists a rich palette of possible post-processing treatments on the set of solutions, either with predefined options or with user-defined C code. Basically, solutions may be sorted, filtered from sequences for masking purpose, clustered together according to pairwise similarities or positions, or chained together in order to obtain maximally covering subsets of matches.

4.3. First steps in modelling

Instead of developing brand new pipelines for an in depth treatment of each mobile element family, it is tempting to go a step further and directly describe their characteristics in a suitable language that is then compiled in order to generate an operational searching procedure. A purely structural approach has some advantages with respect to more procedural approaches. Unlike de novo repeat discovery methods, structure-based methods rely on detecting specific models of transposable elements (TE) architecture, rather than just the expected results of the transposition process (i.e. dispersed repeats with similar boundaries). Potentially, they can detect low copy number families, have high specificity to detect TE repeats and can provide a preliminary

structural classification of the newly identified TE. In contrast to homology-based methods, structure-based methods are less biased by similarity to the set of known elements.

Funding the analysis on models is a tendency that is particularly observable on well known families of mobile elements such as LTR retrotransposons (LTRR) for which a global architecture can be easily described (Figure 5). The SMaRTFinder platform for instance (39) has been developed to conduct efficient searches in DNA for structured sets of motifs, including those shared among LTRR. A structured motif is an ordered set of motifs and a list of intervals specifying the distances between motifs. In the case of LTRR elements, these motifs can be LTR end motifs, the PBS or PPT, or the DNA sequence of a highly conserved domain in an ORF. This generalized approach first starts by locating instances of individual motifs (using suffix trees in this case) and then solves a constraint satisfaction problem, by constructing a graph with motif instances as nodes and edges between nodes which satisfy order and distance constraints. Zhang and Zaki have then proposed an improvement of this method with SMOTIF (40). More recently, LTRharvest (41) incorporates in a flexible way the knowledge on LTR transposon structural features.

Using formal languages, it is possible to define at a more abstract level such structures. Formal languages are a framework introducing models as a set or rewriting rules acting on a starting axiom. The set of rules is called a *grammar*. For instance, a protein in a bacterial genome may be roughly recognized using the following grammar:

Axiom → Start Codons Stop.

Start → NotC TG

Stop → TAG| TGA | TAA

Codons→ Codon

Codons→ Codon Codons

Codon → Na Na Na

NotC-> A | G | T

Na-> A | C| G | T

In such model, the left part of a rule rewrites into its right part. Any genomic sequence that can be generated by a finite application of such rules, starting from the axiom rule, is accepted as a putative ORF by the model. Conversely, it is possible to check (i.e. to parse) a given sequence by applying the rules from right to left. Among general bioinformatics tools for genomic sequences modelling, a major contribution in this framework is due to D. Searls. He was the first to supervise developments allowing users to conceive grammars representing their biological models, and parse real genomic

sequences with them (42, 43). One of the key ideas of D. Searls is to try to find a balance between the well-founded framework of algebraic languages, a particular class of languages that offer a good expressivity/ efficiency trade-off, and the necessity to describe easily basic biological mechanisms such as copy (direct or reverse) that is at the core of genome evolution of. He has proposed to introduce in grammars a new kind of object for this purpose, the string variable, which can represent any substring and be subject to various constraints and transformations. The resulting formalism is called SVG, for String Variable Grammars. From the point of view of expressivity on biological sequences, this allows to take into account hierarchical aspects of life and presence of copies. For instance in the case of LTRR, the top level rule of the grammar could be represented by the following expression –it is given for illustration purpose only and does not pretend to be fully realistic-:

LTRR→ DR:[2..6], «tg», (U5,R,U3):[80..750], «ca »,
[1..100], pbs, [1..100], gag, [1 000..15 000], ppt, [1..100],
«tg», (U5:80%, R:90%, U3:80%), «ca», DR .

In this expression, DR, U5, R, and U3 are string variables. Its meaning is “The sequence is surrounded by two exact copies of a direct repeat (DR) of size between 2 and 6. The LTR are starting by nucleotides “tg” and ending by nucleotides “ca” and are made of three parts named A, R and B with a global length between 80 and 750. The right LTR is

an approximate copy of the left one. The central part (R) is the most preserved -because of the hybridization between both R during duplication- with a 90% minimum identity level whereas U3 and U5 need only to have 80% level identity. The central part of the sequence must contain at constrained distances a primer binding site (pbs), a group-specific antigen (gag), and poly purine tract (ppt), which are described by other grammatical rules".

The community is still lacking efficient parsers for the previous expression (Genlang, a parser developed by D. Searls has not been maintained and is no more available). Although it has not been tested on a large scale mobile elements modelling task, we mention the existence of Logol, an ongoing project developed in our team towards this goal. The Logol Software Suite – a major update of the former program Stan (44) - is a set of software composed of a Logol language interpreter and pattern search tool (LogolMatch), a graphical web-based editor, and a result analyser. It is still a beta-release and its access is provided for research purpose only. The software is designed to run on a single computer (Linux), with one or several CPU, or on a cluster. Additionally, Logol Designer is an online graphical tool to create some Logol grammar templates. It provides a drag and drop component interface to build the template. LogolMatch takes as input a sequence – DNA, RNA or protein- and a grammar file and compile them, using Vmatch and a Sicstus prolog interpreter. Result files contain the matches on the sequence(s) with all required details.

Logol is a highly descriptive language (45) dedicated to the representation and search of complex models on biological sequences. Models use constrained string variables (supporting overlaps, substitution and distance errors) that can be subject to various transformations (e.g. inverse complement), gaps, and repetitions of a pattern along the sequence, negation and alternatives to define different possibilities. As in every formal grammar components can be grouped in a view to get a high level representation of a subset of components.

To sum up, the analysis of TE is clearly facing exciting new challenges that are going beyond the routine procedure of sequence data gathering, Blast comparison, and production of descriptive statistics. We have briefly presented principles and tools that allow to efficiently compare and display sequence *structures* at genome scale. The word "structure" that usually refers to 2D or 3D spatial characteristics of proteins has now to be applied on DNA sequences to denote the issue of understanding the complex architecture of modules that make genomic families. Such structures are closely related to the family associated mechanisms and are thus providing invaluable hints for a better understanding of their role. Mobile elements seem to offer a particularly interesting field of research with respect to this structural analysis. From one side, they are generally hard to analyse using standard local comparison procedures due to the existence of embedded and degenerated structures. On the other side, there exist general transposition mechanisms that constrain the architecture of mobile elements and a growing knowledge on this

architecture built from literature and the results generated by analysis pipelines. Moreover, a same sequence generally contains ancient and recent copies of a given element and this is a unique opportunity with respect to the understanding of the conservation of structures during the evolution of sequences.

The future of bioinformatics will lie in a gradual transition from data management to knowledge management tools, that is, a transition towards more explicit models that can be confronted, refined and validated on the large base of whole genome studies. As in other contexts, the development of new tools is a matter of supply and demand. Producing and keeping a precise trace of mobile elements models, whatever the modelling language used for this purpose and even if tools are still lacking, would be a valuable contribution to the advances in this domain.

Acknowledgement

This work was supported in part by a grant from the Agence Nationale de la Recherche [project Modulome ANR-05-MMSA-0010-01].

References.

1. Jurka J, et al. (2005) Repbase Update, a database of eukaryotic repetitive elements. Cyt Gen Res. 110:462-467

2. Flutre T., et al. (2011) Considering transposable element diversification in de novo annotation approaches. PLoS ONE. 6 :1

3. Reinert G, Schbath S, Waterman M S (2005) Probabilistic and Statistical Properties of Finite Words in Finite Sequences. J Berstel and D Perrin (eds.). In Applied Combinatorics on Words. Cambridge University Press

4. Lefebvre A, Lecroq T, Alexandre J (2003) An improved algorithm for finding longest repeats with a modified factor oracle. Journal of Automata, Languages and Combinatorics 8:347-658

5. Lefebvre A, et al. (2003) FORRepeats: detects repeats on entire chromosomes and between genomes. Bioinformatics 19:319-326

6. Ussery D, Wassenaar T, Borini S (2009) Word Frequencies and Repeats. Computing for Comparative Microbial Genomics: Bioinformatics for Microbiologists. Computational Biology. s.l. : Springer. 2009, Chapters 7 and 8, pp. 111-150

7. Manber U, Myers G (1990) Suffix arrays: A new method for on-line string searches. In Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms. Ed . Edited Dana Randall, pp. 319-327

8. Puglisi SJ, Smyth WF, Turpin AH (2007) A taxonomy of suffix array construction algorithms. ACM Comput. Surv 39 :1-31

9. Abouelhoda MI, Kurtz S, Ohlebusch E (2004) Replacing suffix trees with enhanced suffix arrays. J Disc Algo 4 :53-86

10. Pokrzywa R, Polanski A(2010) BWtrs: A tool for searching for tandem repeats in DNA sequences based on the Burrows-Wheeler transform. Genomics 96:316-321

11. Nong G, Zhang S, Chan W (2009) Linear Suffix Array Construction by Almost Pure Induced-Sorting, Proceedings of 19th IEEE Data Compression Conference (IEEE DCC). Mar. 2009, Snowbird, UT, USA, pp. 193-202

12. Homann R, et al. (2009) mkESA: enhanced suffix array construction tool. Bioinformatics. 25:1084-1085

13. Schnattinger T, Ohlebusch E, Gog S (2010) Bidirectional search in a string with wavelet trees. In Proceedings of the 21st annual conference on Combinatorial pattern matching (CPM'10). Amihood Amir and Laxmi Parida (Eds.). Springer-Verlag. pp. 40-50

14. Price AL, Jones NC, Pevzner PA (2005) De novo identification of repeat families in large genomes. Proceedings of the 13th Annual International conference on Intelligent Systems for Molecular Biology (ISMB-05). Detroit, Michigan

15. Li R, et al. (2005) ReAS: Recovery of ancestral sequences for transposable elements from the unassembled reads of a whole genome shotgun. PLoS Comput 1:4

16. Li M, et al. (2004) highly sensitive and fast homology search. J Bioinform Comput Biol 2:417-439

17. Noe L, Kucherov G (2005) YASS: enhancing the sensitivity of DNA similarity search. Nucl Acids Res 33: 540-W543

18. Weber MJ (2006) Mammalian Small Nucleolar RNAs Are Mobile Genetic Elements PLoS Genet 2:e205

19. Grzebelus D, et al. (2007) Diversity and structure of PIF/Harbinger-like elements in the genome of Medicago truncatula. BMC Genomics 8:409

20. Kucherov G, Noe L, Roytberg M (2006) A unifying framework for seed sensitivity and its application to subset seeds. J. Bioinf Comp Biol 4:553-569

21. Roytberg M, et al. (2009) On Subset Seeds for Protein Alignment. IEEE/ACM Transactions on Computational Biology and Bioinformatics. 6 :483-494

22. Nguyen VH, Lavenier D (2009) PLAST: parallel local alignment search tool for database comparison BMC Bioinformatics 10:329

23. Kiełbasa SM, et al. (2011) Adaptive seeds tame genomic sequence comparison. Genome Res 21:487-493

24. Hughes JF, et al. (2010) Chimpanzee and human Y chromosomes are remarkably divergent in structure gene content. Nature 463:536-539

25. Krumsiek J, et al. (2007) A rapid and sensitive tool for creating dotplots on genome scale. Bioinformatics 23:1026-1028

26. Durand P, et al. (2006) Browsing repeats in genomes: Pygram and an application to non-coding region analysis. BMC Bioinformatics 7:477

27. Sokol D, Atagun F (2010) TRedD: A database for tandem repeats over the edit distance. Database : article ID baq003

28. Rousseau C, et al. (2009) CRISPI: a CRISPR interactive database. Bioinformatics 25:3317-3318.

29. Brudno M, et al. (2007) Multiple whole genome alignments and novel biomedical applications at the VISTA portal. Nucl Acids Res 35:W669-W674

30. Nix DA, Eisen MB (2005) GATA: a graphic alignment tool for comparative sequence analysis. BMC Bioinformatics 6:9

31. Krzywinski M, et al. (2009) Circos: an information aesthetic for comparative genomics. Gen Res 19:1639-1645

32. Darzentas N (2010) Circoletto: visualizing sequence similarity with Circos. Bioinformatics 26:2620-2621

33. Tempel S, et al. (2006) Domain organization within repeated DNA sequences: application to the study of a family of transposable elements. Bioinformatics. 22:1948-1954

34. Tempel S, et al. (2010) ModuleOrganizer: detecting modules in families of transposable elements. BMC Bioinformatics 11:474

35. Feschotte C, et al. (2009) Exploring repetitive DNA landscapes using REPCLASS, a tool that automates the classification of transposable elements in eukaryotic genomes. Gen Biol Evol 1:205-220

36. Estill JC, Bennetzen JL (2009) The DAWGPAWS pipeline for the annotation of genes and transposable elements in plant genomes. Plant Met 5:8

37. Han Y, Wessler SR (2010) MITE-Hunter: a program for discovering miniature inverted-repeat transposable elements from genomic sequences. Nucl Acids Res 38:e199

38. Kurtz S (2011) The Vmatch large scale sequence analysis software. A Manual. Unpublished report. Center for Bioinformatics Univ. of Hamburg, http://www.vmatch.de/virtman.pdf; + 2 other manuals “Chaining pairwise matches using the program chain2dim. Manual” and “Clustering Matches using the program matchcluster. Manual”

39. Morgante M, et al. (2005) A Structured motifs search. J Comput Biol. 12:1065-1082.

40. Zhang Y, Zaki MJ (2006) SMOTIF: efficient structured pattern and profile motif search. Algorithms Mol Biol 21 :1-22

41. Ellinghaus D, Kurtz S, Willhoeft U (2008) LTRharvest, an efficient and flexible software for de novo detection of LTR retrotransposons. BMC Bioinformatics 9 :18

42. Searls DB (1993) String variable grammar: a logic grammar formalism for the biological language of DNA. J Logic Program 24 :73-102

43. Searls DB (2002) The language of genes. Nature 420:211-217

44. Nicolas J et al. (2005) Suffix-tree analyser (STAN): looking for nucleotidic and peptidic patterns in chromosomes. Bioinformatics 21 :4408-4410

45. Belleannée C, Nicolas J (2007) Logol : Modelling evolving sequence families through a dedicated constrained string language. Inria Research report RR-6350 :19

Figure Legends.

Figure 1. Dotplots produced by Gepard. The right dotplot has been obtained by zooming on a repeat-rich CRISPR region, the upper left black square of the left dotplot

Figure2. Pygrams of the same genomic regions than in figure 1

Figure 3. Module organization of Foldback4 in the D. melanogaster genome

Figure 4: Distribution of LTR Retrotransposons in the human genome

Data have been extracted from the most recent version of UCSC genome tables: assembly Feb. 2009 GRCh37/hg19 for the human genome and assembly Oct. 2010 CGSC 2.1.3/panTro3 for the chimpanzee genome. The figure shows the distribution of LTR retrotransposons (LTRR) all along the human genome. Chromosomes are displayed in the outer circles, using an adapted luminance-corrected version of the UCSC genome browser color convention. Histograms are shown for the direct and reverse strand in the orange and green circles just under the circle showing cytogenetic bands. Sexual chromosomes have numerous LTRR and their list is given in the inner right circles for chromosome Y. Note that chromosome Y has been cut at two points where no LTRR occurs in the corresponding regions. Each character is coding for a particular subfamily and displayed using an associated color. For instance character 0= HERVK= {HERVK-int, HERVKC4-int, HERVK3-int, HERVK9-int, HERVK11-int, HERVK13-int, HERVK14-int, HERVK14C-int, HERVK22-int,...} is colored in black. The left inner circles are dedicated to the chimpanzee Y chromosome. This part of the figure shows results of a comparative

analysis between human and chimpanzee LTRR for the direct strand (most inner circle) of the Y chromosome. Links are colored with the same palette than linked occurrences. Intrachromosomal links are also given for the chimpanzee Y chromosome.

Figure 5. Abstract structure of a LTR Retrotransposon

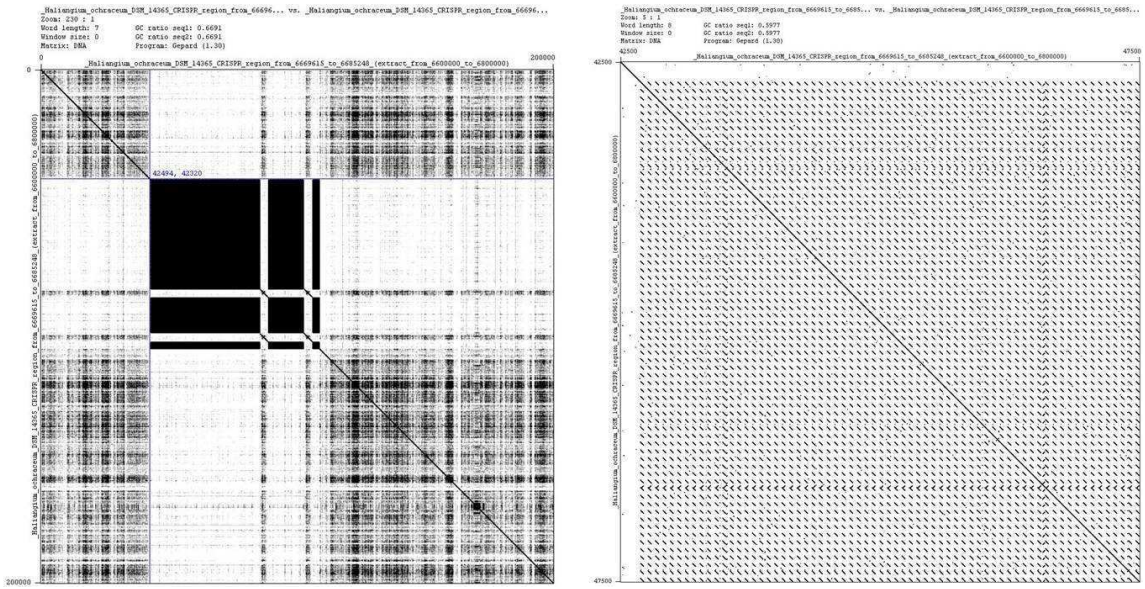

Figure 1: Dotplots produced by Gepard. The right dotplot has been obtained by zooming on a repeat-rich CRISPR region, the upper left black square of the left dotplot

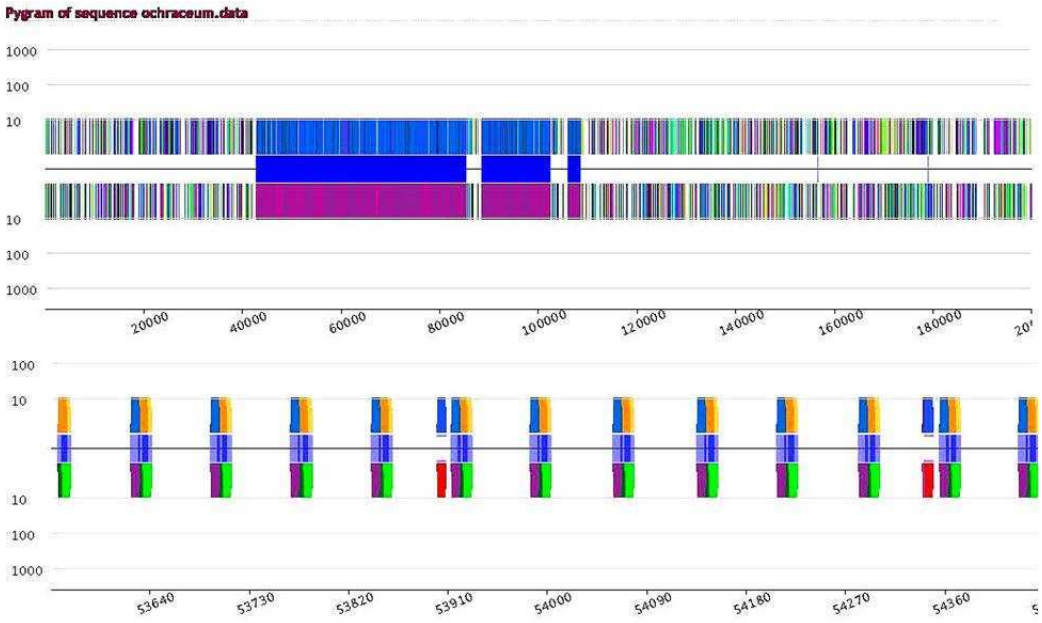

Figure 2 : Pygrams of the same genomic regions than in figure 1

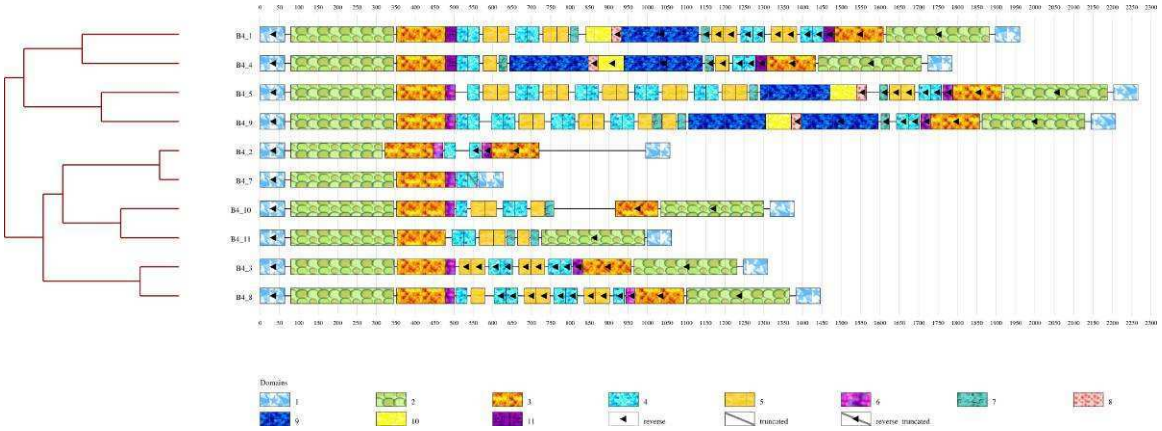

Figure 3: Module organization of Foldback4 in the *D. melanogaster* genome

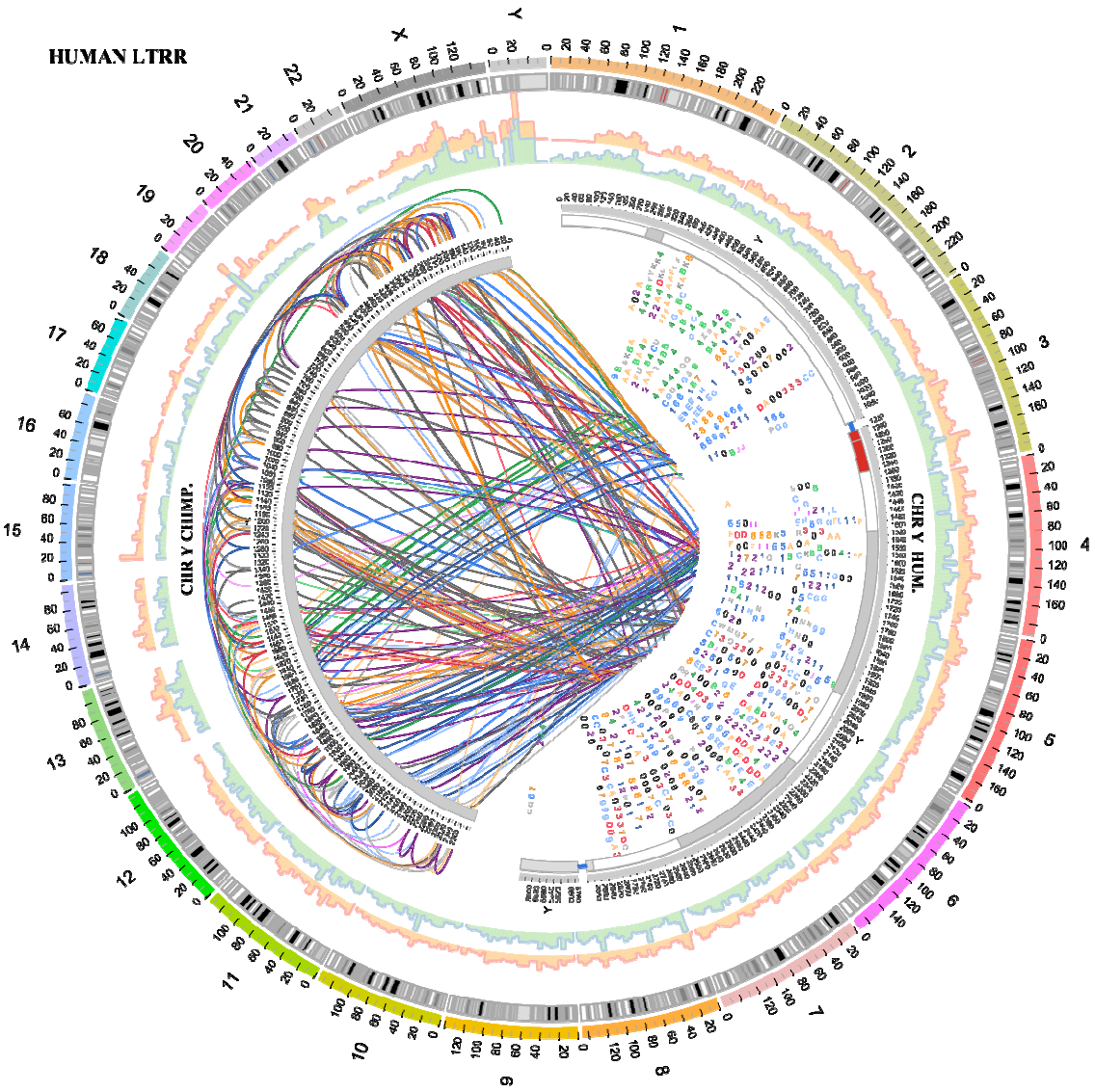

Figure 4: Distribution of LTR Retrotransposons in the human genome

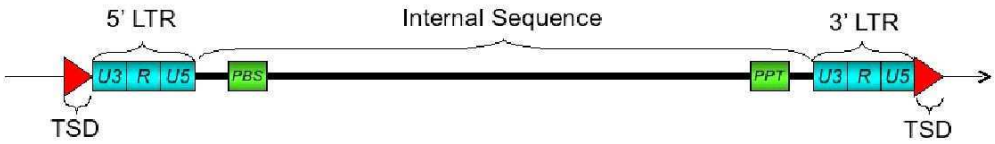

Figure 5: Abstract structure of a LTR Retrotransposon