MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Algorithmic approaches to biological sequence analysis generate new tools for the study of genome structure and function

Habilitation Thesis

MATEJ LEXA

Brno, 2022

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Algorithmic approaches to biological sequence analysis generate new tools for the study of genome structure and function

Habilitation Thesis

MATEJ LEXA

Brno, 2022

FACULTAS ARTIS INFORMATICÆ • UNIVERSITAS MASARYKIANA •

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Matej Lexa

:

Acknowledgements

I thank my coauthors for getting excited by some of the ideas and analyses presented in this thesis and in our common work. I thank my colleagues and the staff at the Faculty of Informatics for discussions and support allowing me to work on independent grant projects, travel and publish freely. I thank my wife Katarina and the close family for understanding and support throughout the many years of my scientific endeavors.

Abstract

This work provides background information and commentary to 6 research papers I co-authored. It describes my entry into and research in bioinformatics from the year 2001 until today. During this time period I explored a number of avenues in biological sequence analysis, proposed and designed new algorithms or redressed well-known algorithms to be applied in novel applications to biological data. The problems that interested me most were:

- How to search and analyze biological sequences in minimum time and space? The answer lied mostly in specialized k-mer data structures and algorithms combined with implementations relying on pointer arithmetics and bit operations.

- Can we identify and evaluate specific DNA subsequences capable of forming interesting structures at the molecular level? I found two novel approaches based on effective evaluation of nucleotide triplets and quadruplets and their mapping to known structures.

- What kind of genomic DNA sequence analysis is necessary to aid biologists in analyzing genome evolution and 3-D structure? It turned out that existing approaches often ignored repetitive regions in DNA, so I concentrated on those and designed computational tools for their annotation.

All the algorithmic ideas were swiftly implemented either in research software prototypes or in more advanced software packages made available to the research community via appropriate channels, such as repository-deposited source code accompanied by publications, downloadable GNU Linux packages made with automake or R/Bioconductor packages. They include the following algorithms or software:

1. **Virtual PCR** - a dynamic simulation model with a biological sequence mining component to predict the results of polymerase chain reactions; written in Perl and C

2. **PRIMEX** – C++/Perl implementation of a k-mer hashing-based short sequence alignment algorithm; an offline string search approach to genome analysis

3. **Triplex** – a new dynamic programming algorithm capable of predicting triplex DNA formation from sequence; implemented as an R/Bioconductor package

4. **PQSfinder** – an algorithm for exhaustive but efficient search in nucleotide sequence space to identify DNA sequence capable of forming G-guadruplex structures implemented as an R/Bioconductor package; written in C++/R

5. **TE-greedy-nester** – a greedy algorithm to identify nested transposable elements in genomic DNA sequences implemented in Python

6. **HiC-TE** – a Nextflow workflow based a novel sequencing data analysis paradigm combining existing software with components written in bash, perl, R and Python

All six commented papers were published in Bioinformatics, an authoritative journal in this area, however I also mention and cite a number of other papers I co-authored. They mostly represent the application of the developed approaches to important biological problems. Some are also additional computer science works where I was in a supporting position and the major input came from a collaborator.

Keywords

bioinformatics, sequence analysis, simulation, pattern matching, genome, repetitive sequences

Contents

Glossary

Base - a special chemical group differing between basic building blocks of DNA, commonly designated as A,C,G,T

Basepair (bp) - a term for two bases used when referring to the length of a DNA molecule or sequence; bases are typically organized in one of two pairs, A-T or C-G

(DNA) clone - a fragment of a DNA molecule that is copied and worked on in a laboratory

DNA - the molecule carrying genetic information found in cells (mostly in the nucleus, organized into chromosomes); the information it carries is coded into a sequence (or string) of bases (or basepairs); it is "read" by the cell to make RNA and protein

DNA or genome sequencing - identification of the precise order of bases/basepairs in the DNA molecules of a given sample or organism

Eukaryote - a type of organisms made of cells that have a nucleus; this group encompasses fungi, plants and animals; bacteria, on the other hand belong to a different group called prokaryotes

Genome - a collection of all genes of a given organism; however it is commonly used in a wider sense to mean all the DNA of an organism

G-quadruplex - a special structure that can be formed by DNA or RNA rich in guanines. Tetrads of guanine bases (the Gs) are formed by Hoogsteen bonding as two or more of these get stabilized by potassium ions and can thereafter form a stable structure with poorly understood/described biological functions

Hoogsteen basepair - chemically and structurally a different kind of basepair than the canonical Watson-Crick; this kind of bonding between bases is seen in triplex and quadruplex DNA

K-mer - a short substring of a string of length K

LTR retrotransposon - a special class of transposable elements that spread by a "copy-paste" mechanism via an RNA intermediate and are then reversely (retro-) transcribed back into the genomic DNA

Nucleic acid - a biochemical substance, a polymer (long string) of nucleotides; includes DNA and RNA

Protein - a polymer of amino acids; in cells the information regading what amino acids a protein should be made of is present in the form of genes in DNA molecules forming the genome; there are many different genes in a genome and many different proteins are made during the lifetime of a cell; if we were to use an analogy where the genome is the "software" that the cell carries along, then proteins would be the "hardware" of the cell

PCR (polymerase chain reaction) - a biochemical reaction used in the laboratory to synthesize a certain fragment of DNA many times over, preserving the sequence of one or a few DNA molecules that are used as a template to be copied; the copying/synthesis is chemically carried out by a DNA polymerase enzyme that has to be added to the reaction together with the building blocks of the new DNA (nucleotides) and short fragments of DNA (primers) from which the synthesis will always start and which delineate the borders of the desired fragment on the template by binding to form a duplex

RNA - a type of nucleic acid that carries information for protein synthesis around the cell as mRNA; a few specialized species of RNA carry out slightly different functions (e.g. t-RNA, rRNA); in contrast to DNA, RNA is typically single-stranded and not strictly helical

(Sequence) assembly - the process of putting together many small sequencing reads (short sequences identified by sequencing (machines)) in order to reconstruct the sequence of the whole genome that was sequenced. Most sequencing technologies are unable to read the sequence of bases of an entire DNA molecule in one reading.

Transposable element – a region of DNA in the genome that has the ability to act autonomously by moving or being copied to a different place in the genome; transposable elements are the most common cause for dispersed repeats – the phenmenon of encountering the same DNA sequence many times in different regions of the genome

Triplex DNA - DNA molecules in which a third strand is attached to the usual helical DNA duplex; intermolecular complexes of this kind are called H-DNA

1 Introduction

Bioinformatics at the crossroads of molecular biology, genomics and computer science

The last two decades of biological and medical research has been marked by ground-breaking progress enabled by the arrival of new technologies and methods in the area of molecular biology and genomics. While molecular biology as a discipline has roots in the previous century when the structure of proteins and nucleic acids was discovered, genomics is the younger sister of molecular biology, providing methods and tools to study life at molecular level using highly parallel techniques. Genomic approaches have in turn flooded the scientific arena with unprecedented volumes of raw data, mostly biological sequences and their annotation in time and space. These developments resulted in an ever increasing reliance on computers and algorithmic solutions in biomedicine, ushering in the era of bioinformatics.

Twenty years ago it were the physicists with particle accelerators and colliders and astronomers with powerful telescopes who were the hottest candidates for big data accumulation and analysis. Today, genomics, or biology and medicine are quickly becoming one of the most data-intensive scientific disciplines on the planet. Having the data available, however, is only the beginning of a long journey.

The good, the bad and the ugly

In the year 2001, the Human Genome Project yielded the long-awaited fruits of its labor, providing us with roughly 3 billion nucleotide bases, the As, C, Gs and Ts of the first reference human genome. Even with such a modest volume of new data, the task of converting this data into knowledge turned out to be a winding road with a lot of dead-ends. For example, a group of people at the forefront of medical research and human genome sequencing in the US, William A. Haseltine and Craig Venter, founded a company called Human Genome Sciences back in 1992 with a vision to build upon the sequencing of the human genome and profit on the ability to use this information to understand the underlying molecular causes behind disease and design medicine tailored to the specific mechanisms discovered in genomic data (Allen 2005, McNamee and Ledley, 2013). After almost ten years of studying the human genome, the company spent all the venture capital money, about 4 billion USD, only to come up with one candidate drug to treat lupus. Even that drug was not approved for everybody in a need of such medicine. The company was taken over by a pharmaceutical whale Glaxo-Smith-Kline before it could go bankrupt.

On the other hand, there are many success stories. They underline the fact that understanding the molecular sequences and structures of life is far from straightforward, but when done with the right tools and mindset, all kinds of basic and practical/actionable knowledge can be derived from these data. To name just a few, one can look at the scientific response to the surprising arrival of the SARS-CoV2 virus. With the ability to isolate and sequence the virus, molecular biologists and other practitioners of science and medicine were able to synthesize an artificial RNA molecule to be used as a vaccine. Later, they were able to do the same with many viral samples, compare the sequences to each other and follow the spread of different variants and lineages. None of this would be possible without necessary genomic and bioinformatic tools already in place, ready to be used at the onset of the pandemic. Another example of an enormous dataset and a resulting biological insight is a recent supercomputing exercise in petabase-scale virus discovery (Edgar et al., 2022). Perhaps the best description of this study is what the authors of the study wrote themselves in the abstract of the above Nature paper:

> *Public databases contain a planetary collection of nucleic acid sequences, but their systematic exploration has been inhibited by a lack of efficient methods for searching this corpus, which* (*at the time of writing*) *exceeds 20 petabases and is growing exponentially*[1]*. Here we developed a cloud computing infrastructure, Serratus, to enable ultra-high-throughput sequence alignment at the petabase scale. We searched 5.7 million biologically diverse samples* (*10.2 petabases*) *for the hallmark gene RNA-dependent*

[1] An account of the controversies surrounding the first RNA sequence of the virus see Campbell (2020) *Exclusive: The Chinese Scientist Who Sequenced the First COVID-19 Genome Speaks Out About the Controversies Surrounding His Work* in Aug 24, 2020 issue of TIME magazine (`https://time.com/5882918/zhang-yongzhen-interview-china-coronavirus-genome/`).

> *RNA polymerase and identified well over 105 novel RNA viruses, thereby expanding the number of known species by roughly an order of magnitude.*

Converting data to knowledge in bioinformatics

These examples illustrate that currently there are several ways to convert data into knowledge (Gagneur et al., 2017). One is to increase the output of the analysis of such data to the point that high-performance computing can find interesting patterns (as shown in the above example). If the patterns to look for are not trivial or not even known, machine learning methods can be trained in what will probably be black-box models of life and all its intricacies, nevertheless models that will most likely provide practical solutions to many problems that were difficult to address only a decade ago[2]. Another is to improve our mechanistic understanding of life at molecular level to the point that we can understand the function of most of the sequences present in past and current genomes of various species of life. Both approaches require not only more data by volume or type but also better ability to store and analyze it, and ultimately draw conclusions from them. Be it in the form of new knowledge or practical guidelines for the given moment. In what amounts to about one lifetime, we witnessed a gradual expansion of the scope of biological research to the arena of molecular biology, further to genomics, with bioinformatics as a necessary chain link that can help to make sense of the accumulated data by following some of the paths outlined above.

Contribution by the author

In my work, first as a biologist, later (since cca 2001) as a bioinformatician, I always strived to place myself at the intersection of biology and computer science. First in embracing numerical simulation models to explain phenomena in agriculture, plant nutrition and plant physiology and biochemistry. Later, with the increasing importance of molecular biology, genomics and bioinformatics, I shifted interests slightly, from analog to digital, from simulation models to biological

[2] AlphaFold protein structure prediction is one of the first better known techniques of this kind (`https://alphafold.com/`).

sequence analysis. Symbolically, my first work in this area combined a dynamic mathematical model of DNA fragment binding and synthesis in PCR (polymerase chain reaction; a commonly used technique for detection or synthesis of DNA molecules in the laboratory) with a sequence mining approach that required the adaptation and implementation of a string matching algorithm. This happened between 2001-2003 and it marked a new phase in my scientific career where I concentrated on efficient and practical ways of biological sequence analysis and the application of the resulting tools and algorithms to some pressing or interesting biological problems at the time.

My contribution to this field is demonstrated here in the 6 attached publications, each with its own chapter and commentary.

2 Included publications

Table 2.1: Author's works tabulated in the order of their appearance in the following chapters, in print, and in the Appendix as original texts.

No.	Publication	My contribution
1	**Lexa et al.** (**2001**). Virtual PCR. *Bioinformatics* **17**(2):192–193 `doi: 10.1093/bioinformatics/17.2.192`	design and implementation of simulation model (100%) data collection (50%) writing (90%)
2	**Lexa et al.** (**2003**). Primex: rapid identification of oligonucleotide matches in whole genomes *Bioinformatics* **19**(18):2486–2488 `doi: 10.1093/bioinformatics/btg350`	design and implementation of string matching algorithm (100%) writing (90%)
3	**Lexa et al.** (**2011**). A dynamic programming algorithm for identification of triplex-forming sequences *Bioinformatics* **27**(18):2510-2517 `doi: 10.1093/bioinformatics/btr439`	idea for the algorithm (80%) data and implementation (20%) writing (80%)
4	**Hon et al.** (**2017**). pqsfinder: an exhaustive and imperfection-tolerant search tool for potential quadruplex-forming sequences in R *Bioinformatics* **33**(21):3373–3379 `doi: 10.1093/bioinformatics/btx413`	idea for the algorithm (40%) data and implementation (10%) writing (80%)
5	**Lexa et al.** (**2020**). TE-greedy-nester: structure-based detection of LTR retrotransposons and their nesting *Bioinformatics* **36**(20):4991–4999 `doi: 10.1093/bioinformatics/btaa632`	design of search and classification algorithm (80%) implementation and data processing (30%) writing (80%)
6	**Lexa et al.** (**2021**). HiC-TE: a computational pipeline for Hi-C data analysis to study the role of repeat family interactions in the genome 3D organization *Bioinformatics* **38**(16):4030–4032 `doi: 10.1093/bioinformatics/btac442`	idea for the algorithm (100%) design and implementation of pipeline (50%) data analysis (80%) writing (80%)

3 Numerical simulation of the polymerase chain reaction

Lexa M., Horak J., Brzobohaty B. (2001). Virtual PCR. *Bioinformatics* **17**(2):192–193.
`doi: 10.1093/bioinformatics/17.2.192`

My contribution: design and implementation of simulation model (100%), data collection (50%), writing (90%)

WoS citations: **28**

Google Scholar citations: **64**

Towards the end of the millennium, biologists have already collected a considerable amount of sequence data. The central repository for such data was the NCBI GenBank sequence database (Benson et al., 2000). The data did not come from the massively parallel techniques used today but from many small experiments carried out in laboratories around the world (Bilofsky and Burks, 1988). Consequently, it contained a heterogeneous mix of sequence data from short clones collected and analyzed to find the DNA sequence of a fragment ranging from a single gene or transcript, to medium-sized BAC clones and other sequences. These were collected systematically for many model genome species in the process of genome sequencing, or as a cheaper alternative to assembling the full genome by simply sampling it.

The polymerase chain reaction (PCR)

I realized this data could be mined in a way that would support decision making in a number of real-world research situations. In one laboratory technique, called PCR, a fragment of DNA is synthesized by choosing a pair of short sequences upstream and downstream of the sequence of interest and then DNA polymerase enzyme is used to make new DNA molecules with identical sequence as the original fragment (template), starting alternatively from one primer or the other (Figure 3.1). At that time, it was possible to choose a good primer

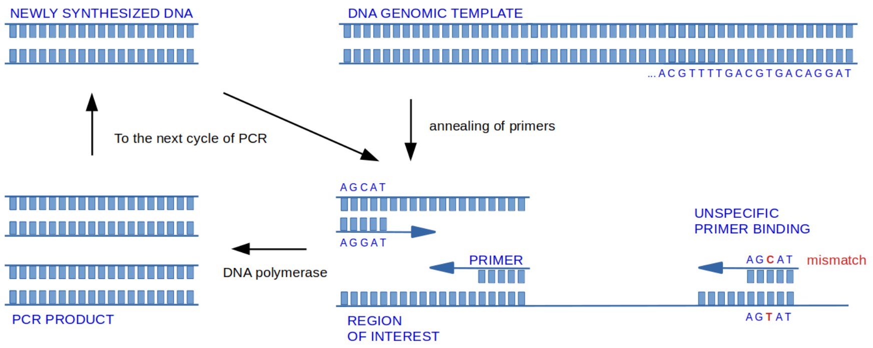


Figure 3.1: The role of primers in the PCR reaction and the need to detect unspecific primer binding.

using software that evaluated several properties of the primer DNA sequences, such as Primer3 (Rozen and Skaletsky, 2000). However, this approach did not look at one important parameter, the uniqueness of the primer binding sequence within the template, especially when amplifying parts of a genome. Often, PCR is used on material that contains the entire genome of a species, or even more than one genome (Jung et al. 1992). Also, often the primers are intentionally made less specific, leading to the so-called multiplex PCR, where the process may result in the synthesis of several different DNA products (Edwards and Gibbs, 1994). PCR is also used in a diagnostic manner, as a proof of the presence of some primer-binding sequence in the analyzed material (Grondahl et al., 1999).

The VPCR software

Compared to Primer3 (Rychlik and Roads, 1989) and similar software, I imagined an improved computational approach that would consult the NCBI GenBank database for any primer pair being tested and help the user determine what product to expect from running a PCR reaction with these primers. Such software would have two main components. Given a pair of primers and the name of the analyzed species as input, one component would find all sequences similar enough so that they could bind the primers (this part initially relied on existing BLAST software (Altschul et al., 1990)), while the second component would then run a dynamic simulation model of the PCR reaction and predict the synthesized (also known as amplified) DNA products. If the primers were not specific enough, or more than two were used as input, multiple products were predicted, showing the ability to simulate DNA products even for multiplex PCR. The software was written in C and Perl. The model was written as a system of differential equations that were solved using the Euler method (Atkinson, 1989).

At the time of publication there was no such software available to researchers, making both, the idea and the software quite popular. I succeeded receiving a two-year research fellowship from the University of Padua (Padova) to work with professor Giorgio Valle on further developing and improving this computational approach. While in Padova, I improved the simulation model to use the latest methods in melting temperature prediction for DNA duplexes (a primer bound to its template in PCR forms a primer-template duplex only at certain temperatures) as proposed by SantaLucia Jr. (1998) and further fine-tuned the simulation process. The paper describing the software published in Bioinformatics is my 4th most cited paper on Google Scholar and the 8th most cited on Web of Science. Ideas from the paper were later used in other primer design software, such as MFEprimer (Qu et al., 2009, Wang et al., 2019), Genomemasker (Anderson et al., 2005), Puns (Boutros and Okey, 2004) and FastPCR (Kalendar et al., 2017). Several research groups used the software to evaluate large sets of primers as a filtration step, to select only primers that showed promising results in the simulation. For example, Laganiere et al. (2005) wrote: *Primer pairs were designed by using the Primer3 algorithm, and the specificity was tested in silico by using a virtual PCR algorithm.*

VPCR improvements

While still at Padova University, I designed and implemented a new module of Virtual PCR later called PRIMEX (see next chapter). It replaced BLAST sequence matching to NCBI databases with full genome searches, using a new algorithm based on short fixed word (k-mer) counting and hashing, thus speeding up the prediction from tens of minutes to minutes or even seconds on a single typical contemporary computer. The speedup was much higher for large genomes, which at the time were beginning to be deposited in the NCBI database (Arabidopsis, human and mouse genomes, for example). The PRIMEX string matching software is fully described in the next chapter.

4 A k-mer based algorithm for pattern matching in an offline mode

Lexa M., Valle G. (2003). Primex: rapid identification of oligonucleotide matches in whole genomes. *Bioinformatics* **19**(18):2486–2488. doi: 10.1093/bioinformatics/btg350

My contribution: design and implementation of string matching algorithm (100%), writing (90%)

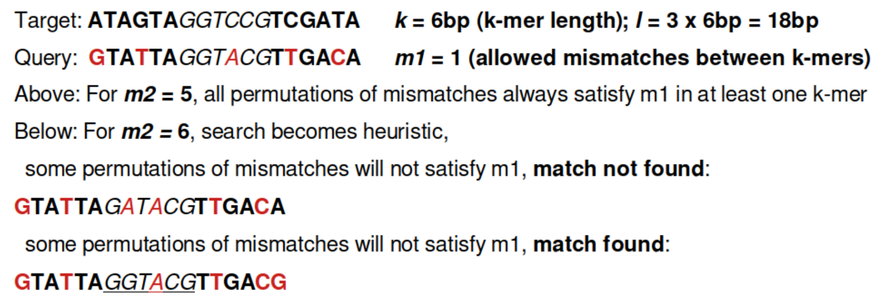
WoS citations: **23**

Google Scholar citations: **49**

As explained in the previous chapter, in 2002 I was an international fellow at the University of Padova looking for ways to improve our ability to simulate the PCR reaction products from genomic templates. Among other components, I was looking for a way to improve string matching tools for identification of short sequences in genomes. The largest genome at that time was the human reference genome with 3 billion nucleotide bases and the tool of choice for biologists was to search such sequences using a very popular software called BLAST (Altschul et al., 1990). It encapsulated a heuristic search approach using candidate sequence filtering based on short approximate hits (similarities between the search sequence and its targets). A small set of candidate hits was then explored using traditional dynamic programming algorithms for local sequence similarity to give a final answer (Galisson, 2000). While using this software as a module in Virtual PCR, I realized the approach was inefficient for repeated detection of extremely short hits in a genome or other static sequence database. I was looking for algorithms that could bring better performance to sequence comparison.

In the study of string matching (or pattern matching), problems are typically classified as one of two main approaches (modes of searching): i) online and ii) offline searching (Karp, 1992). Offline algorithms assume that the data to be searched are not available for any preprocessing and has to be searched in a streaming fashion. These algorithms are appropriate for situation where the target data change frequently or where it actually comes as a stream. When the data is relatively stable, as in the case of genome reference DNA sequences, offline algorithms are a better chosen. They allow for the creation of specialized data structures that facilitate rapid string matching. The actual searching then happens directly on these data structures and the identification of the target string matched is only done at the end of the search. Online algorithms tend to have higher space complexity and lower time complexity. Unlike offline algorithms, they are capable of searching the data in sublinear time.

A version of such offline algorithm was used at the time in software tools called BLAT (Kent, 2002) and SSAHA (Ning et al, 2001). The algorithms reduced a genomic sequence or database to non-overlapping k-mers, used these to create a lookup/hashing table and used it to quickly cover any searched string with the k-mers and their positions in the genome. In case of a match, the string was kept as a candidate and evaluated further. However the short sequences used as PCR primers made these tools ill-tuned for string matching in the context of Virtual PCR, mainly because of the use of k-mer sizes that were two big. Upon reading some *stringology* texts and also realizing that BLAST software filtering based on a pair of hits instead of a single hit performs much better, I modified the k-mer lookup/hashing algorithmic approach to work also with two approximate hits, and further tailored the implementation to allow for easy bit-encoding of bases for the envisioned k-mer length of 8-12 (BLAT and SSAHA used slightly longer k-mers). PRIMEX was written in C++ and had three key parameters to be set before use, k – the k-mer length, $m1$ - the maximal number of mismatches allowed between the query k-mer and the target in the filtration step, and $m2$ – the maximal desired number of mismatches to be reported for query strings (also l is the length of the query). Using a fixed k-mer length k allowed the use of fast pointer arithmetics and offsets in the code. Together with encoding nucleotides into two bits this provided for compact and fast string manipulation. The second parameter, $m1$ was typically 0 or 1 when used in Virtual PCR. I derived a formula for critical value of $m2$ based on k, $m1$ and l that could be used to determine whether the search for sequences was partly heuristic and therefore necessarily incomplete or whether the search guaranteed to find the complete set of matching sequences (Figure



Target: **ATAGTA*GGTCCG*TCGATA** ***k* = 6bp (k-mer length); *l* = 3 x 6bp = 18bp**

Query: **GTATTA*GGTACG*TTGACA** ***m1* = 1 (allowed mismatches between k-mers)**

Above: For ***m2* = 5**, all permutations of mismatches always satisfy m1 in at least one k-mer

Below: For ***m2* = 6**, search becomes heuristic,

some permutations of mismatches will not satisfy m1, **match not found**:

GTATTA*GATACG*TTGACA

some permutations of mismatches will not satisfy m1, **match found**:

GTATTA*GGTACG*TTGACG

Figure 4.1: PRIMEX string matching software and parameter value combination thresholds resulting in a heuristic search.

4.1). The threshold (expressed as the number of mismatches between query and target) above which the search becomes heuristic is:

$$m2_{crit} = floor(l/k) * (m1 + 1) - 1$$

The critical value of $m2_{crit}$ in Figure 4.1 for $k = 6$, $l = 18$ and $m1 = 1$ is 5.

To use PRIMEX with Virtual PCR efficiently another feature was needed. Calculating the lookup table from the genome was a time consuming process that took about 15 minutes for the human genome on a contemporary single desktop computer. I therefore introduced the possibility to save the lookup table to disk, allowing the user to skip this process when repeatedly searching the same sequence database or genome. Today, modern sequence alignment software always allows the creation of such index and saving of the index to disk (Langmead and Salzberg, 2012). However, it is quite possible that at the time, PRIMEX was the only software with such functionality.

To support the kind of real-time response Virtual PCR required, even saving and recovering the index from disk was not enough of a speed-up. A meaningful integration of the two programs for real-time use became only possible after I allowed PRIMEX to run in server mode, listening to commands via a port accessible locally or via the internet. A client software written in Perl took care of issuing commands on the user side. As a result, tens of minutes of BLAST search was reduced to about a minute using standalone PRIMEX and to seconds or fractions of a second when using the server feature (see Table 1 in the paper). One important factor in making this approach possible and successful was the availability of GB-sized computer RAM capacity. The uncompressed index (circumventing compression was important for speed) for the human genome was on the order of 4-12GB and the speed of the server mode relied on the ability to store the entire lookup table in the RAM of the server computer and consult it as needed for rapid filtering of k-mer matches.

Using the hardware at the University of Padova I installed servers for several commonly used genomes that could be accessed from any place in the world with internet access and the PRIMEX client software running (a simple Perl script using an ad-hoc data exchange protocol). The use of this functionality did not gain much traction, probably because of the inflow of new programs from genomic teams in need of short string matching in the context of next-generation sequencing (NGS). It was mostly used via my Padova website providing sequence search support for Virtual PCR simulations. These were also hosted on a webserver in Padova and at one point enjoyed dozens of users each day. The use of the server slowly decreased towards the end of the decade and I stopped updating and supporting the software around the year 2010. The source code for both, Virtual PCR and PRIMEX are available for download from my account on Research Gate (https://www.researchgate.net/publication/257650663_vpcr-21tar and https://www.researchgate.net/publication/233734306_mex-099tar).

Another interesting fact about the PRIMEX software is, that together with BLAT and SSAHA, they were essentially the first short-read aligners, a kind of software that only enjoyed a boom few years later with the oncoming revolution in DNA sequencing (so-called NGS, or next-generation sequencing). While early NGS mapping software used similar algorithmic principles (Li et al., 2008), the use of suffix trees, suffix arrays and a specialized Burrows-Wheeler transformation-based lookup data structures, such as the FM-index (Ferragina and Manzini, 2005) came into use and superseded the previously published and used software (Li and Durbin, 2009, Li et al., 2009). A Wikipedia page listing all short-read mapping software illustrates the timeline and capabilities quite well (https://en.wikipedia.org/wiki/List_of_sequence_alignment_software#Short-read_sequence_alignment).

Another area where string searches showed promise, was the ability to analyze proteins in a manner reminiscent of topic analysis and sentence meaning inference in computational linguistics (Lexa et al., 2009). Since biological sequences are long continuous sequences lacking any clear "punctuation symbols", their analysis often relied on our ability to split them into meaningful subsequences. I discovered that co-ocurrence of short sequence matches determined with PRIMEX can be used to delineate protein domains (regions of proteins carrying out a conserved function, often geometrically and structurally separated from other domains of the same protein) (Lexa and Valle, 2004). My domain identification module was part of a wider software suite of a group participating in the 2004 round of CASP and CAFASP protein structure prediction competition (Jin and Dunbrack, 2005).

The few years of popularity of PRIMEX and Virtual PCR helped me to start a fruitful collaboration upon my arrival to the Faculty of Informatics at the Masaryk University in Brno. I identified a group of scientists at the Brno Technical University that was working on hardware acceleration of various computational problems. Together with Tomas Martinek (and later Ivana Burgetova, and Jiri Hon), we started an informal bioinformatics group in Brno. We published a number of papers together, two of which (covering triplex and pqsfinder software tools) are included here and mentioned in the following chapters. On the string matching front we designed several FPGA-based architectures that could speed up PRIMEX or simpler string matching tasks up to several thousand-fold (Martinek et al., 2006, Martinek et al., 2007, Martinek and Lexa, 2008, Martinek et al., 2009, Martinek and Lexa, 2010, Martinek and Lexa, 2011). The highlight of this period was a Best Poster Award from a hardware conference (Martinek and Lexa, 2011). The design of the accelerated PRIMEX and Virtual PCR was described in a conference paper presented at the 21st European Conference on Modelling and Simulation in Prague in 2007 (Lexa et al., 2007). Hardware acceleration of string matching was a direction that emerged also at other places in the world and our resources and interests would not allow us to pursue this direction further meaningfully. In the meantime several papers and commercial products and services based on FPGAs used for bioinformatics already appeared (Arram et al., 2017) and I moved on to other problems in bioinformatics in my research.

5 A triple-stranded DNA pattern matching algorithm and its implementation

Lexa M., Martinek T., Burgetova I., Kopecek D., Brazdova M. (2011). A dynamic programming algorithm for identification of triplex-forming sequences. *Bioinformatics* **27**(18):2510-2517.
`doi: 10.1093/bioinformatics/btr439`

My contribution: idea for the algorithm (80%), data and implementation (20%), writing (80%)

WoS citations: **21**

Google Scholar citations: **33**

In the year 2006, at a workshop organized by the Biophysical Institute of the Czech Academy of Sciences, I encountered a colleague, dr. Marie Brazdova studying non-B DNA structures in DNA. These are parts of the DNA molecule that have the ability to adopt a different structural organization than the typical B-DNA helix known from textbooks. Such structures have been implicated in important biological processes, including DNA replication, cancer cell life-cycle regulation, or gene regulation. The group of Marie Brazdova in Brno was studying these structures experimentally and were looking for a way to predict their formation by sequence analysis. While at that time they were mostly interested in DNA triplexes (so-called H-DNA), they were also interested in G-quadruplexes and cruciform DNA structures. This (at the time unsolved) problem immediately raised my interest. With the knowledge of a wide range of sequence analysis methods, I recognized the triplex DNA folding problem as a variant of sequence alignment, for which a dynamic-programming algorithm has existed for many years already. With a group of students who studied the problem in their theses (Kamil Rajdl, Daniel Kopecek, Juraj Jurco) and colleagues from Brno Technical University who, perhaps also under my influence, were at that time just making a shift from hardware acceleration to bioinformatics, we started studying the problem based on the above algorithmic insight.

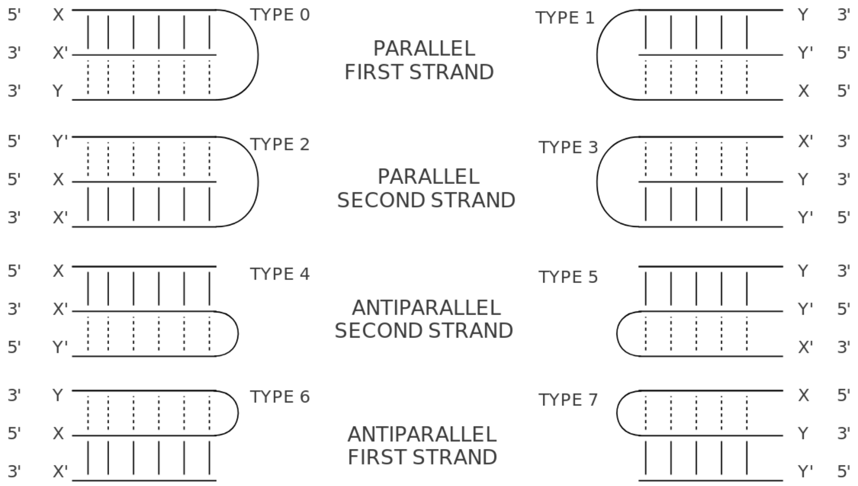


Figure 5.1: Eight types of triplexes that are detected in separate evaluations by the algorithm for a given region. Watson–Crick base pairing is shown by vertical bars, Hoogsteen base pairing typical for triplexes is shown with a dashed line. X and Y are two nucleotides on the same strand that will form a triplet. The eight possible triplets are: Y.X X, Y .XX , Y .X X, Y.XX , X.Y Y, X .YY , X .Y Y and X.YY (′ designates complementarity).

H-DNA is made of three strands of DNA, however two of these strands form the classical Watson-Crick pair (A-T or C-G) and does not require any special computational consideration. The problem was therefore reduced to a molecular palindrome search on the remaining pair of strands with special folding rules. This basic problem has already been solved by Nussinov and Jacobson (1980) and once we found a way to split the triplex DNA formation chemical rules into eight distinct categories (Figure 5.1), each of which could be solved by the prediction of palindrome formation, it became clear how H-DNA structures can be predicted.

The major obstacle was to change the basepair formation rules from classical pairing to rules applicable to triplex formation. These rules were partly known and partly we evaluated them using molecular modelling, to calculate which basepairs are capable of forming the energetically most favorable spatial arrangements and how their geometries would support each other. While working on the problem

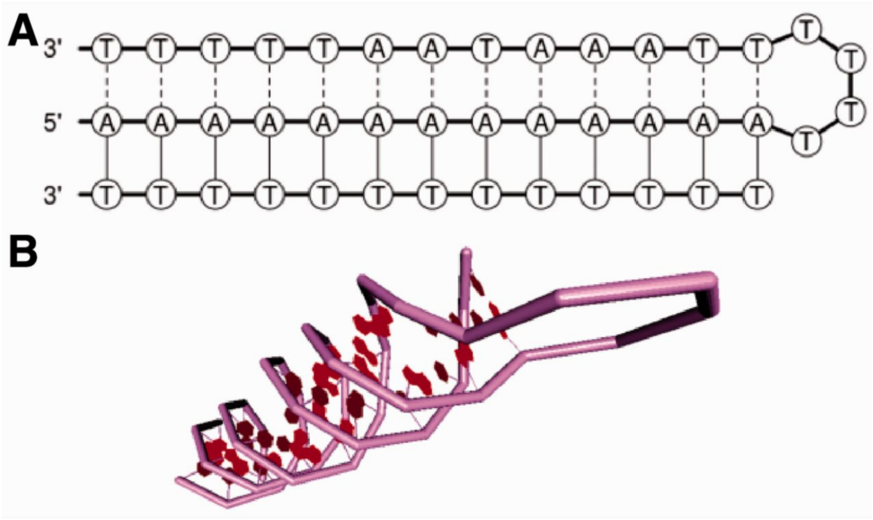

A
3' T T T T T A A T A A A T T
T T T T
5' A A A A A A A A A A A A A
3' T T T T T T T T T T T T T
B

Figure 5.2: The triplex R/Bioconductor package. (A) 2D diagram and (B) 3D model of one of the best scored triplexes.

another group from Australia published their own triplex DNA prediction program called Triplexator (Buske et al., 2012). However, it was based on a different approach and allowed us to soon publish also our own implementation of the above algorithm (Hon et al., 2013).

To motivate people to use our program, I found a bachelor student (Kamil Rajdl) at our faculty who was interested in moving our code to R/Bioconductor. At that time Bioconductor was a repository of bioinformatics software with growing popularity. He packaged our implementation into an R/Bioconductor package that allowed users to easily analyze single sequences or entire sets (Figure 5.2). Since the publication of *triplex*, more than 7000 individual IP addresses downloaded the software from the repository (`http://bioconductor.org/packages/stats/bioc/triplex/`)[1].

Later, the software was further improved with important contributions from two Faculty of Informatics students under my supervision, especially Daniel Kopecek and his thesis titled *Extension and optimization of a program for triplex detection in DNA sequences*.

[1] It should be noted, however, that these numbers include repeated downloads in separate years, possibly of new versions of the software, so that the real number of users is more likely in the higher hundreds to lower thousands.

6 A branch and bound G-quadruplex matching algorithm for sequence analysis

Hon J., Martinek T., Zendulka J., Lexa M. (2017). pqsfinder: an exhaustive and imperfection-tolerant search tool for potential quadruplex-forming sequences in R. *Bioinformatics* **33**(21):3373–3379.
`doi: 10.1093/bioinformatics/btx413`

My contribution: idea for the algorithm (40%), data and implementation (10%), writing (80%)

WoS citations: **54**

Google Scholar citations: **84**

Creating and publishing pqsfinder was a natural extension of my collaboration with the bioinformatics group at the Brno Technical University. We have been studying different non-B DNA structures for a while by then. Upon the success of triplex prediction and its application to human genomic data (Lexa et al., 2014), interest of colleagues from the Academy of Sciences in cruciform DNA, and also the increasing popularity and importance of G-quadruplexes (Yatsunyk et al., 2012), we soon started exploring the need and possibility to identify and evaluate their sequences algorithmically as well.

I spent some time evaluating cruciform DNA which resulted in a conference paper showing how sequences susceptible to their formation could be recognized. The analysis showed the importance of superhelical stress in connection with these structures. Their formation has the capacity to increase or decrease superhelicity of a circular fragment or a piece of DNA stretched between binding proteins, such as nucleosomes, scaffolding proteins, other chromatin components, transcription factors or membranes (Lexa et al., 2012).

However, our main research direction was to be the prediction of G-quadruplex formation. At the time we encountered this problem, most researchers were combing DNA sequences using regular expressions. G-quadruplexes form in loci with several guanines clustered together and a regular expression of the form $G_3 - N_{1-7} - G_3 - N_{1-7} - G_3 -$

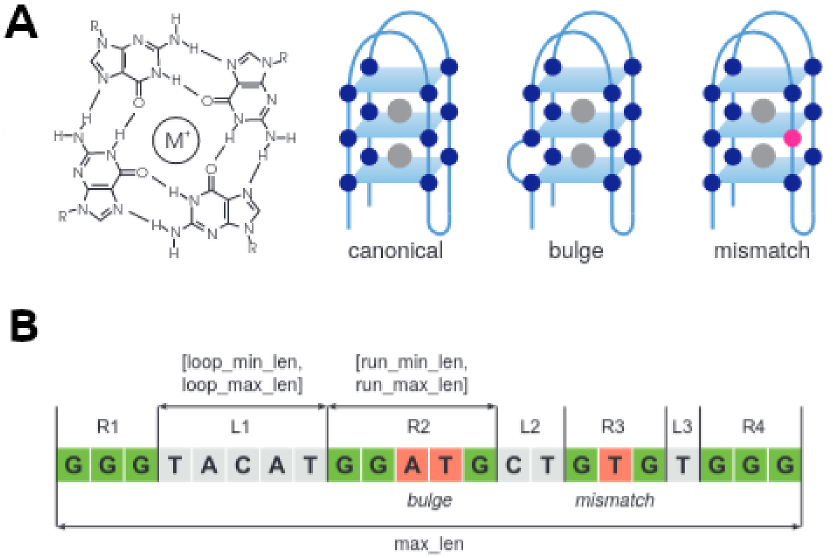
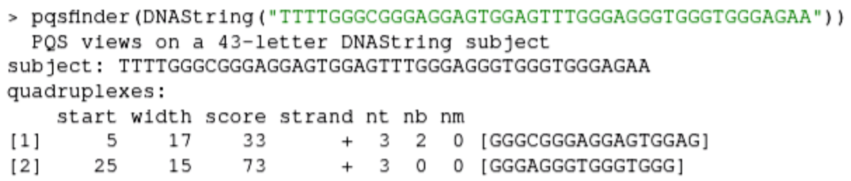


Figure 6.1: (A) Structural aspects of G-quadruplexes (G4s) – a planar tetrad of guanines; several tetrads can form a canonical G4, imperfect structures are possible; (B) – algorithmically important search parameters.

$N_{1-7} - G_3$ (where N is any of the four nucleotides found in DNA) would find the most typical sequences capable of G-quadruplex formation. Experimental evidence in literature pointed towards the possibility that other sequences were able to form the same structure, mostly by tolerating an imperfection or two (Mukundan and Phan, 2013), or by folding differently in space (Figure 6.1A). While some groups addressed this problem successfully by ignoring the precise mechanism of folding (Bedrat et al., 2016), we felt that a branch and bound algorithm evaluating all possible ways to form a G-quadruplex might be feasible[1]. Jiri Hon, a collaborator of mine was most instrumental in finding the best recursion to use in the implementation that can visit all interesting combinations of guanines (G), however, for the sake of efficiency will stop that particular evaluation at a point where the prospective G-quadruplex can not result in a better evaluation score than another that has already been identified in the same region (Figure 6.1B). Once the algorithm was fully designed and implemented, I realized we could use the results of a freshly introduced sequencing technique, called G4-seq (Chambers et al., 2015) to fine-tune and validate our program. With the help of another colleague, Tomas Martinek, using genetic programming as an optimization tool on this data, we trained the model and identified a combination of scoring parameters that gave excellent results. Not only were we able to predict G4-formation from sequence data with high accuracy, we could do so better than any of the five or so software tools used at that time. The accuracy of the various software tools and pqsfinder was evaluated on a dataset used also by the other research groups in this area, called Lit392 (Bedrat et al., 2015). It is a compilation of G4-forming sequences from literature that are supported by experimental evidence. The set also contains negative examples and therefore lends itself for this kind of use.

Upon our partial success with triplex and the growing popularity of R in bioinformatics circles, Jiri Hon steered the implementation of pqsfinder from the beginning to become an R/Bioconductor package (Figure 6.2), although the core of the package is written in C++. Soon it became very popular. It is by far my best-cited paper. The R

[1] Interestingly, B.Subramanian (a well-known researcher in G4 studies) once remarked at a conference that such evaluation would be computationally too complex, when at the time we already had first working implementation.



```
> pqsfinder(DNAString("TTTTGGGCGGGAGGAGTGGAGTTTGGGAGGGTGGGTGGGAGAA"))
  PQS views on a 43-letter DNAString subject
subject: TTTTGGGCGGGAGGAGTGGAGTTTGGGAGGGTGGGTGGGAGAA
quadruplexes:
    start width score strand nt nb nm
[1]     5    17    33      +  3  2  0 [GGGCGGGAGGAGTGGAG]
[2]    25    15    73      +  3  0  0 [GGGAGGGTGGGTGGG]
```

Figure 6.2: A typical use of pqsfinder to search for DNA sequences.

package has been downloaded so far by more than 10000 independent IP addresses (`https://bioconductor.org/packages/stats/bioc/pqsfinder/`)[2]. A recent review of software for identifying potential G-quadruplexes confirmed our own accuracy results (Lombardi and Londono-Vallejo, 2020) and placed pqsfinder among the most accurate. Recently, two new software products with the same motivation appeared, both based on machine learning approaches – Quadparser (Sahakyan et al., 2017) and PENGUINN (Klimentova et al., 2020). While there are indications that PENGUINN produces less false positives and has higher accuracy, the black-box aspect of the machine learning approaches keeps our approach with individually configurable imperfections and their scoring still attractive.

Also, pqsfinder has one extra feature not seen in other similar software. Its scoring function is extensible. In the manual to the software (Hon et al, 2017) we show how this feature can be used to evaluate a different class of G-quadruplexes than the ones envisaged at inception. Only a small change in code is required to provide a new type of scoring that can, for example, evaluate interstrand G-quadruplexes, where half of the guanines come from another strand of DNA, while the sequence (the one that is analyzed) has cytosines in the regions contributing to the quadruplex (Kudlicki, 2016).

My work on non-B DNA structure prediction, especially G-quadruplexes described in this chapter, led me to another collaboration with a research group at the Biophysical Institute of the Czech Academy of Sciences in Brno. In an ad-hoc and informal research collaboration on repetitive sequences in plants, together with my now long-term collaborator, dr. Eduard Kejnovsky who studies plant repeats (many of

[2] It should be noted, however, that these numbers include repeated downloads in separate years, possibly of new versions of the software, so that the real number of users is more likely in the higher hundreds to lower thousands.

which are mobile/transposable elements), we hypothesized that non-B DNA might play some role in their life cycle. Having just finished the work on pqsfinder, I created a collection of tens of thousands of plant transposable elements and analyzed them for potential G-quadruplex forming sequences (PQS). To our surprise, there were certain areas in these DNA sequences that had much higher probability of containing a G-quadruplex then other regions (Lexa et al., 2014). They were the regulatory regions and this finding resonated well with findings in many other organisms, where G-quadruplexes were found to be enriched in gene promoters, another regulatory region in the genome. We were able to confirm similar situation in the human genome (Lexa et al., 2014b) and lately another research group observed the same results in the bovine genome (Stefos et al., 2022), so the observation, which is not currently a well-known fact, seems to hold for most eukaryotic genomes in nature by now.

Much of our later research focused on pinpointing the possible roles these G-quadruplexes could play (Kejnovsky et al., 2015). We formulated a hypothesis that transposable elements, because of their mobility in genomes, are predisposed as ideal candidates for "genomic vehicles" in eukaryotes that would populate different areas of the genome with G-quadruplexes (Kejnovsky and Lexa, 2014). These could later appear as G-quadruplex enriched gene-regulatory sequences, such as promoters or enhancers. For example the group of Stefos et al. (2022) studying G-quadruplexes in the bovine genome found supporting data for the function of LINE trasnposable element as such vehicle. This function of transposable elements, in my opinion, is currently underestimated and most of the research worldwide mainly focuses on specific examples of other two functional classes. One is encountered where transposable elements were exapted (captured in a specific locus of the genome) to aid or otherwise modify the expression or function of a specific gene (Lopez and Bureau, 2018). Another is seen where the machinery of the host which evolved to suppress potentially dangerous repeat expansion inhibits (or silences) not only the transposable element but also some sequences nearby (in case of methylation), or genome-wide (in case of small RNA production).

Together with Eduard Kejnovsky we managed to form a small research group funded by the Czech Grant Agency and occasionally other grants as well, which now focuses on this line of research. In our transposable element studies, I recognized several areas that needed fresh bioinformatics research. These lines of research are described in detail in the following two chapters.

7 A greedy algorithm for detection of nested structures in genomic DNA sequences

Lexa M., Jedlicka P., Vanat I., Cervenansky M., Kejnovsky E. (2020). TE-greedy-nester: structure-based detection of LTR retrotransposons and their nesting. *Bioinformatics* **36**(20):4991–4999.
`doi: 10.1093/bioinformatics/btaa632`

My contribution: design of search and classification algorithm (80%), implementation and data processing (30%), writing (80%)

WoS citations: **6**

Google Scholar citations: **8**

Building on the established collaboration in eukaryotic transposable element research described in the previous chapter, we came across an area in genome evolution and transposable element analysis that was not well covered by suitable software tools. To a certain extent there even was no clearly described and suitable algorithm for that particular sequence analysis. When studying plant genomes, we noticed that often what looks like isolated fragments of transposable elements in the genome, are actually fragments of the same element pushed apart by a later insertion of a younger transposable element into the middle of the fragmented one. This kind of arrangement reminds us a bit of the Russian *matrioshka* dolls and is commonly designated as *nested*. When studying transposable elements, one often is interested in the evolution of the genome and how individual elements were moved or copied in time or in the past. We desperately needed a tool to identify the patterns of nesting, including the order and family membership of the individual elements. While some software existed at the time, there were some downsides to its use. The best of them, TE-nest (Kronmiller and Wise, 2008) suffered from very long computation times and was therefore not suitable for inspection of entire genomes.

After brief experiments with the idea involving a student who used the subject for their thesis (Radovan Lapar), I laid down a blueprint for a reasonable algorithmic solution. The following describes how

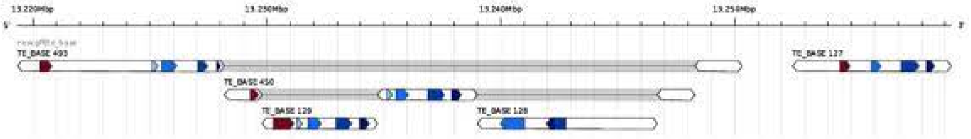


Figure 7.1: Nested structure of LTR retrotransposon insertion in the genome discovered and reconstructed by the TE-greedy-nester software.

the basic algorithm works. A greedy (not necessarily remaining such in the future, but works satisfactorily enough at the moment) iterative algorithm finds the most clearly identifiable full-length transposable elements in the analyzed DNA sequence and removes the nucleotides that are recognized as being part of the element. This, in turn may put together a previously fragmented element which now becomes recognizable as a full-length element and can be removed from the sequence by another iteration of essentially the same calculation (for a detailed description of the algorithm, please, see inset on page 2 and Figure 1 in the paper). We quickly showed that this procedure had promise, however a series of experiments was needed to fine-tune the algorithmic steps and find the best values for a number of parameters. In the end we decided to use a graph-based encoding for the evaluation of transposable element quality. Only elements that score well against this graph are removed as full-length elements. Another trick was to slowly lower the strictness of element identification, since the older sequences were often less conserved and removing imprecisely younger sequences from them also brought some noise into the formula. When no more transposable elements can be extracted from the analyzed sequence, the software backtracks its steps to recover the coordinates of the identified elements and outputs an annotation file in the GFF3 format that can be visualized with common genome browsing and visualization tools (Figure 7). The software was called TE-greedy-nester, with important contributions from two MU Faculty of Informatics students, Radovan Lapar, Michal Cervenansky, Jakub Horvath and Ivan Vanat. Ivan Vanat is currently getting ready to defend his master thesis bringing a number of further improvements to this procedure.

8 Genome annotation workflow implemented as a Nextflow pipeline

Lexa M., Cechova M., Nguyen S.H., Jedlicka P., Tokan V., Kubat Z., Hobza R., Kejnovsky E. (2022). HiC-TE: a computational pipeline for Hi-C data analysis to study the role of repeat family interactions in the genome 3D organization. *Bioinformatics* **38**(16):4030–4032.
`doi: 10.1093/bioinformatics/btac442`

My contribution: idea for the algorithm (100%), design and implementation of pipeline (50%), data analysis (80%), writing (80%)

WoS citations: **0**

Google Scholar citations: **1**

In the past, molecular biology mostly studied genomes as linear arrangements of genes subject to regulatory influences among them. This was the legacy of discoveries made in studies of bacteria where linear gene structure was key to many activities of bacterial cells. However, in eukaryotes, it turns out, cellular (especially gene-regulatory) processes are more complicated (Lelli et al., 2012). Much of the regulatory properties are not encoded in the linear arrangement of genomic elements but their ability to contact each other in 3D.

The latest advances in genomics allow us to ask questions about spatial arrangement of genes and other genomic DNA sequences in the nucleus of the cell (Fraser et al., 2015). In short, fragments of DNA in close proximity are chemically fixed, joined together by bonds and sequenced in a paired mode in such a way that each sequencing read in the pair contains one side of the original contact. While the core bioinformatic support for the necessary calculations has been introduced in parallel with the experimental methods, such as 3C, 4C and HiC (Fraser et al., 2015), the methods are mostly applicable to the non-repetitive parts of the genome. Because of my long-term interest in the repetitive fraction of the genome and bioinformatic methods that can help understand the repeats and their evolution within plant genomes, I started looking at ways to use the spatial HiC data for characterization of repetitive sequences in the 3D context.

It occurred to me that where traditional HiC analysis binning procedures group HiC reads by their location (to increase the signal/noise ratio), we could try binning by repeat family classification or membership. This idea grew into a procedure that could show the most interacting repeat families in heatmaps. Together with a Faculty of Informatics student Son Hoang Nguyen, a new postdoctoral associate Monika Cechova and the transposon group of Eduard Kejnovsky, we organized the main scripts into a robust pipeline based on the increasingly popular Nextflow workflow language. The components of the pipeline include Perl and Python string manipulation scripts, bash glue scripts and a number of R scripts for generating visualizations in the form of heatmaps and circular plots, among other types.

While most of the pipeline relies on existing software and tested visualization R packages such as circlize, karyoplotteR and ggplot, a lot of effort went into the proper normalization of counted repeat contacts. Without the normalization it would be impossible to tell which number of contacts are common/expected and which signal some kind of statistical anomaly that could represent a biologically significant phenomenon. A short description of the normalization procedures follows.

The HiC-TE pipeline accepts paired HiC sequencing data as input. After we count all valid HiC pairs in the pipeline a table is created that contains family names in two columns (family1, family2) and in cases based on the reference genome also mapped positions (pos1, pos2). The number of combinations observed between positions and repeat families contains technical and methodological biases. For example there are many more pairs observed for adjacent positions on the same chromosomes compared to long-distance or interchromosomal HiC pairs. Some kind of normalization is therefore necessary before reporting basic statistics or creating heatmap visualizations. Choosing the right normalization method is far from trivial (Sauria et al., 2015). After careful consideration, we chose three different methods that we use in parallel towards the end of calculations in the pipeline when a family by family matrix underlying each heatmap is calculated. The three normalization methods are:

Joint probability The baseline probability of observing a HiC pair between family A and family B is estimated as joint probability of individual probabilities for observing family A or family B in a given HiC read.

$$p(A, B) = p(A).p(B)$$

All counts of A-B pairs are divided by this number.

Label permutation Family names in the table in columns family1 and family2 are randomly assigned to random (and therefore possibly different) rows of the table. A matrix is also created from this altered table. All counts of A-B pairs are divided by corresponding values in this matrix.

Annotation shuffling While creating the HiC pair table, a parallel table is made, which uses chromosomal positions randomly shuffled along the reference genome, while their size distribution is preserved. The pair count matrix constructed from such table is used for normalization as above (see label permutation).

As output, the pipeline generates a set of tables and images. The most informative images are the heatmaps showing normalized contacts with values differing from 1 (Figure 8).

.

While the pipeline will happily run on any HiC data and the corresponding reference genome, there are some limitation when running the vanilla gitlab version in such manner. Repeat classification done by Repeat Explorer (Novak et al., 2010), TE-greedy-nester and inner BLAST annotation scripts are plant-oriented, using the Neumann et al. (2019) plant TE classification scheme. TE-greedy-nester enriches the annotations for LTR retrotransposons. However, other repeat annotations may be preferred for other organisms. To make the analysis more meaningful for animal species where LTR-retrotransposons are not the main category of repeats, or to provide annotation of additional repeat classes, compared to only LTR-retrotransposons annotated by TE-greedy-nester, we allow the main reference-based repeat annotation to be provided in a GFF3 file. The pipeline is specifically tuned to accept a combination of *.out and *.gff files from Re-

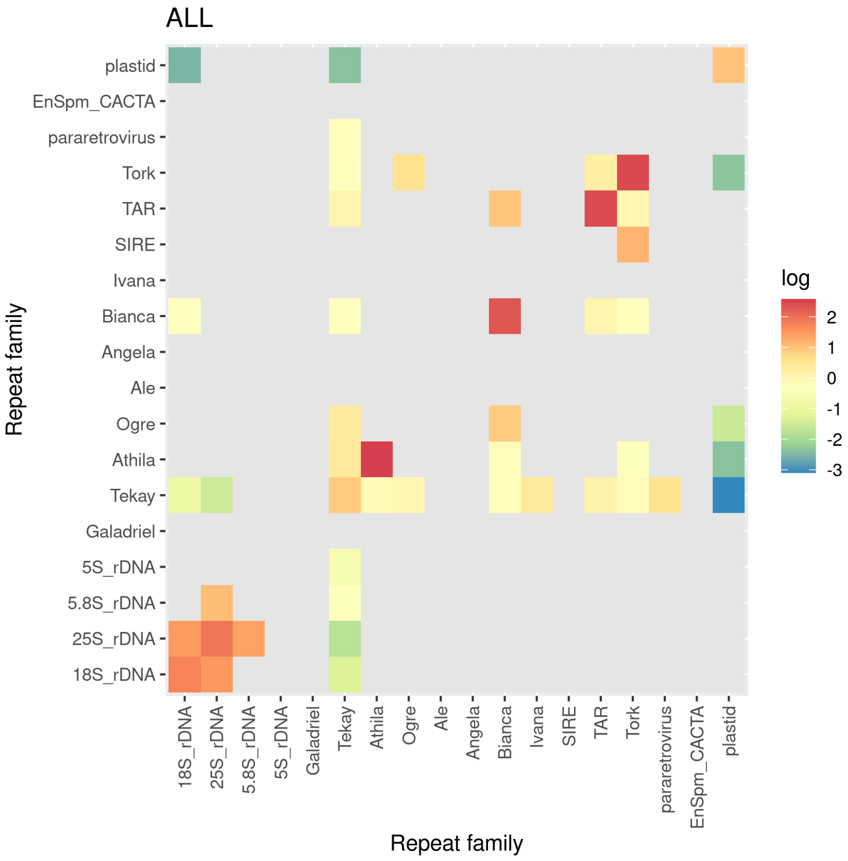


Figure 8.1: Heatmaps are the main output of HiC-TE allowing the user to see whether certain families deviate from the normalization expectations towards more contacts (red), or less contacts (blue) than expected.

peatMasker, but can be adapted to other external sources of annotation. The main requirement is for the GFF3 file to contain an annot="repeat_family" variable and for the corresponding Perl script (here enrich_rmsk_gff3_annotation.pl) to be able to add that name from available output.

HiC-TE pipeline is my first endeavor into the area of reproducible and scalable workflows. While back in the year 2001 it was considered satisfactory to provide a running binary or installable source code for a program to be accepted and used in biological research, nowadays the growing data volumes and increasing team work in all areas of science call for formal workflow definitions, transparent code and the use of robust frameworks that will continue working in many different use cases. HiC-TE was also our first paper published initially as a preprint in biorXiv. The experience was a positive one, with the pipeline earning its first citation while still in the preprint stage.

9 Conclusions

The papers discussed in this text show the importance of algorithmic approaches and data wrangling in contemporary biological research. In many of my papers, we considered a solution to a problem that others deemed impractical or even impossible to solve or improve on. Having the knowledge, access to prior research and perhaps also a bit of luck to identify a new algorithm, an old algorithm that has not been adapted to a specific class of biological data or a need among biologists that has not been met – having all this I sincerely hope I pushed our ability to analyze biological data forward. If along the way I inspired a couple of collaborators and students, I consider myself lucky, indeed.

10 References

1. Allen, A. (2005). The Disappointment Gene. Why genetics is so far a boondoggle. `https://slate.com/technology/2005/10/why-genetics-is-so-far-a-boondoggle.html` Visited on Oct 22, 2022.

2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J. (1990). Basic local alignment search tool. *Journal of Molecular Biology* 215:403-410.

3. Andreson, R., Reppo, E., Kaplinski, L. et al. GENOMEMASKER package for designing unique genomic PCR primers. *BMC Bioinformatics* 7, 172 (2006). https://doi.org/10.1186/1471-2105-7-172.

4. Arram, J., Kaplan, T., Luk, W., Jiang, P. (2017). Leveraging FPGAs for Accelerating Short Read Alignment. *IEEE/ACM Transactions in Computational Biology and Bioinformatics* 14(3):668-677.

5. Atkinson, K.A. (1989). *An Introduction to Numerical Analysis* (2nd ed.). New York: John Wiley & Sons. 693 pp.

6. Bedrat, A., Lacroix, L., Mergny, J.-L. (2016). Re-evaluation of G-quadruplex propensity with G4Hunter. *Nucleic Acids Research* 44(4):1746-1759.

7. Benson, D.A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Rapp, B.A., Wheeler, D.L. (2000). GenBank. *Nucleic Acids Research* 28(1):15–18.

8. Bilofsky, H.S. and Burks, C. (1988). The GenBank ® genetic sequence data bank. *Nucleic Acids Research* 16(5):1861–1863.

9. Boutros, P.C. and Okey, A.B. (2004). PUNS: transcriptomic- and genomic-in silico PCR for enhanced primer design. *Bioinformatics* 20(15):2399-2400.

10. Buske, F.A., Bauer, D.C., Mattick, J.S. and Bailey, T.L. (2012). Triplexator: Detecting nucleic acid triple helices in genomic and transcriptomic data. *Genome Research* 22:1372–1381.

11. Chambers, V.S., Marsico, G., Boutell, J.M., Di Antonio, M., Smith, G.P., Balasubramanian, S. (2015). High-throughput sequencing of DNA G-quadruplex structures in the human genome. *Nature Biotechnology* 33(8):877-81.

12. Cheeseman, J.M., Barreiro, R., Lexa, M. (1996). Plant growth modelling and the integration of shoot and root activities without communicating messengers: Opinion. *Plant and Soil* 185(1):51-64.

13. Edgar, R.C., Taylor, J., Lin, V. et al. (2022). Petabase-scale sequence alignment catalyses viral discovery. *Nature* 602, 142–147.

14. Edwards, M.C., Gibbs, R.A. (1994). Multiplex PCR: advantages, development, and applications. *PCR Methods and Applications* 3(4):S65-75.

15. Ferragina, P. and Manzini, G. (2005). Indexing Compressed Text. *Journal of the ACM*, 52(4):553.

16. Fraser, J., Williamson, I., Bickmore, W.A., Dostie, J. (2015). An Overview of Genome Organization and How We Got There: from FISH to Hi-C. *Microbiology and Molecular Biology Reviews* 79(3):347-72.

17. Gagneur, J., Friedel, C., Heun, V. et al. (2017). Bioinformatics advances biology and medicine by turning big data troves into knowledge. *Informatik Spektrum* 40, 153–160.

18. Galisson, F. (2000). The Fasta and Blast programs. ISMB 2000 Tutorial. `https://www.iscb.org/cms_addon/conferences/ismb2000/tutorial_pdf/galisson5.pdf`. visited Mar 5, 2005.

19. Grondahl, B., Puppe, W., Hoppe, A., Kühne, I., Weigl, J.A., Schmitt, H.J. (1999). Rapid identification of nine microorganisms causing acute respiratory tract infections by single-tube multiplex reverse transcription-PCR: feasibility study. *Journal of Clinical Microbiology* 37(1):1-7.

20. Hon, J., Martinek, T., Rajdl, K. and Lexa M. (2013). Triplex: an R/Bioconductor package for identification and visualization of potential intramolecular triplex patterns in DNA sequences. *Bioinformatics* 29(15):1900-1901.

21. Hon, J., Lexa, M. and Martinek, T. (2017). pqsfinder: User Guide. `https://bioconductor.org/packages/release/bioc/vignettes/pqsfinder/inst/doc/pqsfinder.html`, visited Oct 22, 2022.

22. Jin, Y. and Dunbrack, R.L. Jr. (2005). Assessment of Disorder Predictions in CASP6. *PROTEINS: Structure, Function, and Bioinformatics* Suppl 7:167–175.

23. Joly-Lopez, Z., Bureau, T.E. (2018). Exaptation of transposable element coding sequences. Current Opinion in Genetics and Development 49:34-42.

24. Jung, M., Dritschilo, A. and Kasid, U (1992). Reliable and efficient direct sequencing of PCR-amplified double-stranded genomic DNA template. *Genome Research* 1:171-174.

25. Kalendar, R., Khassenov, B., Ramankulov, Y., Samuilova, O., Ivanov, K.I. (2017). FastPCR: An in silico tool for fast primer and probe design and advanced sequence analysis. *Genomics* 109(3-4):312-319.

26. Karp, R.M. (1992). On-line algorithm versus off-line algorithms: how much is it worth to know the future? Technical report TR-92-044. `https://www1.icsi.berkeley.edu/pubs/techreports/TR-92-044.pdf`, visited Oct 22, 2022.

27. Kejnovsky, E. and Lexa, M. (). Quadruplex-forming DNA sequences spread by retrotransposons may serve as genome regulators. *Mobile Genetic Elements* 4(1):e28084.

28. Kejnovsky, E., Tokan, V. and Lexa, M. (2015). Transposable elements and G-quadruplexes. *Chromosome Research* 23:615-623.

29. Kent, W.J. (2002). BLAT–the BLAST-like alignment tool. *Genome Research* 12(4):656-64.

30. Klimentova, E., Polacek, J., Simecek, P. and Alexiou, P. (2020). PENGUINN: Precise Exploration of Nuclear G-Quadruplexes Using Interpretable Neural Networks. *Frontiers in Genetics* 11:1287.

31. Kronmiller, B.A. and Wise, R.P. (2008). TEnest: Automated Chronological Annotation and Visualization of Nested Plant Transposable Elements. *Plant Physiology* 146(1):45–59.

32. Kudlicki, A.S. (2016). G-Quadruplexes Involving Both Strands of Genomic Dna Are Highly Abundant and Colocalize with Functional Sites in the Human Genome. *PLoS ONE* 11(1).

33. Laganière, J., Deblois, G., Lefebvre, C., Giguère, V. (2005). Location analysis of estrogen receptor α target promoters reveals that FOXA1 defines a domain of the estrogen response. *Proceedings of the National Academy of Sciences USA* 102(33):11651-116.

34. Langmead, B., Salzberg, S. (2012). Fast gapped-read alignment with Bowtie 2. *Nature Methods* 9:357-359.

35. Lelli, K.M., Slattery, M. and Mann, R.S. (2012). Disentangling the Many Layers of Eukaryotic Transcriptional Regulation. *Annual Review of Genetics* 46:43-68.

36. Lexa, M. and Valle, G. (2004). Combining rapid word searches with segment-to-segment alignment for sensitive similarity detection, domain identification and structural modelling. BITS 2004 Conference, Padova, Italy. `http://bioinformatics.hsanmartino.it/bits_library/library/00074.pdf`, visited Oct 22, 2022.

37. Lexa, M., Martinek, T., Beck, P., Fucik, O., Valle, G. and Zara, I. (2007). Genomic PCR simulation with hardware-accelerated approximate sequence matching. Zelinka, I., Oplatková, Z., Orsoni, A. (eds.) *Proceedings 21st European Conference on Modelling and Simulation*. Praha: ECMS 2007:333-338.

38. Lexa, M., Snasel, V., Zelinka, I. (2009). Data-Mining Protein Structure by Clustering, Segmentation and Evolutionary Algorithms. In: Abraham, A., Hassanien, AE., de Carvalho, A.P.d.L.F. (eds) *Foundations of Computational Intelligence Volume 4. Studies in Computational Intelligence* 204:221-248.

39. Lexa, M., Brazdova, M., (2012). Prediction of significant cruciform structures from sequence in topologically constrained DNA - a probabilistic modelling approach. In: *Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms – BIOINFORMATICS* 2012:124-130.

40. Lexa, M., Cheeseman, J.M. (1995) Simulačné modely fotosyntézy a ich vzťah k problémom vodného režimu rastlín. In: *Vedecké práce Výskumného ústavu závlahového hospodárstva v Bratislave*. Bratislava: VÚZH, s. 105-117.

41. Lexa, M., Cheeseman, J.M. (1997). Growth and nitrogen relations in reciprocal grafts of wild-type and nitrate reductase-deficient mutants of pea (Pisum sativum L. var. Juneau). *Journal of Experimental Botany* 48(311):1241-1250.

42. Lexa, M., Martinek, T. and Brazdova, M. (2014). Uneven Distribution of Potential Triplex Sequences in the Human Genome - In Silico Study using the R/Bioconductor Package Triplex. In: *Proceedings of the International Conference on Bioinformatics Models, Methods and Algorithms - BIOINFORMATICS* 2014:80-88.

43. Lexa, M., Kejnovsky, E., Steflova, P., Konvalinova, H., Vorlickova, M., Vyskot, B. (2014). Quadruplex-forming sequences occupy discrete regions inside plant LTR retrotransposons. *Nucleic Acids Research* 42:968-978.

44. Lexa, M., Steflova, P., Martinek, T., Vorlickova, M., Vyskot, B., Kejnovsky, E. (2014b). Guanine quadruplexes are formed by specific regions of human transposable elements. *BMC Genomics* 15(1):1032.

45. Li , H., and Durbin, R.M. (2009). Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics* 25:1754-1760.

46. Li , R., Yu , C., Li , Y., et al. (2009). SOAP2: An improved ultrafast tool for short read alignment. *Bioinformatics* 25:1966-1967.

47. Lombardi, E.P. and Londoño-Vallejo, A. (2020). A guide to computational methods for G-quadruplex prediction. *Nucleic Acids Research* 48(1):1–15.

48. Martinek, T., Korenek, J., Fucik, O. and Lexa, M. (2006). A flexible technique for the automatic design of approximate string matching architectures. *IEEE Design and Diagnostics of Electronic Circuits and Systems*:81-82.

49. Martinek, T., Fucik, O., Beck, P. and Lexa, M. (2007). Automatic generation of circuits for approximate string matching. IEEE Design and Diagnostics of Electronic Circuits and Systems:1-6.

50. Martinek, T. and Lexa, M. (2008). Hardware Acceleration of Approximate Palindromes Searching. *International Conference on Field-Programmable Technology*:65-72.

51. Martinek, T., Vozenilek, J. and Lexa, M. (2009). Architecture model for approximate palindrome detection. *12th International Symposium on Design and Diagnostics of Electronic Circuits & Systems*:90-95.

52. Martinek, T. and Lexa, M. (2010). Hardware Acceleration of Approximate Tandem Repeat Detection. *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*:79-86.

53. Martinek, T. and Lexa, M. (2011). Architecture model for approximate tandem repeat detection. *ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*:239-242.

54. McNamee, L.M. and Ledley, F.D. (2013). Assessing the history and value of Human Genome Sciences. *Journal of Commercial Biotechnology* 19(4):3-10.

55. Mukundan, V.T., Phan, A.T. (2013). Bulges in G-quadruplexes: broadening the definition of G-quadruplex-forming sequences. *Journal of the American Chemical Society* 135(13):5017-28.

56. Neumann,P., Novák, P., Hoštáková, N., Macas, J. (2019). Systematic survey of plant LTR-retrotransposons elucidates phylogenetic relationships of their polyprotein domains and provides a reference for element classification. *Mobile DNA* 10:1.

57. Ning, Z., Cox, A.J., Mullikin, J.C. (2001). SSAHA: A Fast Search Method for Large DNA Databases. *Genome Research* 11:1725-1729.

58. Novak, P., Neumann, P., Macas, J. (2010) – Graph-based clustering and characterization of repetitive sequences in next-generation sequencing data. *BMC Bioinformatics* 11:378.

59. Nussinov, R. and Jacobson, A.B. (1980). Fast algorithm for predicting the secondary structure of single-stranded RNA. *Biochemistry* 77(11):6309-6313.

60. Qu, W., Shen, Z., Zhao, D., Yang, Y., Zhang, C. (2009). MFEprimer: multiple factor evaluation of the specificity of PCR primers. *Bioinformatics* 25:276-278.

61. Rozen, S. and Skaletsky, H.J. (2000) Primer3 on the WWW for General Users and for Biologist Programmers. In: Krawetz, S. and Misener, S., Eds., *Bioinformatics Methods and Protocols: Methods in Molecular Biology*, Humana Press, Totowa, 365-386.

62. Rychlik, W., Rhoads, R.E. (1989). A computer program for choosing optimal oligonucleotides for filter hybridization, sequencing and in vitro amplification of DNA. Nucleic Acids Res 17(21):8543-51.

63. Sahakyan, A.B., Chambers, V.S., Marsico, G. et al. (2017). Machine learning model for sequence-driven DNA G-quadruplex formation. *Scientific Reports* 7:14535.

64. SantaLucia, Jr. J. (1998). A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proceedings of the National Academy of Sciences USA* 95(4):1460-1465.

65. Sauria, M.E.G., Phillips-Cremins, J.E., Corces, V.G. and Taylor, J. (2015). HiFive: a tool suite for easy and efficient HiC and 5C data analysis. *Genome Biology* 16:237.

66. Stefos, G.C., Theodorou, G., Politis, I. (2022). Genomic landscape, polymorphism and possible LINE-associated delivery of G-quadruplex motifs in the bovine genes. Genomics 114(2):110272.

67. Wang, K., Li, H., Xu, Y., Shao, Q., Yi, J., Wang, R., Cai, W., Hang, X., Zhang, C., Cai, H., Qu, W. (2019). MFEprimer-3.0: quality control for PCR primers. *Nucleic Acids Research* 47(W1):W610–W613.

68. Yatsunyk, L.A., Bryan, T.M., Johnson, F.B. (2012). G-ruption: the third international meeting on G-quadruplex and G-assembly. *Biochimie* 94(12):2475-83.

A Lexa et al., 2001

BIOINFORMATICS APPLICATIONS NOTE Vol. 17 no. 2 2001
Pages 192–193

Virtual PCR

M. Lexa[1,*], J. Horak[1] and B. Brzobohaty[1, 2]

[1]Laboratory of Plant Molecular Physiology, Masaryk University Brno, Faculty of Science, Kotlarska 2, 611 37, Brno, Czech Republic and [2]Institute of Biophysics AS CR, Kralovopolska 135, 612 65, Brno, Czech Republic

Received on August 3, 2000; revised on August 25, 2000; accepted on September 29, 2000

ABSTRACT

Summary: We present an algorithm that uses public sequence data to predict PCR products. The algorithm is implemented as a CGI script. Output is compared to real-world PCR.

Availability: Perl code and instructions for installation are freely available over the internet at http://www.sci.muni.cz/LMFR/vpcr.html

Contact: m-lexa@sci.muni.cz

INTRODUCTION

PCR has become an indispensable tool in molecular biology. Utilizing oligonucleotide primers and DNA polymerase, it amplifies DNA up to many thousand bases in length (Bej *et al.*, 1991). Success of a particular application mostly depends on the choice of proper primers. Optimally, they will not form dimers or hairpins. Most importantly, they will anneal to DNA sequences of interest, but not to other sequences. A quick similarity search will reveal possible complementary sequences for a primer. However, it will not tell whether one can expect a PCR product because of another primer annealing to the template in close proximity. The abundance of genomic sequences in public databases makes it now possible to simulate amplification results. We describe a 'virtual PCR' software tool that uses public DNA sequence databases to predict PCR products for arbitrary primer pairs.

ALGORITHM AND IMPLEMENTATION

The 'virtual PCR' program (VPCR) exists as a CGI script written in Perl, that after installation can be accessed using a WWW browser. It processes user-given primers and obtains BLAST search results (Altschul *et al.*, 1990) with these primers to identify sequences in public databases that are complementary to any two of them. Finally, it prints out potential PCR products.

In more detail, the process of using the script starts by filling a form that accepts user input. The only strictly required inputs are the primer sequences to be used. By default, the algorithm searches the entire GenBank database for matches and after approximately 60s displays identified PCR products. Most often, the user will find it useful to restrict the search to a particular species. Changing the E-value limit for the BLAST search will change the level of homology required for a successful match, thereby increasing or decreasing the specificity of the VPCR. This is somewhat similar to changing annealing temperature in a real-world reaction. Appropriate search mode and sequence database can be chosen in the input form, with detailed explanation of those given on NCBI BLAST and our webpages.

EXPERIMENTAL VERIFICATION

To demonstrate the potential of the VPCR algorithm, we show its use and how it compares to data from real-world PCR. We have chosen to show two envisioned uses: (i) evaluation of primers to be used in amplification from genomic DNA; and (ii) identification of PCR products with primers amplifying a gene family.

PCR reactions used to evaluate the algorithm were carried out by standard methods using *Arabidopsis* DNA and these primers:

ARR5a	=	GTTGATTCTCTCTATCTCTCTCACG
ARR5b	=	CACACCACCATTTTACATATCTC
ARR7a	=	GTTGGTGAGGTCATGAGGATGGAGATTC
ARR7b	=	GTTTTGCTAAGGTCTTGGCCTCTATACAT
GEN1	=	CATGTTCTTGCYGTYGATGAYAGT
GEN2	=	CCAGTCATKCCAGGCATWSAG
GEN3	=	ATAARAAATCYTCAGCWCCTTC

Single gene-specific primer evaluation

Two pairs of primers previously designed to amplify a region of ARR5 and ARR7 genes were used to test the performance of the algorithm. The resulting VPCR products are displayed and visually compared with a gel of real PCR products in Figure 1. Simulated bands which correspond to experimental bands are marked by arrows in Figure 1A. The algorithm has correctly identified the ARR7 product obtained in a parallel PCR reaction. It found only one additional product of low

[*]To whom correspondence should be addressed.

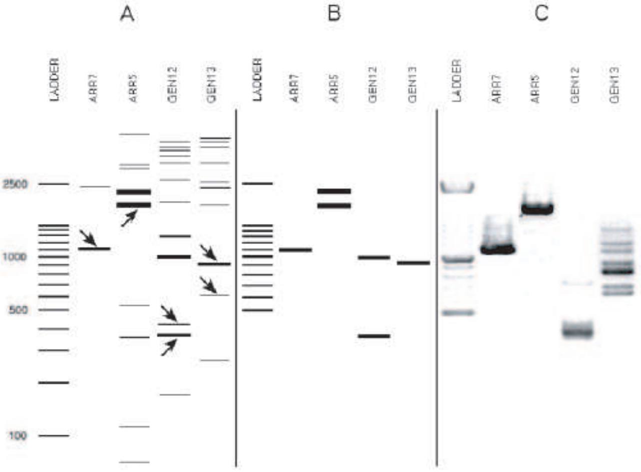


Fig. 1. The VPCR algorithm and real-world PCR. (A) Graphical presentation of VPCR output for the following primer sets: ARR7a, b (lane 2), ARR5a, b (lane 3), GEN1,2 (lane 4) and Gen1,3 (lane 5). The width of the bands was chosen to represent the E-value of the BLAST search. Wider bands represent higher homology between primers and template. Arrows show bands present on gels of actual PCR products. (B) Graphical presentation of simulation results as in (A). Only high-specificity bands are shown. (C) Agarose gels of corresponding PCR reactions. Primers as in (A) and (B). *Arabidopsis* genomic DNA was used as template.

specificity. This product could not be seen on the gel (Figure 1C). With ARR5 primer pair, the algorithm identified the product seen in real PCR. However, a number of additional PCR products, most of which were much longer and had lower specificity values, were found (Figure 1A). Figure 1B shows only simulated products of high specificity. Comparison with the bands given in Figure 1C shows that eliminating the low-specificity bands improves the match between simulation and experiment.

Gene family-specific primer evaluation

A set of degenerate primers previously designed to amplify a conserved region of all ARR genes was used to further test the performance of the algorithm. These were used as pairs GEN1,2 and GEN1,3. The resulting VPCR products are contrasted to real PCR products in Figure 1. Simulated bands which correspond to real PCR products seen on a gel are marked by arrows in Figure 1A. While the algorithm has correctly identified only a small fraction of ARR products, it identified additional PCR products, most of which were longer, low-specificity sequences (Figure 1A).

DISCUSSION

We have constructed and tested a VPCR algorithm based on the BLAST local alignment search of GenBank non-redundant sequences. The corresponding perl CGI script is available on the Internet. To our knowledge, this is a novel resource useful in PCR primer design to evaluate primer specificity. Additionally, people obtaining multiple PCR products may consult VPCR to identify them.

Currently the script utilizes BLAST searches on the NCBI server. However, it is the policy of NCBI to protect their server and prevent frequent queries originating from a single installation. Therefore, we are unable to run the VPCR script for general public from our web pages. Instead, readers are encouraged to download and install their private copy on their own.

In principle, test results corresponded with expectations, but specificity of the prediction must be improved before wider use. Using VPCR in its present form, one should limit output to high-specificity products by using E-value < 10. We see the following areas of improvement to increase the predictive ability of VPCR:

(i) using databases that cover the whole genome of a given organism;

(ii) identification of PCR products across boundaries of GenBank entries;

(iii) replacement of BLAST routines with calculations of primer binding; and

(iv) correcting for dimer and hairpin formation.

With these improvements the algorithm could become a standard component of primer design software.

ACKNOWLEDGEMENTS

This work was supported by the Grant Agency of the Czech Republic (204/99/D024; Matej Lexa; software development) and the Czech Ministry of Education (J07/98:143100008; Bretislav Brzobohaty; real-world PCR). We wish to thank Dr Hana Konecna for primer synthesis and Prof John M. Cheeseman for reading the manuscript.

REFERENCES

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

Bej,A.K., Mahbubani,M.H. and Atlas,R.M. (1991) Amplification of nucleic acids by polymerase chain reaction (PCR) and other methods and their applications. *Crit. Rev. Biochem. Mol. Biol.*, **26**, 301–334.

B Lexa and Valle, 2003

BIOINFORMATICS APPLICATIONS NOTE
Vol. 19 no. 18 2003, pages 2486–2488
DOI: 10.1093/bioinformatics/btg350

PRIMEX: rapid identification of oligonucleotide matches in whole genomes

Matej Lexa[1,*] *and Giorgio Valle*[2]

[1] *Laboratory of Functional Genomics and Proteomics, Masaryk University Brno, Kotlarska 2, 61137 Brno, Czech Republic and* [2] *CRIBI Biotechnology Center, University of Padova, via U.Bassi, 58/b, 35131 Padova, Italy*

Received on March 24, 2003; revised on May 30, 2003; accepted on July 1, 2003

ABSTRACT

Summary: PRIMEX (PRImer Match EXtractor) can detect oligonucleotide sequences in whole genomes, allowing for mismatches. Using a word lookup table and server functionality, PRIMEX accepts queries from client software and returns matches rapidly. We find it faster and more sensitive than currently available tools.

Availability: Running applications and source code have been made available at http://bioinformatics.cribi.unipd.it/primex

Contact: m-lexa@sci.muni.cz

INTRODUCTION

The latest developments in genomics have provided researchers with a number of whole-genome sequences and a new generation of bioinformatic tools. The prediction of non-trivial PCR amplification products from genomic DNA is a field that despite its considerable practical value, has not been satisfactorily investigated at the bioinformatic level.

To predict the outcome of an arbitrary PCR reaction, it is crucial to identify all the potential priming sites for the primers in use. The search for candidate matches must be sensitive enough to extract all the relevant candidate sequences. At the same time it should be relatively fast, because with large genomes it tends to be the time-limiting step in PCR simulation. Using BLAST in our first algorithms to predict PCR products (VPCR 1.0; Lexa *et al*., 2001), we realized several shortfalls of that program for our purposes. We decided to write a new program that would be better suited for this purpose. The new program called PRIMEX (PRImer Match EXtractor) may also find applications in other areas, such as oligonucleotide probe selection for hybridization experiments or genome alignment.

We provide interfaces allowing one to rapidly query whole genomes for occurrences of short oligonucleotide sequences with mismatches. These include a CGI script, a Perl script with a developer's library and a standalone C++ program. We propose a distributed network of PRIMEX servers to provide search capabilities for most of the sequenced genomes in a short period of time.

SYSTEM AND METHODS

PRIMEX has been written in C++, compiled with gcc and executed under Debian Linux 2.4.16 on a dual-processor Athlon 1600 machine. The server/client functions of the program are built around a socket library written by Tougher (2002, http://www.linuxgazette.com/issue74/tougher.html).

ALGORITHM

In designing the algorithm, we were inspired by some of the ideas behind BLAT (Kent, 2002) and SSAHA (Ning *et al*., 2001) that give our software speed, such as lookup table use and server functionality.

Lookup table

The program creates a word-sized window positioned at the first nucleotide of the searched genomic sequence. It moves the window along the sequence, recording the position of each word encountered into a lookup table. The lookup table is an array of lists. The indices of the array represent the words, while the lists contain all the positions at which the word has been encountered. Lookup table may be saved for future use.

Queries

The program enters a server mode and starts listening to queries. After receiving a query, it extracts words from the query sequence and searches for matches, using the lookup table. Pre-defined parameters set the allowed number of mismatches to be tolerated. The collected matches are then filtered for the allowed number of mismatches, duplicates are eliminated and the result is written out.

[*] To whom correspondence should be addressed at CRIBI Biotechnology Center, University of Padova, via U.Bassi, 58/b, 35131 Padova, Italy.

Table 1. Performance of various search programs when looking for oligonucleotide `AAAAAATGATCAATTTACAT` in the *A.thaliana* genome

Program	Total	Mismatches							Search time (s)
		0	1	2	3	4	5	6	
BLAST	162	1							12
BLAST-O	2	1							5
FASTA	72	1	0	4	13	23	17	12	53
FASTA-O	1	1							19
BLAT	0	0							80
SSAHA	0	0							71
SSAHA-O	3	1							10
CGC FP-O	1	1							10 (estimate)
EMBOSS	983	1	0	5	104	86	8		18
EMBOSS-O	1	1							14
TACG	1	1							49
AGREP	1204	1	0	5	100	779	3632		34
AGREP	1	1							2
PRIMEX (3)	4517	1	0	14	199				56
PRIMEX-S (5)	12140	1	0	14	199	1686	10240		19
with insertions	24902	1							38
PRIMEX-S (4)	1900	1	0	14	199	1686			4
with insertions	3621	1							12
PRIMEX-S (3)	214	1	0	14	199				1
with insertions	286	1							1
PRIMEX-S (2)	15	1	0	14					<1
with insertions	19	1							<1
PRIMEX-S (1)	1	1	0						$\ll 1$
PRIMEX-SO (0)	1	1							$\ll 1$

The -O suffix represents searches for high-similarity matches. The -S suffix indicates that the program ran in server mode. The numbers in parentheses are mismatch limits. References: BLAST (Altschul *et al*., 1990), FASTA (Pearson and Lipman, 1988), BLAT (Kent, 2002), SSAHA (Ning *et al*., 2001), CGC FindPatterns (Accelrys, San Diego, CA, USA), EMBOSS Fuzznuc (http://www.hgmp.mrc.ac.uk/Software/EMBOSS/), TACG (Mangalam, 2001) and AGREP (Wu and Manber, 1994).

IMPLEMENTATION

Important server functions

query_remote S	Find matches for oligonucleotide S
dump_state_remote	Report the current settings of the server
get_seq_remote M:N	Report the sequence between M and N

The results of a query are returned in lines. Each line contains the following data: primer number, query sequence, matched sequence, clone name, position within the clone, orientation and number of matching basepairs. For example:

0 `AAAAATTTTTCCCCCGGGGG` `AAAAATTCTGCC-ACCGGGGG`
15237134 111102206 + 17

Performance

We carried out a series of performance tests. Table 1 shows, which of the related routines were able to find an exact or approximate match to a 20 bp oligonucleotide in the more than 100 MB-long genomic sequence of *Arabidopsis thaliana*. It also lists the times in seconds necessary to provide the answers. Without the server speed-up, AGREP was the best program. However, this program does not provide server functionality that could be used to accelerate the search in DNA sequences, therefore, PRIMEX remains currently the only high-speed choice for oligonucleotide searches in whole genomes.

DISCUSSION

The future of PRIMEX depends on the applications that could benefit from its abilities. For instance, primer design software may query the server and check primers against whole genomes rapidly. PCR simulation software (Rubin and Levy, 1996; Lexa *et al.*, 2001) could use PRIMEX. Genome alignment and analysis software could use repeated PRIMEX queries to find large-scale similarities between two or more genomes.

We currently run two PRIMEX servers. A master-list file defining how to access these services is available at http://bioinformatics.cribi.unipd.it/primex/primex_master.txt

Current example: *147.162.3.227|30000|Arabidopsis thaliana |10|NCBI A.thaliana genome release 147.251.24.2|30000| Heliobacter species|8|NCBI Helicobacter sp. genome release*

This file is also consulted by the Perl PRIMEX communication library available for download from our website. Each line specifies the IP number, port, organism, word size and optional notes for running servers. Please, contact the authors, if would like to include your server.

REFERENCES

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

Kent,W.J. (2002) BLAT—the BLAST-like alignment tool. *Genome Res.*, **12**, 656–664.

Lexa,M., Horak,J. and Brzobohaty,B. (2001) Virtual PCR. *Bioinformatics*, **17**, 192–193.

Mangalam,H.J. (2001) tacg—a grep for DNA. *BMC Bioinformatics*, **3**, 8.

Ning,Z., Cox,A.J. and Mullikin,J.C. (2001) SSAHA: a fast search method for large DNA databases. *Genome Res.*, **11**, 1725–1729.

Pearson,W.R. and Lipman,D.J. (1988) *Proc. Natl Acad. Sci. USA*, **85**, 2444–2448.

Rubin,E. and Levy,A.A. (1996) A mathematical model and a computerized simulation of PCR using complex templates. *Nucleic Acid Res.*, **24**, 3538–3545.

Tougher,R. (2002) Linux socket programming in C++. Linux Gazette 74.

Wu,S. and Manber,U. (1994) A fast algorithm for multi-pattern searching. Technical Report, The Computer Science Department, The University of Arizona.

C Lexa et al., 2011

BIOINFORMATICS ORIGINAL PAPER

Vol. 27 no. 18 2011, pages 2510–2517
doi:10.1093/bioinformatics/btr439

Sequence analysis

Advance Access publication July 16, 2011

A dynamic programming algorithm for identification of triplex-forming sequences

Matej Lexa[1],*, Tomáš Martínek[2], Ivana Burgetová[2], Daniel Kopeček[1] and Marie Brázdová[3]

[1]Department of Information Technology, Faculty of Informatics, Masaryk University, 60200 Brno, [2]Department of Computer Systems, Faculty of Information Technology, Brno Technical University, 61266 Brno and [3]Department of Biophysical Chemistry and Molecular Oncology , Institute of Biophysics, Academy of Sciences of the Czech Republic v.v.i., CZ-61265 Brno, Czech Republic

Associate Editor: John Quackenbush

ABSTRACT

Motivation: Current methods for identification of potential triplex-forming sequences in genomes and similar sequence sets rely primarily on detecting homopurine and homopyrimidine tracts. Procedures capable of detecting sequences supporting imperfect, but structurally feasible intramolecular triplex structures are needed for better sequence analysis.

Results: We modified an algorithm for detection of approximate palindromes, so as to account for the special nature of triplex DNA structures. From available literature, we conclude that approximate triplexes tolerate two classes of errors. One, analogical to mismatches in duplex DNA, involves nucleotides in triplets that do not readily form Hoogsteen bonds. The other class involves geometrically incompatible neighboring triplets hindering proper alignment of strands for optimal hydrogen bonding and stacking. We tested the statistical properties of the algorithm, as well as its correctness when confronted with known triplex sequences. The proposed algorithm satisfactorily detects sequences with intramolecular triplex-forming potential. Its complexity is directly comparable to palindrome searching.

Availability: Our implementation of the algorithm is available at http://www.fi.muni.cz/˜lexa/triplex as source code and a web-based search tool. The source code compiles into a library providing searching capability to other programs, as well as into a stand-alone command-line application based on this library.

Contact: lexa@fi.muni.cz

Supplementary Information: Supplementary data are available at Bioinformatics online.

Received on March 16, 2011; revised on June 24, 2011; accepted on July 13, 2011

1 INTRODUCTION

Triplexes are local structural variants of DNA, wherein the molecule adopts a specific secondary structure differing from a canonical duplex by the recruitment of a third DNA strand. The third strand binds to the duplex by Hoogsteen or reverse Hoogsteen bonds with stringency of the same order of magnitude as duplex-forming strands for the most stable nucleotide combinations (reviewed by Frank-Kamenetskii and Mirkin, 1995). Depending on the source of the third strand, triplex DNA can be *intrastrand* and *interstrand*, or *intramolecular* and *intermolecular*. The third strand may just come from the other strand of the same DNA duplex or from a completely different DNA molecule, as is the case with triplex-forming oligonucleotides (Knauert and Glazer, 2001). Nucleotides in the middle strand of a triplex have Watson–Crick base pairing to one nucleotide and Hoogsteen or reverse Hoogsteen pairing to another nucleotide. Together they form a triplex-forming triplet (also called triad) (Mirkin and Frank-Kamenetskii, 1994; Soyfer and Potaman, 1995). Depending on the orientation of the third strand, we distinguish *parallel* and *antiparallel* triplexes, named according to the orientation of the third strand in respect to the central strand. Figure 1 shows eight types of *intramolecular* triplex structures considered in this article. A given sequence on the (+) strand of a DNA molecule can possibly support all eight types, but necessarily, only one of the types will be formed at any particular moment. In DNA triplexes, there is a requirement for neighboring triplets to be isomorphic, otherwise the potential triplet would be under strain, hindering the binding of the third strand (Rathinavelan and Yathindra, 2006; Thenmalarchelvi and Yathindra, 2005). Regardless of orientation and geometry, the middle nucleotide is generally a purine-containing one, to support the extra hydrogen bonds needed to bind the third nucleotide.

Because the middle nucleotide is almost invariably one with a purine base, attempts to correlate sequence with triplex-forming properties usually involve detection of homopurine and homopyrimidine tracts in the analyzed sequence. For example, Gaddis *et al*. (2006) created a web-based program that identifies target sequences for triplex-forming oligonucleotides. The program identifies homopurine stretches that are allowed to be occasionally interrupted by a pyrimidine. While this is an appropriate method for detection of strong triplex-forming signals, we consider this to be an oversimplification. Numerous papers have reported the existence of imperfect triplexes (Mergny *et al*., 1991; Roberts and Crothers, 1991; Xodo *et al*., 1993), including cases where the authors deliberately changed individual nucleotides to observe the effects of such change. Changes resulting in the formation of non-canonical triplets did not necessarily disrupt the entire triplex. It is conceivable that many of the imperfect triplexes may still have similar biological activity to their ideal counterparts. One possible explanation for the existence of imperfect triplexes is that they may allow an overlap between the structural signal and some other sequence feature,

*To whom correspondence should be addressed.

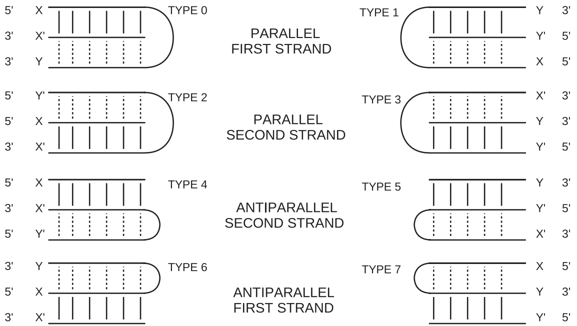


Fig. 1. Eight types of triplexes that are detected in separate runs of the algorithm for a given region. Numbering of types is shown as used in the accompanying software (Supplementary Material). Watson–Crick base pairing is shown by vertical bars. X and Y are two nucleotides on the same strand that will form a triplet. The eight possible triplets are: $Y.X'X$, $Y'.XX'$, $Y'.X'X$, $Y.XX'$, $X.Y'Y$, $X'.YY'$, $X'.Y'Y$ and $X.YY'$ (N', a nucleotide complementary to N; '.', Hoogsteen or reverse Hoogsteen bond).

such as nucleosome positioning pattern or a regulatory protein-binding sequence. Kinniburgh (1989) proposed a triplex structure containing a single deletion to explain his experimental results. Additionally, analyzed sequences may contain errors, including occasional deletions and insertions.

The existence of triplex DNA has been repeatedly associated with important biological processes at the molecular level, making them an attractive target in sequence analysis. Most of the observed associations suggest roles in mutagenesis, recombination and gene regulation. Non-B DNA structures, including DNA triplexes, have been shown to cause deletions, expansions and translocations in both prokaryotes and eukaryotes (Raghavan *et al.*, 2005). Their distribution is not random and often colocalizes with sites of chromosomal breakage (Zhao *et al.*, 2010). Triplex structures can block the replication fork and result in double-stranded breaks (Dixon *et al.*, 2008). Unlike other non-canonical structures, triplex-forming sequences are found frequently in promoters and exons and have been found to be involved in regulating the expression of several disease-linked genes (Wang and Vasquez, 2004). In some cases, the mutagenesis induced by such sequences is enhanced by their transcription (Belotserkovskii *et al.*, 2007), possibly via transcriptional arrest.

Sequence–structure relationships of triplexes were brought into a small number of computational tools for identifying relevant sequences in genome sequences. Schroth and Ho (1995) analyzed the occurrence of inverted and mirror repeats in three genomes. Hoyne *et al.* (2000) analyzed the *Escherichia coli* genome for intrastrand triplex sequences. Another recent work (Cer *et al.*, 2010) created a web-based catalog of non-B DNA sequences in major mammalian genomes. Their definition of triplex covers the most stable canonical triplexes made of G.GC/A.AT and C.GC/T.AT triplets, but leaves little room for possible errors. Jenjaroenpun and Kuznetsov (2009) created a web-based analysis tool for triplex target sequences.

Intramolecular triplex DNA (also called H-DNA) has been shown to exist both *in vivo* and *in vitro* (Hanvey *et al.*, 1988). Its formation also depends on the topological state of the given DNA molecule. While sequences supporting canonical triplets, such as $(CT(T))_n$ and $(GA(A))_n$ tracts, form triplexes readily, imperfect triplexes may require special conditions, such as low superhelical density or certain pH to form. *In vitro*, superhelical density and pH can be easily controlled. *In vivo*, pH is tightly controlled by the cell, while the topological state of any stretch of genomic DNA is generally unknown, but presumed to be under regulatory control as well. This uncertainty is the main reason for using the term 'triplex-forming sequence' or 'triplex-forming potential', which hints that while the sequence should be capable of forming a triplex, it may only be formed under special circumstances.

2 APPROACH

Based on available literature, we assume there are two important classes of sequence-based imperfections (errors) destabilizing potential triplex structures.

- Base pairing mismatch
- Geometrical mismatch

A base pairing mismatch occurs upon the formation of a nucleotide triplet that does not support strong Hoogsteen or reverse Hoogsteen bonds. The ability to form the bond and its strength is related to the number of hydrogen bonds that can be made between the second and third strand base. In this article, we present an algorithm that is based on scores assigned to base triplets. The scores are meant to approximate energy contributions of individual triplets, but at the same time to be simple enough to support rapid searching that could be used as pre-filtering, preceding detailed energy calculations on the candidate sequences.

A geometrical mismatch occurs when directly neighboring triplets in a structure are not isomorphic. This places extra stress on the backbone of the third DNA strand preventing it from creating optimal hydrogen bonds. According to Thenmalarchelvi and Yathindra (2005), conformational changes necessitated by triplet non-isomorphism are found to induce an alternative zig-zag backbone structure for the third strand in special cases. Accordingly, we made our algorithm favor triplet combinations that are either isomorphic or made of non-isomorphic pairs that could form a zig-zag shape by canceling their geometric effect on the third strand backbone.

We currently ignore other known factors of triplex DNA formation, such as the competition between alternative structures (Rippe *et al.*, 1992), fourth strand (the strand which is not part of the predicted triplex) secondary structure, effects of C+ distribution (James *et al.*, 2003; Seidman and Glazer, 2003) and other distortions caused by electrostatic forces (Kang *et al.*, 1992; Tan and Chen, 2006). Most of these factors depend non-trivially on the environment (Plum *et al.*, 1995). Since the algorithm does not consider the environment, we focus primarily on sequence-coded effects and the resulting constraints which can be computed using the information from primary structure. Destabilizing effects of loop lengths that differ from the optimum of about five nucleotides (Haasnoot *et al.*, 1986) and the overall length of the triplex (Tan and Chen, 2006) are partly accounted for, since these parameters can be set as hard limits in our implementation, to narrow the search space.

3 METHODS

Datasets: to evaluate the algorithm on selected datasets, we prepared a set of sequences to work with (all ~4.7 Mb to match the size of *E.coli* genome): (i) a random nucleotide sequence; (ii) *E.coli* K-12 MG1655 complete genome (the 1995 U00096.1 version to be able to compare our results to previous publications); (iii) *E.coli* K-12 MG1655 complete genome (the current U00096.2 version for proper positioning in genome browsers); (iv) a randomized nucleotide sequence of the same *E.coli* genome; (v) a part of the human chromosome 5 sequence (positions 144635154–149340649) and (vi) a randomized version of the same human sequence. For the human randomized sequence, we also generated a triplex-seeded version with 418 triplex-forming sequences from literature inserted at positions $\sim$10 000 bp apart. All the sequence data are available as Supplementary Material and can also be downloaded from *http://www.fi.muni.cz/~lexa/triplex*. Random sequences were generated with equal probability for all four bases, and were prepared with an in-house algorithm seqmix-0.2 (Supplementary Material).

Molecular simulations of triplets: to obtain objective information about isomorphic groups, we analyzed the angle and radius formed by C1 atoms of triplet nucleotides as defined in Thenmalarchelvi and Yathindra (2005). The groups were determined using the following procedure. First, the structures of all considered triplets were constructed using the NAB language from AmberTools 1.4 and their potential energy surface was explored for local minima by moving and rotating the third (Hoogsteen) base in the plane formed by the other two bases. The energy function was parametrized using the *ff99bsc0* set (Perez *et al*., 2007). The obtained local minima were filtered according to the values of the C1 angle (t) and the ratio $|WH|/|CH|$, where $|WH|$ represents the distance between the C1 atoms of the Hoogsteen pair and $|CH|$ represents the distance between the C1 atoms of the mutually unpaired bases. Filtering thresholds were derived from measurements on a set of real structures, namely the structures 135D, 149D, 1BCB, 1D3X (PDB identifiers). The specific thresholds used were $70 \leq t \leq 130$ and $0.54 \leq |WH|/|CH| \leq 0.88$. From the resulting set of local minima, the structure with the lowest potential energy was selected as the source of the parameters t and r (the radius of the circle formed by the C1 atoms). Finally, the groups were established by performing cluster analysis using Ward's method and euclidean distance between the (t,r) vectors. These results were interpreted to obtain isomorphic groups in Table 1, and detailed results are available as Supplementary Material.

Testing overview: we tested our implementation for correctness and usability. Clearly, the algorithm will only be useful, if it is capable of identifying potential triplex-forming sequences in a genomic background with a reasonable success rate. To test the implementation in this respect, we performed statistical tests on real and randomized sequences, a sequence recovery test on the triplex-seeded sequences, and we compared our solution to previously published results for the *E.coli* genome (Hoyne *et al*., 2000) and a currently published non-B DNA database (Cer *et al*., 2010).

Statistical tests: the statistical tests served to find parameters for the distribution of scores on randomized sequences and establish a proper threshold above which candidate hits should be considered significant. The distribution of scores was modeled according to principles used for evaluating BLAST results and other sequence similarity scores (Altschul *et al*., 1994; Korf *et al*., 2003), since the alignment of a DNA strand against itself is statistically similar to aligning two different sequences. This treatment allowed us to fit the score distribution with an extreme value distribution function and fit the parameters λ and μ as described by Korf *et al.* (2003). To carry out the calculation, we used a function from hmmer-2.3.2 source code (Eddy, 1997).

Recovery tests: the recovery tests evaluated how many of the introduced triplex-forming sequences were recovered for a selected significance threshold (P-value) from different backgrounds sequences. We used the commonly used characteristics for such experiments: specificity (precision), sensitivity (recall), F_2 measure and accuracy (Manning *et al*., 2008). The algorithm was tested against our triplex-seeded sequence and a database of non-B DNA (Cer *et al*., 2010).

Table 1. Triplex scoring of canonical and less usual triplets

Triplex type	Triplet H.WC:WC	Score (*tts*)	Isomorphic group	References
PARALLEL	T.A:T	2	a	Goni *et al*. (2004)
	T.G:C	1	a	Ghosal and Muniyappa (2006)
	C.G:C	2	a	Walter *et al*. (2001),Goni *et al*. (2004)
	G.G:C	1	b	Soyfer and Potaman (1995)
	G.T:A	2	b	Gowers and Fox (1998)
	T.C:G	1	b	Soyfer and Potaman (1995)
ANTIPARALLEL	A.A:T	2	c	(Goni *et al*. (2004), Mirkin and Frank-Kamenetskii (1994))
	A.G:C	1	d	Mirkin and Frank-Kamenetskii (1994), Raghavan and Lieber (2007)
	T.A:T	2	c	Goni *et al*. (2004), Mirkin and Frank-Kamenetskii (1994)
	T.C:G	1	e	Raghavan and Lieber (2007), Beal and Dervan (1992)
	C.A:T	1	d	Raghavan and Lieber (2007), Soyfer and Potaman (1995),Dayn *et al*. (1992)
	G.G:C	2	e	Goni *et al*. (2004), Mirkin and Frank-Kamenetskii (1994)

The final score values for both Hoogsteen and reverse-Hoogsteen bonds are in accordance with tables 4.1 and 4.2 in Soyfer and Potaman (1995). Isomorphic groups shown here are based on residual twist calculations using molecular dynamics simulations with the *nbd* program (AmberTools). ., Hoogsteen bp; :, Watson–Crick bp; *tts*, tabulated triplet score.

Escherichia coli tests: we compared our tool and its performance on the *E.coli* genome sequence to the results published by Hoyne *et al.* (2000). Additionally, we calculated the genome positioning of program output in respect to known *E.coli* genes, counting the frequency with which predicted triplexes fell inside the gene, outside any genes or intersected with them. Distance to the closest gene was calculated as shown in Figure 6.

4 THE ALGORITHM

Our approach to search for approximate triplexes is based on a dynamic programming (DP) algorithm to search for approximate palindromes that can be traced back to Landau and Vishkin (1986). The relationship between triplex DNA and palindromes stems from the fact, that one of the DNA strands in the triplex must fold back onto itself, either for Hoogsteen base pairing or for reverse Hoogsteen base pairing, depending on the type of triplex that is to be formed (parallel or antiparallel) and the nucleotide sequence present at the site in question. We will call the part of the triplex that folds back onto itself *self-recognizing*.

A DP matrix is constructed so that one side represents the original sequence, while the other contains the same sequence written backwards (Fig. 2). With such setup, the main antidiagonal of the DP matrix represents the n possible central starting positions for the self-recognizing parts of triplexes with an odd number of nucleotides in the loop. The neighboring antidiagonal contains the other $n-1$ possible starting sites for the triplexes with even number of nucleotides in their loops. Naturally, diagonals starting at any of these positions represent potential triplexes. If we fill the cells representing the starting positions with zeros, we can start filling the DP matrix along the diagonals. At each position $[i,j]$ of the DP matrix, we compare the symbols at positions i and j in the original sequence. If they represent a pair present in triplex-forming triplets (tabulated in Table 1), they are evaluated with positive score. In opposite case, they are penalized with a negative score value. The numbers entered represent the best score in the subsequence evaluated so far.

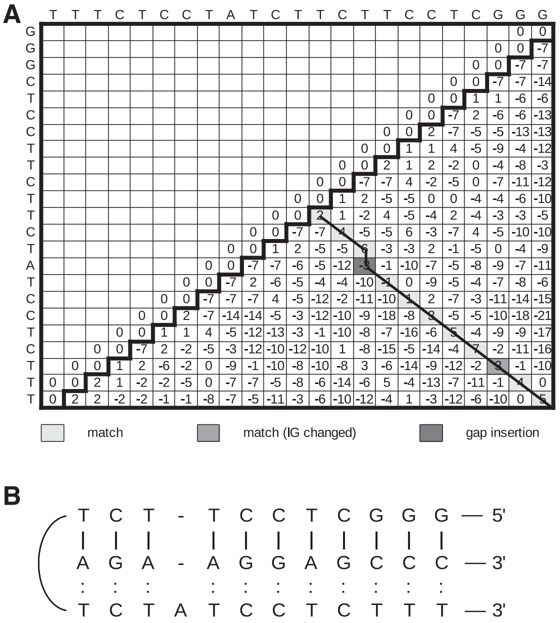

A
T T T C T C C T A T C T T C T T C C T C G G G
G G G C T C C T T C T T C T A T C C T C T T T
match
match (IG changed)
gap insertion
B
T C T - T C C T C G G G — 5'
A G A - A G G A G C C C — 3'
T C T A T C C T C T T T — 3'

Fig. 2. Triplex detection by the DP algorithm demonstrated on the string *gggctccttcttctatcctcttt*. (**A**) The DP matrix with calculated score values. Because of space limitations, loop size was forced to 0. (**B**) Triplex alignment. Hoogsteen bonds are shown by semicolons.

The necessity for a dynamic programming algorithm comes from the possibility to insert gaps into the triplexes, where symbols in some positions have no symbols to pair up with in the other arm of the self-recognizing sequence. In terms of the described algorithm, this means moving from one diagonal to a neighboring one when calculating the score. At any position, three possibilities are evaluated:

(1) Extending the existing triplex along the diagonal - *match* or *mismatch*,

(2) Inserting a gap at position *i* of the original sequence - *insertion*,

(3) Inserting a gap at position *j* of the original sequence - *deletion*.

The solution that leads to the maximum score value is recorded in the DP matrix, while the other possibilities are discarded.

In comparison to a similar algorithm for approximate palindrome detection, we have introduced three important modifications. First, we redefined the concept of match and mismatch. Instead of being made up by pairs of nucleotides with only two possible base pairs, triplexes can be thought of as sequences of triplets with many possible combinations of nucleotides in the triplet. There are 16 possible base pairs for parallel DNA strands and another 16 for antiparallel strands. For these reasons, we constructed a general similarity matrix instead of using a single match rule and score.

Second modification brings geometrical considerations into the algorithm, making certain sequences of triplets less desirable than others. This is similar to the nearest-neighbor scoring used in duplexes, although we are not as much concerned about base stacking as we are about the geometry of the third strand and its ability to position itself for optimal hydrogen bonding. As discussed by Rathinavelan and Yathindra (2006); Thenmalarchelvi and Yathindra (2005), some combinations disrupt the backbone geometry. We therefore decided to divide the triplets into isomorphic groups. Groups of triplets from one group are more likely to form stable triplexes than other sequences. Our modification assigns the information about isomorphic groups to the last computed DP matrix cell on each diagonal. When calculating a new cell, we lower the score if the newly evaluated triplet belongs to a different isomorphic group than the preceding one. The score calculation is

$$S[i,j] = \max \begin{cases} S[i,j-1]+gp \\ S[i-1,j]+gp \\ S[i-1,j-1]+tts[a,b]+nip \end{cases} \qquad (1)$$

where *a*, *b* are characters at appropriate row and column, *tss* is tabulated triplet score, *gp* is gap penalty and *nip* is no-isomorphism penalty.

The third consideration is to account for all the possible ways a triplex can form from a given sequence, i.e. which three strands combine together and in which orientation (Fig. 1). There are always eight ways that can give rise to a intramolecular triplex at a given position, since there are two strands that can serve as the third strand, each having two ends that can loop back onto the double-stranded region and in each of these cases it can attach on either side of the duplex in a parallel or antiparallel fashion, forming Hoogsteen and reverse Hoogsteen bonds, respectively. In order to detect all types of triplexes the computation is repeated eight times with scoring matrices specific for parallel and antiparallel triplexes.

4.1 Scoring function

We evaluate the combinations based on their ability to form Hoogsteen base pairs, tabulating the 32 values as complementarity scores. One way to populate such table is to consider all canonical triplets to represent a match and everything else a mismatch. Because the ability to form Hoogsteen bonds depends partly on the environment of the given nucleotide, we took a semi-empirical approach, giving all canonical triplets a match score of 2, scanning triplex literature for examples of less usual triplets and giving those a score of 1, while all other combinations are scored as a mismatch (Table 1). Other approaches leading to a better scoring scheme are certainly possible, but beyond the scope of this article.

4.2 Triplex loop detection

The algorithm introduced in this section has been designed to detect the best candidates for triplex formation. To avoid the inclusion of free-strand and loop nucleotides into the overall score for a particular triplex (because these nucleotides do not participate in Watson–Crick or Hoogsteen base pairing), our calculations use a technique composed of a combination of local and global alignment.

In terms of the DP matrix, potential loops always begin at the main antidiagonal, extending up to $l_{loop_{max}}$ (user-defined algorithm parameter), using Equation (1) to calculate new values. The first $2l_{loop_{max}}$ antidiagonals are therefore calculated by a technique similar to the one used in Smith–Waterman local sequence alignment. In this part, we allow the score of a growing triplex to grow or decline. However, if the density of errors is high enough to bring the score into the negative territory (potential loop occurrence), we do not allow the score to become negative.

Once the calculations exit the area of a potential loop, the calculations continue in a global alignment mode. This way the algorithm can detect high-quality triplex candidates without considering errors that fall within potential loops.

4.3 Triplex detection

The best triplexes in the DP matrix can be identified as those reaching the highest score. To allow detection of such *high scoring segments* (HSS) during the calculation, we use a technique similar to the one used in the BLAST program. Once the score rises above a preset threshold value, the region responsible for the score is considered a potential triplex. The score is monitored (allowed both to increase and decrease) until it falls below a preset threshold. The sequence from the beginning (the first antidiagonal) up to the maximum score becomes the HSS of the potential triplex (Fig. 3).

A number of filtration mechanisms can be applied to the step of HSS segment detection. One of the problems we had to deal with (causing false HSS detection), was the transfer of scores from neighboring diagonals.

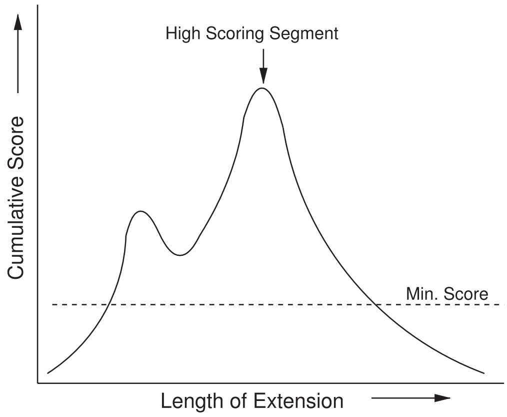


Fig. 3. Detection of high scoring segments.

In the presence of a high-quality triplex sequence, neighboring diagonals adopt its high score by introduction of an extra insertion or deletion. We therefore check for such cases and only report genuine HSS scores and not the neighboring derivatives.

Further filtration is carried out based on statistical significance of the results, eliminating all short or low-quality potential triplexes below a user-defined E-value or P-value threshold (see Section 5 for details on P-value calculations on experimental datasets). A pair of filtering programs (prefilter_gff.c and filter_gff.c, see Supplementary Material) were used to filter out results not supporting a local score maximum (meaning there is a better result nearby).

4.4 Time and space complexity

Time complexity: the calculation of the entire triangle of the DP matrix has $n^2/2$ steps. However, when analyzing real or random sequences, the likelihood of finding a potential triplex decreases with its length (see section 5 for a detailed description of this effect). Therefore, for most practical purposes we only need to evaluate a limited number of antidiagonals, say $2l$, where l is the maximal length of detected triplexes. Time complexity thus becomes $O(2ln)$.

Space complexity: with respect to data dependencies, only the values for the last two antidiagonals are necessary for calculation. Thus, the space complexity of our algorithm is $O(2n)$.

Both simplifications/efficiency enhancements used to derive the time and space complexities allow us to easily extend the algorithm to perform an *incremental calculation*. If upon completion of the calculation we find that the number of antidiagonals was not sufficient, leaving several potential triplexes unresolved, we can pick up the score values from the last two diagonals and continue in the calculations in another $2l$ antidiagonals.

5 RESULTS AND DISCUSSION

We subjected the algorithm to increasing levels of scrutiny to verify the validity of our searching procedures, fine-tune some of the parameters and establish the biological relevance of selected results.

Initial experiments were directed towards establishing reasonable mismatch and insertion/deletion penalties. The penalties have to be high enough to allow for a negative average score per triplet (Korf *et al.*, 2003). Without any rigorous optimization, we found the combination *mismatch* −7, *insertion* or *deletion* −9, *no_isomorphism* −5 to fulfill these criteria and work reasonably well on all sequences.

Table 2. The results of fitting an extreme value distribution function to score distribution data obtained from randomized sequences of *E.coli* and human genomes

Randomized sequence data	λ	μ	Threshold
Escherichia coli	0.91	6.00	20
Human chr5	0.84	6.28	21

The threshold shown here for reference purposes is the score above which <10 sequences were found in randomized data. Precise E-values and P-values can be calculated from values of λ and μ according to Equation (2).

Identification of a higher number of potential triplexes in real-world sequences compared with random and randomized sequences is the first confirmation that the patterns we are collecting using this approach are not random, but rather specific combinations with a possible function that are less frequent in random sequences.

For a rigorous test of non-randomness of the identified candidates, we tested our implementation of the algorithm against a set of 4.7 Mb DNA sequences from *E.coli* and human genomes, their randomized version and a triplex-seeded randomized *E.coli* genome (see Section 3). For each of the sequences, we used the program to identify all potential triplexes and their scores. Since an incrementally detected triplex-forming sequence must obey similar rules as an incrementally growing sequence alignment (only with different base pairing rules), we would expect the obtained scores to obey an extreme value distribution described by Altschul *et al.* (1994).

$$P(S>x)=1-e^{-e^{-\lambda\times(x-\mu)}} \tag{2}$$

We used a maximum likelihood method described by Eddy (1997) to fit our scores to this function. The resulting values of λ and μ are given in Table 2. Figure 4 shows a graphical representation and corresponding parameter values of triplex scores for the different datasets used. Clearly, randomized sequences have a lower content of high-scoring sequence patterns. Also, human sequences seem to be richer in potential triplex-forming sequences, comparable in density to the artificially seeded *E.coli* sequence with one triplex sequence per every 10 000 bp.

We used the λ and μ values to derive statistical thresholds for searching (Table 2). These are different for parallel and antiparallel triplexes, since the two use a different similarity matrix, resulting in different score distributions.

Next, we analyzed the non-B DNA database triplex predictions (Cer *et al.*, 2010) and our triplex-seeded sequence containing 418 inserted triplexes with artificial mismatches and insertions. Our program preferentially recovered the positions of known triplex sequences. Figure 5 shows sensitivity, specificity, accuracy and F_2 measure for these two sets. F measure is the harmonic mean of sensitivity and specificity. F_2 measure is its commonly used modification, which gives higher priority to recall. F_2 measure values >40% are satisfactory, given that 100% of potential triplexes are recovered with a P-value better than 0.01. Some loss of performance on triplex-seeded data is understandable, since mismatches and insertions/deletions were introduced in sequences as short as 6 bp.

One of the detected sequences, is a well-studied triplex from human metallothionein-I promoter (Bacolla and Wu, 1991). This sequence was the second highest-scoring sequence in the

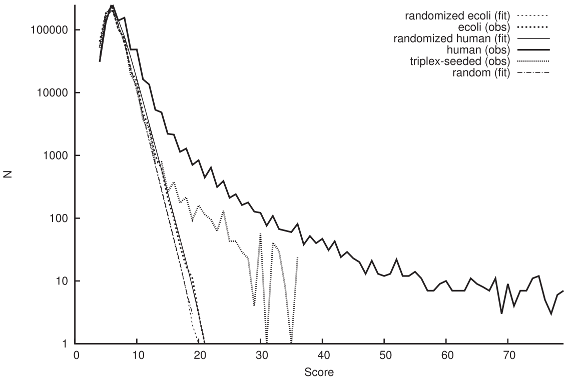


Fig. 4. Log-scale extreme value distribution functions for *E.coli* (dashed line), human (solid line) and triplex-seeded datasets (dotted line) compared with background random sequences (thin lines), including a random sequence, randomized *E.coli* and human sequences. A maximal likelihood fit to the random sequences is available in Table 2. While the *E.coli* genome contains potential triplex sequences only slightly above background levels, the human genome seems to be rich in such sequences with density similar to the triplex-seeded dataset.

triplex-seeded data, scoring 34 with a *P*-value of 5.10^{-9}. Interestingly, we detected two high-scoring subsequences within the MT-I promoter potential triplex, supporting the view of Bacolla and Wu (1991) and Becker and Maher (1998) that alternative triplex structures may be formed at this specific site.

For an alternative evaluation of the validity of our algorithm, we analyzed the *E.coli* genome for triplex-forming sequences and compared the results with those described in Hoyne *et al.* (2000). They searched for potential intrastrand triplex (PIsT). The PIsT element requires the consecutive occurrence of all three triplex-forming blocks of nucleotides, while potential intramolecular triplex (PImT) element requires the consecutive occurrence of just two triplex-forming blocks (the third block is provided by the parallel strand). Thus, every PIsT element by definition contains also a PImT element.

For each of the 25 PIsT elements presented in Hoyne *et al.* (2000), we are able to identify the corresponding PImT element in *E.coli* genome with appropriate parameter settings. The score of these elements range from the value of 6 to the value of 20 and the corresponding *P*-values vary from 4.7×10^{-1} to 2.9×10^{-6}. The best potential triplex element in *E.coli* genome found by our algorithm scored 21 with a *P*-value of 1.2×10^{-6}.

Finally, we examined some of the identified potential triplex sites for biological relevance. Producing a GFF file with results enabled us to view them in the UCSC Genome Browser. Here, we noticed a possible relationship to known *E.coli* genes. To test this, we counted the number of predicted triplexes falling within genes, outside genes or <100 bp from gene boundaries (Fig. 7A). We also calculated the number of predicted triplexes occurring at different distances from the closest gene (Fig. 6) and calculated the ratio of this value to randomly placed positions. There seems to be some preference for potential triplexes to occur in the −50 to −160 region of known genes (Fig. 7B). Given the relatively high *P*-value at which this effect was still visible, it is possible that it is not directly related to the presence of triplexes, but rather a result of shared sequence characteristic between triplexes and regulatory sequences, such as their underlying palindromic nature.

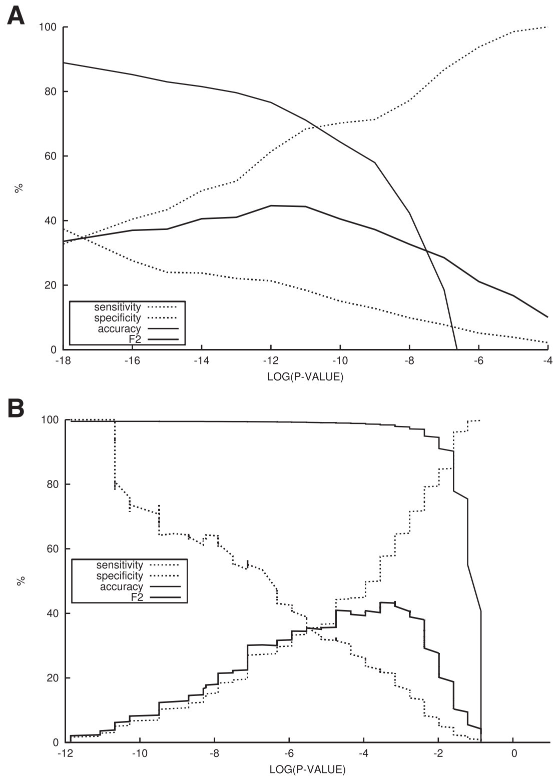


Fig. 5. Sensitivity, specificity, accuracy and F_2 measure calculated for (**A**) the non-B DNA database (Cer *et al.*, 2010); (**B**) the triplex-seeded dataset. The figure shows that the best matches obtained with the described algorithm and settings are entirely made up of the seeded sequences. At lower *P*-values, we start picking up some sequences from the background sequence; acceptable results before accuracy drops sharply are achieved for *P*-values of $<1.0\times10^{-2}$.

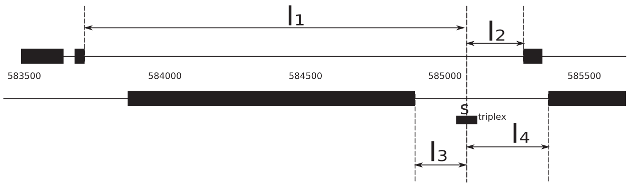


Fig. 6. The definition of the closest gene as used in the numerical experiment. For each triplex we identified its center *S* (rounded up for even triplexes), and calculated the distances l_1, l_2, l_3 and l_4 to the closest upstream and downstream gene borders on both DNA strands. The minimum of these four values was used.

Another observation showed these positions to be clustered at boundaries of evolutionarily poorly conserved regions. A quick literature search revealed a possible connection. Non-B DNA structures are likely to pose a physical barrier to transcriptional apparatus, causing possible transcriptional arrest at such sites (Young *et al.*, 1991). Transcriptional arrest has been directly linked to increased mutation rate (Belotserkovskii *et al.*, 2007), which could explain some aspect of the above-mentioned positioning in genomes.

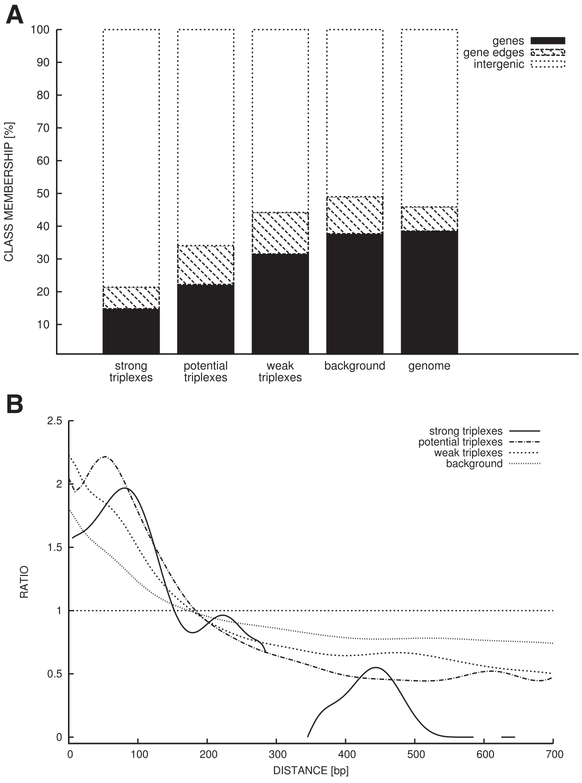


Fig. 7. Graphs showing how potential triplexes identified by the program are positioned in respect to genes in *E.coli*. (**A**) The percentage of triplexes in the results falling inside genes, intersecting with a gene or falling within intergenic segments of the genome. Bars are shown for results of decreasing specificity (from left to right); (**B**) the relative abundance of high-scoring sequences at different distances from nearby genes (relative to randomly placed positions). Both figures were generated after applying the following cutoffs to the results: top 122 (strong triplex), top 1391 (potential triplex), top 15300 (weak triplex), top 106623 (background) and random selection of positions (genome).

While the main purpose of this article is to present the algorithm itself, a more detailed analysis of the best parameter settings and performance with specific DNA sequences is needed to further increase confidence in this kind of sequence analysis.

Because of the increased complexity of scoring, the outlined procedure for scoring individual triplets within the DP matrix cannot be easily extended to take advantage of suffix arrays as is done with palindromes, to further speed up computation.

Overall, we consider it an advantage that triplex identification can be mapped to a well-researched family of DP algorithms and possibly take advantage of approaches aimed originally at other problems, such as sequence alignment.

6 CONCLUSION

We present a novel approach to identifying triplex-forming sequences in genomes and other DNA sequence data. The approach is presented in the form of an algorithm based on previously published algorithms for detection of palindromes. The novelty stems from the adaptation of DP for use with triplexes instead of relying on simpler identification of homopurine and homopyrimidine tracts, which are most appropriate for detection of perfect triplexes. We implemented our algorithm as a program written in C, using a reasonable set of parameters based on published data. The test runs of this program are encouraging, suggesting that the algorithm can provide high speed searches with increased sensitivity for approximate triplex-forming sequences.

Funding: Grants (No. 204/08/1560 and No. 301/10/2370) from the Czech Grant Agency; MSMT Research Grant no. (0021630528), Security-Oriented Research in Information Technology; BUT grants (FIT-S-11-1), Advanced secured, reliable and adaptive IT and (FIT-S-11-2), Recognition and presentation of multimedia data; ASCR Institutional Research Plans (No. AV0Z50040507 and No. AV0Z50040702).

Conflict of Interest: none declared.

REFERENCES

Altschul,S.F. *et al.* (1994) Issues in searching molecular sequence databases. *Nat. Genet.*, **6**, 119–129.

Bacolla,A. and Wu,F.Y-H. (1991) Mung bean nuclease cleavage pattern at a polypurine-polypyrimidine sequence upstream from the mouse metallothionein-I gene. *Nucleic Acids Res.*, **1**, 1639–1647.

Beal,P.A. and Dervan,P.B. (1992) The influence of single base triplet changes on the stability of a pur.pur.pyr triple helix determined by affinity cleaving. *Nucleic Acids Res.*, **20**, 2773–2776.

Becker,N.A. and Maher, L.J. III (1998) Characterization of a polypurine/polypyrimidine sequence upstream of the mouse metallothionein-I gene. *Nucleic Acids Res.*, **26**, 1951–1958.

Belotserkovskii,B.P. *et al.* (2007) A triplex-forming sequence from the human c-MYC promoter interferes with DNA transcription. *J. Biol. Chem.*, **282**, 32433–32441.

Cer,R.Z. *et al.* (2010) Non-B DB: a database of predicted non-B DNA-forming motifs in mammalian genomes. *Nucleic Acids Res.*, **39**, D383–D391.

Dayn,A. *et al.* (1992) Intramolecular DNA triplexes: unusual sequence requirements and influence on DNA polymerization. *Proc. Natl Acad. Sci. USA*, **89**, 11406–11410.

Dixon,B.P. *et al.* (2008) RecQ and RecG helicases have distinct roles in maintaining the stability of polypurine.polypyrimidine sequences. *Mutat Res.*, **643**, 20–28.

Eddy,S.R. (1997) Maximum likelihood fitting of extreme value distributions. *Technical Report*. Available at ftp://selab.janelia.org/pub/publications/Eddy97b/Eddy97b-techreport.pdf (last accessed date August 07, 2011).

Frank-Kamenetskii,M.D. and Mirkin,S.M. (1995) Triplex DNA structures. *Annu. Rev. Biochem.*, **64** 65–95.

Gaddis,S.S. *et al.* (2006) A web-based search engine for triplex-forming oligonucleotide target sequences. *Oligonucleotides*, **16**, 196–201.

Ghosal,G. and Muniyappa,P. (2006) Hoogsteen base-pairing revisited: resolving a role in normal biological processes and human diseases. *Biochem. Biophys. Res. Commun.*, **343**, 1–7.

Goni,J.R. *et al.* (2004) Triplex-forming oligonucleotide target sequences in the human genome. *Nucleic Acids Res.*, **32**, 354–360.

Gowers,D.M. and Fox,K.R. (1998) Triple helix formation at (AT)$_n$ adjacent to an oligopurine tract. *Nucleic Acids Res.*, **26**, 3626–3633.

Haasnoot,C.A.G. *et al.* (1986) On loop folding in nucleic acid hairpin-type structures. *J. Biomol. Struct. Dyn.*, **3**, 843–857.

Hanvey,J.C. *et al.* (1988) Intramolecular DNA triplexes in supercoiled plasmids. *Proc. Natl Acad. Sci. USA*, **85** 6292–6296.

Hoyne,P.R. *et al.* (2000) Searching genomes for sequences with the potential to form intrastrand triple helices. *J. Mol. Biol.*, **302**, 797–809.

James,P.L. *et al.* (2003) Thermodynamic and kinetic stability of intermolecular triple helices containing different proportions of C+·GC and T·AT triplets. *Nucleic Acids Res.*, **31**, 5598–5606.

Jenjaroenpun,P. and Kuznetsov,V.A. (2009) TTS Mapping: integrative WEB tool for analysis of triplex formation target DNA sequences, G-quadruplets and non-protein coding regulatory DNA elements in the human genome. *BMC Genomics*, **10** (Suppl. 3), S9.

Kang,S.M. *et al.* (1992) Metal ions cause the isomerization of certain intramolecular triplexes. *J. Biol. Chem.*, **267**, 1259–1264.

Kinniburgh,A.J. (1989) A cis-acting transcription element of the c-myc gene can assume an H-DNA conformation. *Nucleic Acids Res.*, **17**, 7771–7778.

Knauert,M.P. and Glazer,P.M. (2001) Triplex forming oligonucleotides: sequence-specific tools for gene targeting. *Hum. Mol. Genet.*, **10**, 2243–2251.

Korf,I. *et al.* (2003) *BLAST*. O'Reilly & Associates, Inc., Sebastopol, 368 pages.

Landau,G.M. and Vishkin,U. (1989) Fast parallel and serial approximate string matching. *J. Algorithms*, **10**, 157–169.

Manning,C.D. *et al.* (2008) *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, 496 pp.

Mergny,J.L. *et al.* (1991) Sequence specificity in triple helix formation: experimental and theoretical studies of the effect of mismatches on triplex stability. *Biochemistry*, **30**, 9791–9798.

Mirkin,S.M. and Frank-Kamenetskii,M.D. (1994) H-DNA and related structures. *Annu. Rev. Biophys. Biomol. Struct.*, **23**, 541–576.

Perez,A. *et al.* (2007) Refinement of the AMBER force field for nucleic acids: improving the description of α/γ conformers. *Biophys. J.*, **92**, 3817–3829.

Plum,G.E. *et al.* (1995) Nucleic acid hybridization: triplex stability and energetics. *Annu. Rev. Biophys. Biomol. Struct.*, **24**, 319–350.

Raghavan,S.C. and Lieber,M.R. (2007) DNA structure and human diseases. *Front. Biosci.*, **12**, 4402–4408.

Raghavan,S.C. *et al.* (2005) Evidence for a triplex DNA conformation at the bcl-2 major breakpoint region of the t(14;18) translocation. *J. Biol. Chem.*, **280**, 22749–22760.

Rathinavelan,T. and Yathindra,N. (2006) Base triplet nonisomorphism strongly influences DNA triplex conformation: effect of nonisomorphic G* GC and A* AT triplets and bending of DNA triplexes. *Biopolymers*, **82**, 443–461.

Rippe,K. *et al.* (1992) Alternating d(G-A) sequences form a parallel-stranded DNA homoduplex. *EMBO J.*, **11**, 3777–3786.

Roberts,R.W. and Crothers,D.M. (1991) Specificity and stringency in DNA triplex formation. *Proc. Natl Acad. Sci. USA*, **88**, 9397–9401.

Schroth,G.P. and Ho,P.S. (1995) Occurrence of potential cruciform and H-DNA forming sequences in genomic DNA. *Nucleic Acids Res.*, **23**, 1977–1983.

Seidman,M.M. and Glazer,P.M. (2003) The potential for gene repair via triple helix formation. *J. Clin. Invest.*, **112**, 487–494.

Soyfer,V.N. and Potaman,V.N. (1995) *Triple-Helical Nucleic Acids*. Springer, Heidelberg, 360 pp.

Tan,Z.J. and Chen,S.J. (2006) Nucleic acid helix stability: effects of salt concentration, cation valence and size, and chain length. *Biophys. J.*, **90**, 1175–1190.

Thenmalarchelvi,R. and Yathindra,N. (2005) New insights into DNA triplexes: residual twist and radial difference as measures of base triplet non-isomorphism and their implication to sequence-dependent non-uniform DNA triplex. *Nucleic Acids Res.*, **33**, 43–55.

Walter,A. *et al.* (2001) Evidence for a DNA triplex in a recombination-like motif: I. Recognition of Watson-Crick base pairs by natural bases in a high-stability triplex. *J. Mol. Recognit.*, **14**, 122–139.

Wang,G. and Vasquez,K.M. (2004) Naturally occurring H-DNA-forming sequences are mutagenic in mammalian cells. *Proc. Natl Acad. Sci. USA*, **101**, 13448–13453.

Xodo,L.E. *et al.* (1993) Sequence-specific DNA-triplex formation at imperfect homopurine-homopyrimidine sequences within a DNA plasmid. *Eur. J. Biochem.*, **212**, 395–401.

Young,S.L. *et al.* (1991) Triple helix formation inhibits transcription elongation in vitro. *Proc. Natl Acad. Sci. USA*, **88**, 10023–10026.

Zhao,J. *et al.* (2010) Non-B DNA structure-induced genetic instability and evolution. *Cell Mol. Life Sci.*, **67**, 43–62.

D Hon et al., 2017

Bioinformatics, 33(21), 2017, 3373–3379
doi: 10.1093/bioinformatics/btx413
Advance Access Publication Date: 4 July 2017
Original Paper

Sequence analysis

pqsfinder: an exhaustive and imperfection-tolerant search tool for potential quadruplex-forming sequences in R

Jiří Hon[1], Tomáš Martínek[1], Jaroslav Zendulka[1] and Matej Lexa[2,]*

[1]IT4Innovations Centre of Excellence, Faculty of Information Technology, Brno University of Technology, 61266 Brno, Czech Republic and [2]Department of Information Technology, Faculty of Informatics, Masaryk University, 60200 Brno, Czech Republic

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on April 20, 2017; revised on June 6, 2017; editorial decision on June 19, 2017; accepted on June 23, 2017

Abstract

Motivation: G-quadruplexes (G4s) are one of the non-B DNA structures easily observed *in vitro* and assumed to form *in vivo*. The latest experiments with G4-specific antibodies and G4-unwinding helicase mutants confirm this conjecture. These four-stranded structures have also been shown to influence a range of molecular processes in cells. As G4s are intensively studied, it is often desirable to screen DNA sequences and pinpoint the precise locations where they might form.

Results: We describe and have tested a newly developed Bioconductor package for identifying potential quadruplex-forming sequences (PQS). The package is easy-to-use, flexible and customizable. It allows for sequence searches that accommodate possible divergences from the optimal G4 base composition. A novel aspect of our research was the creation and training (parametrization) of an advanced scoring model which resulted in increased precision compared to similar tools. We demonstrate that the algorithm behind the searches has a 96% accuracy on 392 currently known and experimentally observed G4 structures. We also carried out searches against the recent G4-seq data to verify how well we can identify the structures detected by that technology. The correlation with *pqsfinder* predictions was 0.622, higher than the correlation 0.491 obtained with the second best G4Hunter.

Availability and implementation: http://bioconductor.org/packages/pqsfinder/ This paper is based on pqsfinder-1.4.1.

Contact: lexa@fi.muni.cz

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

DNA sequences capable of forming alternative secondary structures, called non-B DNA, have long been at the center of research interest because of their possible biological functions (Du *et al.*, 2013) and their involvement in mutagenesis and disease (Bacolla and Wells, 2009). Instead of forming canonical B-DNA helices with Watson-Crick base pairing, these regions of DNA can engage in different types of base pairing and form cruciforms, triplexes (or H-DNA), G-quadruplexes (G4s), i-motifs and a few other alternative structures (Wells, 2007). After previous work on algorithms and practical solutions to identify triplex DNA (Hon *et al.*, 2013; Lexa *et al.*, 2011), we focus here on identifying potential quadruplex-forming sequences (PQS).

As evidenced by sequencing (Chambers *et al.*, 2015), as well as a large number of other experimental and *in silico* studies, PQS are found in high numbers in eukaryotic genomes (Huppert, 2005; Lexa et al., 2014). They are implicated in several genome-wide processes, mostly as positive or negative regulators of transcription (Rhodes and Lipps, 2015), negative regulators of replication which require specialized helicases for the processes to continue (Mendoza et al., 2016) and may be dispersed into critical locations of the genome by the activity of transposable elements (Kejnovsky and Lexa, 2014).

Today, several software tools for identification of PQS in biological sequences are available. The oldest and most commonly used algorithms are based on a simple folding rule representing four runs of guanines separated by relatively short loops (or spacers). These include quadparser (Huppert, 2005), QGRS Mapper (D'Antonio and Bagga, 2004; Kikin et al., 2006) and Quadfinder (Scaria et al., 2006). The folding rule used in these tools is usually of the form G{3,6}.{1,8}G{3,6}.{1,8}G{3,6}.{1,8}G{3,6} reflecting the fact that PQS with short loops and four perfect G runs form the most stable G4s *in vitro*. These tools consider only sequences that match the sequence formula perfectly.

In recent years, different *in vitro* experiments have confirmed the existence of imperfect G4s (Mukundan and Phan, 2013). They have also been explored *in silico* by molecular dynamics (Varizhuk et al., 2017). As a result, new tools for prediction of imperfect G4s began to be developed. Such tools include TetraplexFinder/QuadBase2 (Dhapola and Chowdhury, 2016), ImGQfinder (Varizhuk et al., 2014) and G4Hunter (Bedrat et al., 2016). For example, TetraplexFinder considers potential bulges of defined length in runs of three guanines, while ImGQfinder considers the possibility of a single bulge or mismatch in a wider variety of guanine run lengths. Finally, G4Hunter does not define individual defect types, but uses a simple encoding and statistics over a sliding window, that can accomodate different types of defects.

It has also been discovered that a given DNA segment (sequence) can form several overlapping G4s, by definition mutually exclusive, where individual nucleotides in the sequence compete with each other for binding via Hoogsteen bonds (Agrawal et al., 2014). In these cases, it is very useful to have a tool for predicting all overlapping instances and evaluate them with scores that correlate with the propensity for G4 formation. The only tool predicting overlapping G4s and at the same time capable of assigning scores to their individual instances is QGRS Mapper. Its score function considers the number of Gs in each run, loop lengths as well as the difference in loop lengths. Features of existing software tools for PQS identification are summarized in Table 1.

In this paper, we introduce an R package and the underlying algorithm for PQS detection that addresses certain shortcomings of the available tools.

Five main ideas projected into the package functioning are to: (i) allow imperfections in PQS as mismatches or bulges in G runs and excessively long loops between the G runs, (ii) provide a PQS score that is closely related to G4 stability, (iii) give the user a choice between reporting all overlapping PQS and/or only the locally best, (iv) provide the overall number (density) of possible PQS conformations covering each position in the input sequence and (v) allow users to define their own criteria for matching and scoring, overriding the defaults determined by calculations in this paper.

The package and the algorithm were called *pqsfinder* and accepted into Bioconductor (Huber et al., 2015) in April 2016. Here, we explain how the ideas were implemented in the package and apart from tuning its default parameters and settings, we show how *pqsfinder* predictions relate to recently carried out G4 sequencing (also called G4-seq or G-seq) (Chambers et al., 2015).

2 Approach and algorithm

The main principle of the algorithmic approach presented here is based on the fact that monomolecular G4 structures arise from compact sequence motifs composed of four consecutive and possibly imperfect guanine runs (G runs) interrupted by loops of semi-arbitrary lengths.

The algorithm first identifies four consecutive G run sequences (G run quartet). Subsequently, it examines the potential of such G run quartet to form a stable G4 and reports a corresponding quantitative score.

The *pqsfinder* algorithm can be divided into three logical steps: (i) identification of all possible G run quartets, (ii) score assignment and (iii) overlap resolution. All three parts are described in the following sections.

2.1 Identification of all possible G run quartets

The first G run is matched freely in the sequence by a regular expression G{1,10}.{0,9}G{1,10} with limited minimal and maximal length. This regular expression allows us to match imperfect G runs containing both mismatches and bulges while requiring at least two guanines. The remaining three G runs are matched by the same regular expression with the following additional constraints: (i) each subsequent G run must lie beyond the 3'-end of the previous one (no overlap), (ii) the distance of each G run to the previous G run must be in the range of minimal and maximal loop length and at most one loop is allowed to have zero length (Marusic et al., 2013) and (iii) each G run has to fit in a sequence window defined by the first G run starting position and the user-defined maximal PQS length. These constraints are summarized in Figure 1.

As regular expressions are able to capture only one match (usually the maximal one), to list all possible combinations we use a backtracking approach. After four initial G runs are matched and processed, the last successfully matched G run is shortened by one

Table 1. Feature comparison of existing tools for PQS identification

Name	Model	Overlaps	Imperf.	Score	Avail.
quadparser	Folding rule	✓	✗	✗	✗
QGRS Mapper	Folding rule	✓	✗	✓	Web
Quadfinder	Folding rule	✓	✗	✗	✗
ImGQfinder	Folding rule	✓	✓[a]	✗	Web
TetraplexFinder	Regular expression	✓	✓[b]	✗	Web
G4Hunter	Sliding window	✗[c]	✓	✓	R script

[a] ImGQfinder allows at most one imperfection.

[b] TetraplexFinder supports only bulges of fixed length between 0 and 7.

[c] G4Hunter model inherently merges overlapping and neighbouring PQS. For this reason, the boundaries of individual PQS are not well-defined.

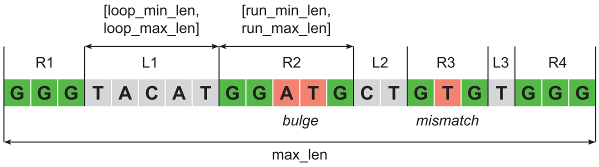


Fig. 1. PQS constraints. Every PQS consists of two types of elements: G runs (R1–4) and loops (L1–3). The minimal and maximal length of each element type is constrained by the corresponding options depicted in the picture as well as the overall PQS length. All these options can be freely customized when using the *pqsfinder* package

nucleic acid base from the end and if it is still a valid G run, the algorithm proceeds normally to scoring and overlap resolution. On the other hand, if the shortened G run is not valid, the algorithm tracks back to the previous successfully matched G run and applies the same shortening modification. In this case, if the modified G run is valid, the algorithm proceeds to match all the following G runs again. Once the backtracking procedure gets to the first G run and finds its shortened variant to be invalid, the whole process of G run identification is rerun from position one after the starting position of the first G run. The backtracking procedure increases the computational complexity of the search, but allows us to rigorously model the competition between overlapping PQS.

2.2 Score assignment

The *pqsfinder* scoring scheme was designed to quantitatively approximate the relationship between G4 sequence and the stability of its structure. While the scoring function is purely empirical, we intentionally chose an approach where the score is modular and, obtained by addition of scores representing the binding affinities of smaller regions within the G4. This kind of approach has already been proven to work for simpler DNA structures, such as nucleic acid duplexes and hairpins. (SantaLucia, 2012; Zuker, 2003)

The first part of the scoring scheme quantifies the quality of individual G runs. It awards the PQS a score for each G-tetrad stacking and penalizes mismatches and bulges in G runs.

The scoring is then defined by Equation 1, where N_t is the number of tetrads, B_t is a G-tetrad stacking bonus, N_m is the number of inner mismatches, P_m is mismatch penalization, N_b is the number of bulges, P_b is bulge penalization, F_b is bulge length penalization factor, L_{bi} is the length of the i-th bulge and E_b is bulge length exponent.

$$S_r = (N_t - 1)B_t - N_m P_m - \sum_{i=1}^{N_b} P_b + F_b L_{bi}^{E_b} \tag{1}$$

However, discrimination between bulges and mismatches can be a demanding task requiring multiple sequence alignment. To avoid this, we made two simplifying assumptions that allowed us to efficiently analyze bulges and mismatches by only counting lengths of G runs and their G content. First, we require at least one G run to be perfect (consisting of just guanines). Second, we limit the number of imperfections to one per G run. Based on the available literature, we consider bulges and long loops to be strong destabilizers of G4s and do not expect more than a few of these imperfections to be possible at the same time.

In the scoring procedure, a perfect G run is taken as a reference and other G runs are assessed relatively to the reference. A G run is classified as mismatched, if it has the same length as the reference and the G content lower by one. When a G run has a greater length than the reference and at least the same G content, it is classified as bulged. Finally, all G runs can only be either perfect, mismatched or bulged. Other cases are considered to be invalid G runs. When there are multiple perfect G runs present, the shortest one is used as the reference.

The second part of the scoring scheme quantifies the destabilizing effect of the loops on G4 stability. At this time we have no mechanistic understanding of possible loop sequence and length effects. Hence, we limit ourselves to an empirical formula that can accommodate some of the observations made by Guédin *et al.* (2010). Loop length mean L_m is multiplied by the factor F_m and raised to the power of E_m. Complete scoring function is then expressed by Equation 2.

$$S = \max(S_r - F_m L_m^{E_m}, 0) \tag{2}$$

F_m and E_m are numerical parameters that empirically model the relationship between loop lengths and their destabilization effects on the quadruplex. These permit a non-linear relationship, while their values are derived by fitting the model to experimental results (see Section 4). S_r is the value from Equation 1.

2.3 Overlap resolution

The overlap resolution is an iterative process that is designed to always prefer dominant PQS. First, all PQS sharing the highest obtained score are selected (in subsequent iterations, PQS sharing the highest remaining score are used). Second, the selected PQS are processed one by one in the order of their increasing starting position as follows: (i) if the current PQS overlaps the previous PQS, the current PQS is removed, (ii) if the current PQS is completely included in the previous PQS, the previous PQS is removed. Third, all lower-scoring PQS overlapping with any of the remaining selected PQS are discarded. Fourth, all selected PQS are reported and removed. Fifth, the next iteration begins again with the remaining PQS. Iterations continue until all PQS are checked (either reported or removed).

We implemented the process above effectively in order to reduce the memory usage. The main optimization idea is to run the iterative process progressively as the identification algorithm proceeds through the sequence. As a result, only a small set of recently identified overlapping PQS has to be in memory.

3 Implementation

The *pqsfinder* package was created following recommended practices for R/Bioconductor packages and all functions are well-documented within the inline R documentation system. A detailed user guide with convenient examples was also prepared as a package vignette. Source code is written in both R and C++, each having its own important role in the package architecture.

The R code implements the interface that is needed for a seamless user interaction within the Bioconductor framework, relying on the following R packages: Biostrings (Pagès *et al.*, 2016a), GenomicRanges, IRanges (Lawrence *et al.*, 2013), S4Vectors (Pagès *et al.*, 2016b), Rcpp (Eddelbuettel and François, 2011) and BH (Eddelbuettel *et al.*, 2016). The package provides one main function *pqsfinder* for running the PQS search algorithm and several secondary functions that operate on the search results.

The central data structure for results is the *PQSViews* class which is derived from the *XStringViews* class from the Biostrings package. It maintains the sequence coordinates of the identified PQS along with other useful metadata: (i) score, (ii) strand, (iii) number of tetrads, (iv) number of bulges, (v) number of mismatches and (vi) loop lengths.

Table 2. Overview of *pqsfinder* options

Group	Name	Description
Filters	*strand*	Strand symbol: +, – or * (both).
	overlapping	Enables overlapping PQS.
	max_len	Maximal PQS length.
	min_score	Minimal PQS score.
	run_min_len	Minimal G run length.
	run_max_len	Maximal G run length.
	loop_min_len	Minimal loop length.
	loop_max_len	Maximal loop length.
	max_bulges	Maximal number of bulges.
	max_mismatches	Maximal number of mismatches.
	max_defects	Maximal number of all defects.
Scoring	*tetrad_bonus*	G-tetrad stacking bonus B_t.
	mismatch_penalty	Inner mismatch penalization P_m.
	bulge_penalty	Bulge penalization P_b.
	bulge_len_factor	Bulge length penal. factor F_b.
	bulge_len_exponent	Bulge length penal. exponent E_b.
	loop_mean_factor	Loop mean penal. factor F_m.
	loop_mean_exponent	Loop mean penal. exponent E_m.
Advanced	*run_re*	G run regular expression.
	custom_scoring_fn	User-defined scoring function.
	use_default_scoring	Enables internal scoring system.
	verbose	Enables detailed text output.

This aside, the *PQSViews* object provides access to two additional vectors. The first is a *density* vector—for each sequence position it gives the number of different PQS conformations overlapping that position. The second vector *maxScores* reports the PQS quality along the sequence—for each sequence position it gives the maximal score of all PQS overlapping that position. We consider these two vectors particularly useful as additional information to the exact PQS coordinates and metadata. The *density* and *maxScores* vectors can be easily used to discriminate low-complexity regions (full of guanines) that inherently allow a large amount of folded PQS conformations from regions that on the other hand contain a singular high-scoring PQS.

The main PQS search logic is implemented purely in the C++ language for speed since the algorithm is based on an exhaustive search of the PQS topological space and it is computationally intensive by definition. The Rcpp library was used to easily link the C++ code with R scripts. We also employed the Boost regular expression library (Maddock, 2016) to match individual G runs. However, we soon realized that the general regular expression engine has a significant overhead and is too slow for our needs. For this reason, we implemented an optimized matching function for the default G run regular expression. At the same time, we are linking the Boost library for the case where users would like to use their own definition of a G run using an alternative regular expression.

3.1 Customization

Since we strongly support the Bioconductor goal to further scientific understanding by producing extensible, scalable and interoperable software, we designed *pqsfinder* to be easily customizable. The users can tweak the algorithm options for their personal needs or test new hypotheses about PQS conformations and develop novel innovative scoring schemes. Supported options are divided into three logical groups: (i) filters, (ii) scoring and (iii) advanced (see Table 2).

Filter options control the main algorithmic constraints (see Fig. 1). These have great impact on the algorithm sensitivity and speed. All PQS that do not satisfy the basic constraints are excluded immediately and do not proceed further to the scoring step.

Scoring options include all the constants that appear in the scoring Equations 1 and 2. By default, these constants are set to reasonable values as described in the next section and its modification is recommended only to users who would like to bias the scoring systems towards a specific type of G4 or to refine the constants on novel data.

Advanced options allow to get full control over the search algorithm by providing alternative G run regular expression and scoring function. However, the custom scoring function can negatively influence the overall algorithm performance, particularly on long sequences, since there is a significant overhead linked to the calling of custom R function instead of efficient inline C++ implementation. Thus, this feature is recommended only for rapid prototyping of novel scoring techniques, which can be later implemented efficiently in C++ and delivered in the next version of the *pqsfinder* package.

4 Model training

As described in the foregoing section, the scoring model requires several constants to be chosen (see *scoring* group in Table 2). It is, however, very difficult to estimate these parameters. For this reason, we decided to construct a training set from available experimental data and search for a setting that gives the best performance on these data. The dataset construction process and parameter-search algorithm are discussed in detail in this section.

4.1 Existing datasets

Methods for G4 prediction are usually evaluated on a set of experimentally verified (*in vitro*) G4s, extracted from different publications. For example, a recently published method G4Hunter involved collecting a set of 392 experimentally verified G4s consisting of 298 positive and 94 negative samples (later referred to as Lit392).

However, these datasets have several disadvantages: (i) they are unbalanced regarding the number of positive and negative samples, (ii) significant number of items differ only by a single mutation and (iii) datasets are very small and cover only a small proportion of possible G4 conformations given all the possible loop lengths, bulges, mismatches and other defects.

On the other hand, (Chambers *et al.*, 2015) recently published a novel approach for high throughput sequencing of DNA G4 structures called G4-seq. The technique detects noisy sequences that emerge on treatment of DNA samples with K^+ or PDS (pyridostatin, a chemical G4 stabilizer). As a result of this technology, the authors released a track (in BED format) that shows the propensity of reference Human DNA sequence (*hg19*) to form G4s.

This track has two disadvantages. First, it only shows the level of mismatches at given sequence positions that were observed during the sequencing process. Hence, in reality, we have no evidence that a G4 has been formed, but based on the G4-seq method the level of mismatches should show high correlation with the probability that the sequence forms the G4 structure. Second, as the G4 structure is formed during sequencing, the level of mismatches remains high, until the end of the sequenced read, even downstream of the actual G4 structure. As a result, the BED file constructed by mapping the reads onto the reference sequence, can be affected by this ‘memory effect’.

Despite these disadvantages, the G4-seq dataset is extremely valuable, because it shows the G4 structure propensity for the entire human genome and thus it covers many more possible conformations and imperfect structures (including long loops and bulges) than any dataset extracted from the published literature.

Based on these facts we decided to use a subset of G4-seq data for training of the *pqsfinder* scoring model. We then used two additional independent datasets for testing: Lit392 and a different (non-overlapping with training data) subset of G4-seq data. The whole process was operated as follows:

1. We prepared independent training and test sets from G4-seq data.
2. We trained *pqsfinder* parameters on G4-seq training set.
3. We selected those parameters that performed best on the G4-seq training set.
4. Finally, the selected *pqsfinder* parameters were evaluated and compared to other tools on the Lit392 dataset and G4-seq test set.

In the following subsection, individual steps of this procedure are described in more detail.

4.2 Preparation of the training and test sets

From the G4-seq data, we used BED files representing the level of mismatches from two experimental treatments. In the first treatment, the authors stabilized G4s using K^+ while in the second case they used PDS. In both cases, measurements were done on both DNA strands separately resulting in four BED files (two treatments with two strands each).

In the first step, as the K^+ and PDS measurements do not cover 100% of *hg19* genome, we identified only those DNA fragments where both K^+ and PDS measurements were available. Then, we filtered out fragments shorter than 10 kbp and longer fragments were trimmed to 10 kbp. In the next step, we combined K^+ and PDS BED files by calculating the average value from both treatments. Subsequently, we filtered out those fragments that did not include a significant level of mismatches (where the averaged level of mismatches from K^+ and PDS never exceeded threshold 40). In order to eliminate cases where potential G4 overlapped the beginning or the end of the fragment, we also filtered out those fragments that included a significant level of mismatches in the first 30 bp or the last 30 bp of the fragment.

The described procedure was applied to each strand separately. Finally, 1100 fragments were chosen at random, 100 as G4-seq training set (for a total of 1 Mbp) and 1000 as G4-seq test set (for a total of 10 Mbp). Both datasets are available as Supplementary Data.

4.3 Training of scoring parameters on the G4-seq training set

We used the genetic algorithm implemented in the R package GA (Scrucca, 2013) as a method for parameter-space exploration and training. In order to make the exploration process easier, the G-tetrad stacking bonus was fixed at 40. The remaining scoring options were trained. Their names, number of bits allocated in GA chromosome and ranges of values considered are summarized in Table 3. Total GA chromosome length was 33 bits. Other *pqsfinder* options were fixed to the default values.

To evaluate fitness, we calculated Pearson's correlation coefficient between the vector *maxScores* generated by *pqsfinder* (see section 3) and the averaged level of mismatches from K^+ and PDS treatments of G4-seq training set. More specifically, maximal values of the *pqsfinder* score were calculated for all positions of all DNA fragments in the training set and these values were correlated with appropriate positions in the G4-seq training set (experimentally verified level of mismatches). The basic idea behind this fitness function is: the higher the correlation coefficient between *pqsfinder* score and G4-seq mismatch level, the better the prediction of putative G4 structures will be.

Table 3. Trained parameters and their encoding in chromosome

Name	Bits	Range	Step	Result
bulge_penalty	6	0–63	1	20
mismatch_penalty	6	0–63	1	28
bulge_len_factor	5	0–3.1	0.1	0.2
bulge_len_exponent	5	0–3.1	0.1	1
loop_mean_factor	6	3–9.3	0.1	6.6
loop_mean_exponent	5	0–3.1	0.1	0.8

A genetic algorithm was set up with the following parameters: (i) population size 24, (ii) probability of crossover 0.5, (iii) probability of mutation 0.5 and (iv) number of generations 200. During the exploration process, we used monitor function and recorded 1157 unique combinations of parameters and their fitness values.

As the final parameters, we selected the combination with the maximal fitness value. Concrete values of selected parameters are listed in Table 3 (column *Result*). The table of all explored parameter combinations and their fitness values is available as Supplementary Data.

5 Results

In the first step, we compared *pqsfinder* to other tools capable to predict whether a given sequence can form a G4 or not. As candidate tools that are still working and available online/offline, we selected: G4Hunter, QGRS Mapper, TetraplexFinder and ImGQfinder. We applied these to a recently published dataset (Bedrat *et al.*, 2016) containing 392 *in vitro* verified G4s (Lit392), originally used to test G4Hunter.

In the next step, we configured and executed the selected tools with the following parameters. (i) *pqsfinder* was executed with the parameters that had the best fitness value on the G4-seq training set. (ii) G4Hunter was executed with the default parameters. (iii) QGRS Mapper was executed with the most relaxed parameters, i.e. minimal G run length was 2, loop length was in the range 0 to 36 and maximal length was 45. As *pqsfinder*, G4Hunter and QGRS Mapper report scores, to calculate accuracy and Matthews correlation coefficient (MCC), we always systematically found a threshold that resulted in the highest possible values for each tool. Interestingly, we found out that for G4Hunter, the threshold 0.71 works even better than thresholds 1.0, 1.2 and 1.5 that are recommended by the authors. (iv) TetraplexFinder was executed with the following combinations of parameters: G run length 2 and 3, greedy and non-greedy approach, bulge length in the range 0 to 7 and maximal loop length 50. Of all possible TetraplexFinder parameter combinations, only the best ones are reported in Table 4. (v) ImGQfinder was executed with G run length in the range 2 to 5, maximal loop length 25 and number of defects 0 and 1. Again, only the best combinations are presented in Table 4.

Finally, for all selected tools and their configurations, we measured basic performance characteristics, namely accuracy (ACC) and Matthews correlation coefficient (MCC). For tools that report a score or allow us to specify a threshold, we also measured the area under the ROC curve (AUC). The results are summarized in Table 4. Since the Lit392 dataset is unbalanced, MCC is the most relevant value. As we can see, *pqsfinder* outperformed other tools significantly.

Table 4. Performance comparison of different tools on Lit392 dataset

Tool	Configuration	ACC	MCC	AUC
pqsfinder	Best on G4-seq training set	0.964	0.902	0.975
G4Hunter	Default	0.952	0.865	0.969
QGRS Mapper	g≥2, $ll = 36$, $l = 45$	0.954	0.872	0.968
TetraplexFinder	$g = 2$, $ll = 50$, gr, $bl = 0$	0.946	0.850	–
TetraplexFinder	$g = 2$, $ll = 50$, ngr, $bl = 0$	0.946	0.850	–
ImGQfinder	$g = 2$, $ll = 25$, $d = 0$	0.941	0.835	–
ImGQfinder	$g = 2$, $ll = 25$, $d = 1$	0.918	0.767	–

Note: The meaning of the configuration options is as follows: *t* is threshold, *g* is G run length, *ll* is maximal loop length, *l* is maximal G4 length, *gr* is greedy approach, *ngr* is non-greedy approach, *bl* is bulge length and *d* is number of defects. For tools that report score (*pqsfinder*, G4Hunter and QGRS Mapper), we systematically determined thresholds that resulted in the highest possible ACC and MCC. We also found that for tools without scoring system (TetraplexFinder and ImGQfinder) it is always better to disable imperfections.

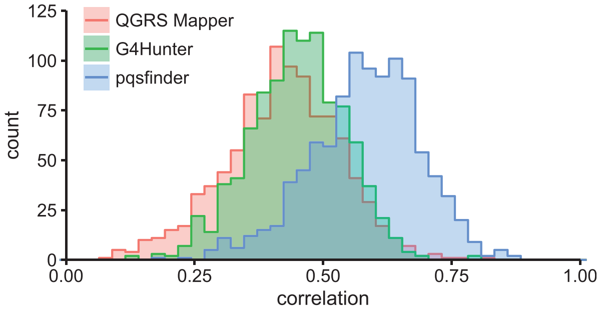


Fig. 2. Histogram of correlation coefficients for QGRS Mapper, G4Hunter and *pqsfinder* on the G4-seq test set fragments. The correlation was measured between the averaged level of mismatches of the G4-seq test set fragments (see Section 4.2) and the vector of maximal scores predicted by each tool. While histograms of QGRS Mapper and G4Hunter correlations are almost the same, the histogram of *pqsfinder* correlations is much more positively skewed

Subsequently, we identified tools capable of predicting overlapping G4s and assigning them a score. Only those tools could also be evaluated on the G4-seq test set. The basic idea behind this test is to calculate all possible overlapping G4s for a given sequence and extract the characteristics of maximal score values (for every sequence position maximal score of all overlapping G4s is selected). Such characteristic can then be correlated with the level of mismatches at the same positions of the G4-seq test set. From the set of available tools, only the QGRS Mapper and G4Hunter met the requirement. As a dataset, we used G4-seq test set consisting of 1000 randomly selected DNA fragments with length of 10 kbp (procedure for dataset construction is described in Section 4.2).

In the next step we configured and executed selected tools with the following parameters: *pqsfinder* was executed with parameters trained on G4-seq training set. G4Hunter was executed with all thresholds between 0 and 4 (with step 0.05). Predicted G4s were refined and merged together. QGRS Mapper was evaluated with the most relaxed parameters as before, i.e. minimal G run length is 2, loop length is in range 0 to 36 and maximal length is 45. For results from each tool, the characteristic of maximal score value was calculated and compared with the G4-seq test set. This comparison was done in two ways. First, Pearson correlation coefficient was calculated for every fragment separately. As the result, we got a distribution of correlation coefficients with individual means and standard deviations (see Fig. 2 and Table 5, columns *CC mean* and *CC SD*). Second, Pearson correlation coefficient was calculated for all fragments joined together to get a single overall value (see Table 5, column *Overall CC*). As we can see, the *pqsfinder* significantly outperformed other tools.

Table 5. Comparison of correlation coefficient (CC) statistics for different tools

Tool	CC mean	CC SD	Overall CC
pqsfinder	0.583	0.106	0.622
G4Hunter	0.450	0.093	0.491
QGRS Mapper	0.422	0.112	0.479

Note: The individual CCs were measured between the averaged level of mismatches of the G4-seq test set fragments (see Section 4.2) and the vector of maximal scores predicted by each tool. Overall CC was calculated between concatenated averaged level of mismatches of all G4-seq test fragments and concatenated vector of the corresponding predicted maximal scores.

6 Discussion

The objective of the tools for G4 prediction is to model the complex relationship between DNA sequence and G4 structure. Despite our ability to model this relationship directly at the molecular level, using for example molecular dynamics Amber tool (Salomon-Ferrer *et al.*, 2013), this approach is computationally demanding and the accuracy of the state-of-the-art force fields is still limited. For these reasons, existing tools for G4 prediction use much simpler models.

The majority of tools are based on a simple folding rule and are very fast, but do not allow for possible defects (mismatches and bulges) easily. There are tools, such as TetraplexFinder and ImGQfinder that allow for imperfections in G-quadruplexes. However, without a properly trained scoring model this can easily lead to a large number of false positives. These tools performed better in our tests when imperfections were limited or not allowed at all.

A very interesting approach allowing imperfections that is based on specific encoding and simple statistic over a sliding window was implemented in G4Hunter. Despite its simplicity, it shows very good performance characteristics. Unfortunately, we believe that such simple encoding and statistics cannot reveal all complex relationships between sequence and G4 stability, and thus the accuracy of such approach is limited.

On the other hand, the approach proposed in this article that combines pattern matching and detailed inspection of possible defects is configurable and easily extensible. Using advanced options, it can be quickly customized to detect novel and experimental G4 types that are currently not commonly studied or might be discovered in the future. One such example is the recently postulated interstrand G4s (Kudlicki, 2016) or G4s formed in *cis*, as proposed by Hegyi (2015).

By default, the *pqsfinder* provides a scoring function that was trained on G4-seq experimental data and performs better than competing tools. We are aware that G4-seq data essentially represent conditions *in vitro* and may not necessarily be directly related to the ability of G4s to form *in vivo*, but our current view is that *in vivo* G4 formation is a function of their *in vitro* stability. Therefore, G-seq experimental data is the best publicly available dataset we could find at this moment.

However, detailed inspection and modularity are at the cost of lower processing speed. In the extremely sensitive configuration having the minimal G run length set to 2, the algorithm is able to process approximately 4 kb per second on current hardware. For example, *pqsfinder* running time on the G4-seq test set (in total 10 Mbp) was around 40 minutes. When the minimal G run length is increased by one, the speed is usually more than doubled. We do not consider the speed limitations to be critical. For the most frequently studied sequences, *pqsfinder* results can be precomputed and provided to many users, for example as an R data file or a GFF3-formatted file.

7 Conclusion

We created a PQS detection tool with a sequence scoring function that has a moderate number of tunable parameters reflecting sequence properties previously associated with observed G4s or their destabilization (number of Gs in G runs, loop length, presence of mismatches and bulges). To model G-quadruplexes and search for the responsible sequences, we selected a mix of known and novel approaches that give the *pqsfinder* several desirable characteristics. In our tests it achieved the best accuracy on both experimentally verified G-quadruplexes (Lit392) and the independent part of G4-seq data (none of these datasets were used for training). The *pqsfinder* estimates the total number of possible local conformations, accounts for competition between them and allows for imperfections with a sound, carefully trained structure-based scoring model. The presented model was trained on a subset of G4-seq data that represents the largest set of experimentally verified quadruplex-forming sequences available so far and includes a wide variety of imperfections. This new tool also evaluates all the competing conformations and can be easily expanded or modified for newly discovered rules and scoring functions in future. We provide evidence that the *pqsfinder* is a convenient R/Bioconductor package compatible with many other packages available in this environment.

Acknowledgements

Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme ‘Projects of Large Research, Development, and Innovations Infrastructures’ (CESNET LM2015042) is greatly appreciated.

Funding

This work was supported by the Czech Science Foundation [15-02891S to M.L.], IT4Innovations excellence in science project of the Ministry of Education, Youth and Sports of the Czech Republic [IT4I XS – LQ1602 to T.M. and J.Z.] and ICT tools, methods and technologies for smart cities project of the Brno University of Technology [FIT-S-17-3964 to J.H.].

Conflict of Interest: none declared.

References

Agrawal,P. *et al.* (2014) The major G-quadruplex formed in the human BCL-2 proximal promoter adopts a parallel structure with a 13-nt loop in K^+ solution. *J. Am. Chem. Soc.*, **136**, 1750–1753.

Bacolla,A. and Wells,R.D. (2009) Non-B DNA conformations as determinants of mutagenesis and human disease. *Mol. Carcinogenesis*, **48**, 273–285.

Bedrat,A. *et al.* (2016) Re-evaluation of G-quadruplex propensity with G4Hunter. *Nucleic Acids Res.*, **44**, 1746–1759.

Chambers,V.S. *et al.* (2015) High-throughput sequencing of DNA G-quadruplex structures in the human genome. *Nat. Biotechnol.*, **33**, 877–881.

D’Antonio,L. and Bagga,P. (2004) Computational methods for predicting intramolecular G-quadruplexes in nucleotide sequences. In: *Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference*, 2004, *CSB 2004*, Institute of Electrical and Electronics Engineers (IEEE). pp. 590–591.

Dhapola,P. and Chowdhury,S. (2016) QuadBase2: web server for multiplexed guanine quadruplex mining and visualization. *Nucleic Acids Res.*, **44**, W277–W283.

Du,X. *et al.* (2013) The genome-wide distribution of non-B DNA motifs is shaped by operon structure and suggests the transcriptional importance of non-B DNA structures in *Escherichia coli*. *Nucleic Acids Res.*, **41**, 5965–5977.

Eddelbuettel,D. and François,R. (2011) Rcpp: Seamless R and C++ integration. *J. Stat. Softw.*, **40**, 1–18.

Eddelbuettel,D. *et al.* (2016) BH: Boost C++ Header Files. R package version 1.62.0-1.

Guédin,A. *et al.* (2010) How long is too long? Effects of loop size on G-quadruplex stability. *Nucleic Acids Res.*, **38**, 7858–7868.

Hegyi,H. (2015) Enhancer-promoter interaction facilitated by transiently forming G-quadruplexes. *Scientific Rep.*, **5**, 9165.

Hon,J. *et al.* (2013) Triplex: an R/Bioconductor package for identification and visualization of potential intramolecular triplex patterns in DNA sequences. *Bioinformatics*, **29**, 1900–1901.

Huber,W. *et al.* (2015) Orchestrating high-throughput genomic analysis with Bioconductor. *Nat. Methods*, **12**, 115–121.

Huppert,J.L. (2005) Prevalence of quadruplexes in the human genome. *Nucleic Acids Res.*, **33**, 2908–2916.

Kejnovsky,E. and Lexa,M. (2014) Quadruplex-forming DNA sequences spread by retrotransposons may serve as genome regulators. *Mobile Genet. Elements*, **4**, e28084.

Kikin,O. *et al.* (2006) QGRS mapper: a web-based server for predicting G-quadruplexes in nucleotide sequences. *Nucleic Acids Res.*, **34**, W676–W682.

Kudlicki,A.S. (2016) G-quadruplexes involving both strands of genomic DNA are highly abundant and colocalize with functional sites in the human genome. *Plos One*, **11**, e0146174.

Lawrence,M. *et al.* (2013) Software for computing and annotating genomic ranges. *PLoS Comput. Biol.*, **9**, e1003118.

Lexa,M. *et al.* (2011) A dynamic programming algorithm for identification of triplex-forming sequences. *Bioinformatics*, **27**, 2510–2517.

Lexa,M. *et al.* (2014) Guanine quadruplexes are formed by specific regions of human transposable elements. *BMC Genomics*, **15**, 1032.

Maddock,J. (2016). *Boost.Regex 5.1.2.*

Marusic,M. *et al.* (2013) G-rich vegf aptamer with locked and unlocked nucleic acid modifications exhibits a unique g-quadruplex fold. *Nucleic Acids Res.*, **41**, 9524–9536.

Mendoza,O. *et al.* (2016) G-quadruplexes and helicases. *Nucleic Acids Res.*, **44**, 1989–2006.

Mukundan,V.T. and Phan,A.T. (2013) Bulges in G-quadruplexes: Broadening the definition of G-quadruplex-forming sequences. *J. Am. Chem. Soc.*, **135**, 5017–5028.

Pagès,H. *et al.* (2016a) *Biostrings: String objects representing biological sequences, and matching algorithms.* R package version 2.42.1.

Pagès,H. *et al.* (2016b) *S4Vectors: S4 implementation of vectors and lists.* R package version 0.12.1.

Rhodes,D. and Lipps,H.J. (2015) G-quadruplexes and their regulatory roles in biology. *Nucleic Acids Res.*, **43**, 8627–8637.

Salomon-Ferrer,R. *et al.* (2013) An overview of the Amber biomolecular simulation package. *Wiley Interdisc. Rev. Comput. Mol. Sci.*, **3**, 198–210.

SantaLucia,J. Jr., (2012) A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. *Proc. Natl. Acad. Sci. USA*, **95**, 1460–1465.

Scaria,V. *et al.* (2006) Quadfinder: server for identification and analysis of quadruplex-forming motifs in nucleotide sequences. *Nucleic Acids Res.*, **34**, W683–W685.

Scrucca,L. (2013) GA: a package for genetic algorithms in R. *J. Stat. Softw.*, **53**, 1–37.

Varizhuk,A. *et al.* (2014) An improved search algorithm to find G-quadruplexes in genome sequences. *bioRxiv*.

Varizhuk,A. *et al.* (2017) The expanding repertoire of G4 DNA structures. *Biochimie*, **135**, 54–62.

Wells,R.D. (2007) Non-B DNA conformations, mutagenesis and disease. *Trends Biochem. Sci.*, **32**, 271–278.

Zuker,M. (2003) Mfold web server for nucleic acid folding and hybridization prediction. *Nucleic Acids Res.*, **31**, 3406–3415.

E Lexa et al., 2020

Bioinformatics, 36(20), 2020, 4991–4999
doi: 10.1093/bioinformatics/btaa632
Advance Access Publication Date: 14 July 2020
Original Paper

Sequence analysis

TE-greedy-nester: structure-based detection of LTR retrotransposons and their nesting

Matej Lexa [1,2,†,*], Pavel Jedlicka [1,†], Ivan Vanat [2], Michal Cervenansky [1,2] and Eduard Kejnovsky [1]

[1]Department of Plant Developmental Genetics, Institute of Biophysics of the Czech Academy of Sciences, 61200 Brno, Czech Republic and [2] Department of Machine Learning and Data Processing, Faculty of Informatics, Masaryk University, 60200 Brno, Czech Republic

*To whom correspondence should be addressed.

†The authors wish it to be known that, in their opinion, the first two authors should be regarded as Joint First Authors.

Associate Editor: Pier Luigi Martelli

Received on September 11, 2019; revised on June 8, 2020; editorial decision on July 3, 2020; accepted on July 7, 2020

Abstract

Motivation: Transposable elements (TEs) in eukaryotes often get inserted into one another, forming sequences that become a complex mixture of full-length elements and their fragments. The reconstruction of full-length elements and the order in which they have been inserted is important for genome and transposon evolution studies. However, the accumulation of mutations and genome rearrangements over evolutionary time makes this process error-prone and decreases the efficiency of software aiming to recover all nested full-length TEs.

Results: We created software that uses a greedy recursive algorithm to mine increasingly fragmented copies of full-length LTR retrotransposons in assembled genomes and other sequence data. The software called TE-greedy-nester considers not only sequence similarity but also the structure of elements. This new tool was tested on a set of natural and synthetic sequences and its accuracy was compared to similar software. We found TE-greedy-nester to be superior in a number of parameters, namely computation time and full-length TE recovery in highly nested regions.

Availability and implementation: http://gitlab.fi.muni.cz/lexa/nested.

Contact: lexa@fi.muni.cz

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

Genomes of most eukaryotic organisms contain repetitive sequences present as dispersed repeats created by different classes of transposable elements (TEs) (Kapitonov and Jurka, 1999; Smit, 1999). The dispersed repeats are produced throughout evolution by the activity of TEs, often in transposition bursts of various intensities, with many transposition events happening in a short evolutionary time frame, such as those after polyploidization events (Hirochika, 1997; Vicient and Casacuberta, 2017). In genomes with high TE content, some insertions necessarily result in the fragmentation of another transposon already present at that particular insertion locus. This leads to nesting where only the youngest full-length copies can be recognized by software not accounting for fragmentation. Previous estimates of TE nesting in plant genomes ranged from no nesting detected in *Physcomitrella patens* to 14.3% of TEs fragmented by a TE insertion in *Oryza sativa* (Gao *et al.*, 2012). There are many tools and approaches searching for repeated sequences and their families (Bergman and Quesneville, 2007; Saha *et al.*, 2008; Valencia and Girgis, 2019). An exhaustive list has been published in a recent review (Goerner-Potvin and Bourque, 2018). To discover nesting, people have come up with strategies to identify transposon fragments that may have originally been a part of a full-length element. Perhaps the most popular is RepeatMasker in its newer version http://www.repeatmasker.org. It identifies fragments based on sequence similarity to a library of known repeats and stitches together nearby fragments that look like they are continuations of each other when mapped to a model element. LTRtype (Zeng *et al.*, 2017) identifies different types of structurally complex LTR retrotransposon elements as well as the nested configuration of these TEs. The system is capable of rapidly scanning large-scale genomic sequences characterizing eight complex types of LTR retrotransposon elements in addition to the common configuration of two LTR sequences positioned around an internal sequence with protein coding regions. The authors claim the program is able to correctly annotate a large number of structurally complex elements as well as nested insertions. It includes complex elements, e.g. those made of up to two internal sequences and three LTRs. REannotate (Pereira, 2008) processes RepeatMasker annotation for: automated defragmentation of dispersed repetitive elements; resolution of the temporal order of insertions in clusters of nested elements; and the estimation of the age of elements with long terminal repeats. Another specialized software tool is TEnest, for untangling nested insertions of LTR retrotransposons, also using sequence similarity and classification of identified repeats into families (Kronmiller and Wise, 2008, 2013). If nearby fragments belong to the same family, the software will recognize them as a valid power set and assign them to the same full-length element, thus establishing a nesting order. Greedier (Li *et al.*, 2008) is an alignment-based software tool also used to discover nested insertions of transposons. Stitzer *et al.* (2019) mentioned a recursive approach of annotating nesting of TEs, although no software is provided or mentioned in their paper.

All available tools rely heavily on the evaluation of sequence similarity at some key step but it is evident that the structure of elements (order of domains and regulatory regions) can help to reconstruct fragmented elements. Structure-based tools are specifically available for certain classes of repetitive sequences, such as LTR retrotransposons (Ellinghaus, 2008; McCarthy and McDonald, 2003; Xu and Wang, 2007), however, none are capable of recognizing element nesting. Therefore, here, we present an alternative approach to detect nesting, using structure-based recognition of repetitive sequences, relying primarily on identification of component features of a typical transposon and their relative position.

2 Algorithm and implementation

We felt there is a possibility for improvement on TE detection by combining structure-based tools with a greedy algorithmic approach that would eliminate all detected TEs from analyzed sequences before going to the next round of detection. Initial rounds of analysis would help reconstruct many TEs that were fragmented by insertion of younger TEs but underwent little other change. Such an approach is inherently modular, it allows us to use an external, independently tested tool to detect LTR retrotransposons (or any other classes and tools in future modifications) and to separate full-length TE detection from their scoring and establishment of the most likely nesting order.

Besides developing the TE-greedy-nester (labeled as TE-g-nester in all figures and tables), we also used the common code base for creating an application that runs in the opposite direction, to generate sequences containing nested TEs (‘TE-generator’). The TE-generator can be used for the limited testing of TE-greedy-nester. The TE-greedy-nester, TE-generator and other softwares are available from our GitLab project homepage at https://gitlab.fi.muni.cz/lexa/nested. It is written in Python and will run on most Unix/Linux platforms. It requires prior installation of LTR FINDER (Xu and Wang, 2007), BLAST (Altschul *et al.*, 1990) and GenomeTools (Gremme *et al.*, 2013). An installation script and a link to an Ubuntu virtual machine with the latest version installed are provided.

We have also previously packaged our tools for integration and easier deployment. A Snakemake pipeline was created to run TE-greedy-nester on larger datasets and store results of the analysis in GFF files and also in a relational database http://hedron.fi.muni.cz/TEDb/index.html. A Linux Mint virtual machine has been created to enable users not only to work with the above pipeline avoiding potential installation issues but also to modify it to meet their specific requirements http://hedron.fi.muni.cz/TE-nester_Mint.zip.

2.1 TE-greedy-nester

Our main goal was to design an application capable of processing sequences automatically and finding nested TEs in reasonable time. We needed to address specific problems related to the correct detection of element nesting. First, while sensitive enough, the procedure should be resistant to detecting false positives. To this end, we incorporate a greedy algorithm that evaluates several possible candidates for full-length TEs but ultimately picks only the best ones, based on the presence of typical full-length TE sequence features. As a result, false positives are quite rare initially and may become more frequent at later stages which, however, can be stopped at that point. To support precision, we chose to use LTR FINDER (Xu and Wang, 2007), a TE detection tool that showed low false-positive results and high precision in our experience as well as recent tests (Valencia and Girgis, 2019). Another requirement is the ability to detect deep nesting. In such cases, the oldest elements are barely recognizable because of ageing and the procedure must allow for imperfections without compromising the ability to detect the partly eroded elements.

Several rounds of design produced a procedure according to the following pseudocode (see alsoFig. 1A) where capitalized terms in parentheses represent typical components of an LTR retrotransposon, non-coding sequences (LTR, PBS, PPT and TSD) detected by LTR Finder and conserved protein-coding sequences often called protein domains (GAG, PROT, RH, RT, INT and CHD) detected using BLASTX.

```
algorithm TE-greedy-nester is
  input: DNA sequences
  while changed(export_TEs) = TRUE do
    foreach DNA sequence do
      with LTR Finder detect module =
      (LTRs|PBS|PPT|TSD)
        save TE
      with BLASTX get domain =
       (GAG|PROT|RH|RT|INTT|CHD)
    foreach TE do
        hsno_TE = calculate_score(TE, domain,
        module)
      //using greedy algorithm, scoring graph
  save hsno_TE//highest-score_non-overlapping TE
  removed_positions = positions(hsno_TE)
  move removed_positions to export_TEs
  save removed_positions//to adjust export_GFF
export_GFF = adjust(export_GFF, removed_
 positions)
output: export_TEs, export_GFF
```

Evaluation of full-length TE candidates is done by constructing a weighted directed graph, where nodes represent required sites in a full-length element (such as domains, PBS, PPT and TSD) (Fig. 1B). The program is designed to find a path from the left LTR to the right LTR, whilst visiting every required node in the correct order (domains are ordered differently in Gypsy and Copia families, some, like ENV are family-specific, all components are optional). By assigning weights to the edges, we prioritize a path that has as complete a structure as possible. At the same time, we allow alternative paths with respective penalties if there is a missing node, or an incorrect order of available nodes. The graph structure is quite universal and is open for future refinements.

We also need a way to recover various subsequences of the analyzed sequence, such as the original unfragmented sequences of older TEs fragmented by nesting and the identified features annotated to the analyzed sequence. This is achieved by a procedure where the removed sequences are virtually returned to their positions in the genome and the coordinates of TEs and their features are adjusted for the inserted elements. Once all TEs that have been removed in the first phase are processed, we generate a GFF3 file with coordinates that map to the analyzed sequence. The final GFF output file can be used to visualize all the identified features with specialized software, such as Genome Tools Annotation Sketch (Fig. 2) (Gremme *et al.*, 2013), a genome browser, such as IGV (Robinson *et al.*, 2011; Thorvaldsdottir *et al.*, 2013), or to extract sequences for certain features using e.g. bedtools (Quinlan and Hall, 2010). In addition, the sequences of all TEs detected by TE-greedy-nester are clipped out of the genomic matrices and stored in FASTA format in a separate directory.

2.2 TE-generator

To carry out tests of the software, especially its ability to recover nested sequences, we used TE-generator, part of the code that is designed to carry out virtual insertions of TE sequences from a library into a background sequence. The precise position of each inserted sequence in the resulting test sequence is recorded, to compare the generated GFF3 file with results of analysis of the same sequence by TE-greedy-nester.

3 System and methods

3.1 Testing data

3.1.1 Randomly inserted sequences

A first group of testing sequences was prepared as a random mixture of full-length sequences from maize. A databases of full-length TEs were downloaded from Maize TE Database at http://people.oregonstate.edu/~fowlerjo/MaizeRepeatDatabases/uniqueTEDB_1526.fasta.txt in 2018 and Mips-REdat database at ftp://ftpmips.helmholtz-muenchen.de/plants/REdat/mipsREdat_9.3p_ALL.fasta.gz (Nussbaumer *et al.*, 2013) for maize and rice full-length TEs, respectively. The generator module of TE-greedy-nester was used to generate 10 Mbp sequences with 10 and 90% TE sequence content. Settings were calculated from average TE length (see commandsUsed.txt in Supplementary Materials).

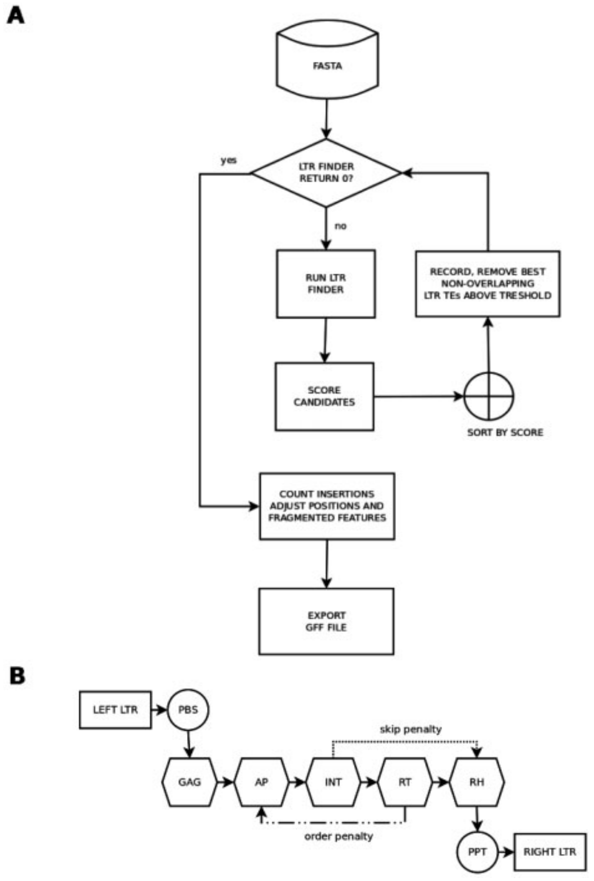


Fig. 1. Nesting algorithm essentials. (**A**) Algorithm overview showing data processing in TE-greedy-nester; (**B**) Scoring graph structure used to evaluate structural completeness of LTR retrotransposon candidates (shown as SCORE CANDIDATES in panel A). Any deviation from prescribed order of structural components (full arrows) is penalized (dotted arrows)

3.1.2 Deeply nested sequences (Russian doll/Matryoshka)

To test the depth of nesting discoverable by existing tools, as well as TE-greedy-nester, we wrote a short program (see file matryoshka.pl) which creates annotated sequence files with an arbitrary depth of mutually nested TE elements. It complements TE-generator, which also creates pockets of multiply nested TEs in the generated sequence, however, TE-generator is biased towards shallow nested sets. Matryoshka is written in Perl (code available in Supplementary Materials). The program takes a multifasta file of LTR retrotransposon sequences as input, samples it *i* times (*i* can be set on the command line), nesting each consecutive sequence into a random position of the previously inserted sequence. FASTA and BED files are written as output. Two sets of sequences containing 20 TEs taken from either *Zea mays* TE database (zea_matryoshka.fa) or *O.sativa* (oryza_matryoshka.fa) were generated.

3.1.3 Maize *adh1* neighborhood sequence

To test TE-greedy-nester on biological sequence which is rich in nested LTR retrotransposons with known position and annotation, the *Z.mays* alcohol dehydrogenase 1 gene (SanMiguel *et al.*, 1998) (*adh1-F* gene accession number: AF050457.1) flanked with 150 kbp on both 5′ and 3′ regions, was analyzed. The maize genome was downloaded from Phytozome 12.0 https://phytozome.jgi.doe.gov/pz/portal.html (Goodstein *et al.*, 2012). TE sequences obtained by TE-greedy-nester were annotated using the maize specific retrotransposon database multifasta http://people.oregonstate.edu/~fowlerjo/MaizeRepeatDatabases/uniqueTEDB_1526.fasta.txt and BLASTN tool (Altschul *et al.*, 1990). Thereafter, the GT Annotation Sketch figure from TE-greedy-nester was rearranged following TE orientation given in the study by SanMiguel *et al.* (1998) as later adapted by Fedoroff (2012).

3.1.4 Maize sequence (first 2 MB from Chr10)

The second biological sequence from the maize genome, Chromosome10: 0–2 Mb, was used previously to compare LTRtype, REannotate and TEnest tools (Zeng *et al.*, 2017). We used this sequence to maintain continuity and to obtain reliable data for users of TE-greedy-nester.

3.2 Testing software and data analysis

Performance of TE-greedy-nester and other software able to detect TE nesting (TEnest, LTRtype and REannotate) were tested on three different types of data: (i) synthetic data with known randomly inserted sequences from maize and rice (TE-generator and matryoshka), (ii) thoroughly studied and annotated *adh1* locus from maize and (iii) biological data analyzed by other similar software (2 MB from maize Chr10).

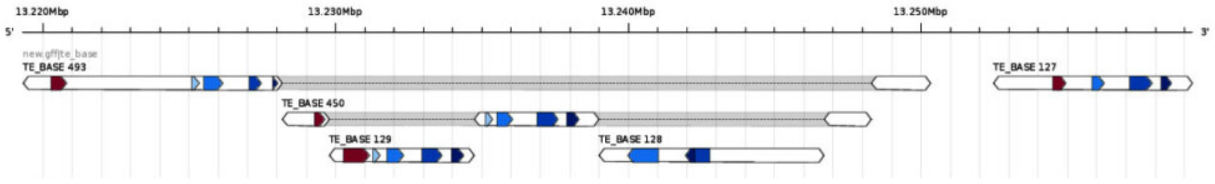


Fig. 2. An example of TE-greedy-nester output visualized using Annotation Sketch from the Genome Tools (Gremme *et al.*, 2013) software suite (Command: `gt sketch output.png example.gff`)

Testing on synthetic data was done by comparing GFF files produced by TE-greedy-nester with TE-generator and matryoshka GFF files describing the introduced nesting. We calculated the number of intervals representing TEs that overlap by a predefined percentage of length on both sides using the python script compGffs2Generator.py (Supplementary Materials).

Testing on the *adh1* locus was done by visual inspection of published *adh1* annotations (Fedoroff, 2012) and our own visualization with GT Annotation Sketch.

Testing on biological data was carried out by counting the number of detected full-length TEs in assembled or partly assembled genomes of 18 species downloaded from Phytozome or other sources (where applicable): *Arabidopsis thaliana*, *Arabidopsis lyrata*, *Azolla filiculoides*, *Brachypodium distachion*, *Chlamydomonas reinhardtii*, *Dunaliella salina*, *Glycine max*, *Gossypium raimondii*, *Lotus japonicus*, *Medicago truncatula*, *Micromonas pusilla*, *Musa acuminata*, *O.sativa*, *P.patens*, *Populus trichocarpa*, *Pseudotsuga menziesii*, *Solanum lycopersicum* and *Sorghum bicolor*.

To confirm a quality of TEs retrieved by TE-greedy-nester, their long terminal repeat (LTR) sequences were cut with bedtools package (Quinlan and Hall, 2010) and subsequently LTR identity was measured using global alignment by STRETCHER tool (Emboss 6.6.0; Rice *et al.*, 2000).

3.3 Performance measures

Performance measures used here were calculated according to the formulas used in the study by Ou and Jiang (2018)

$$\text{Sensitivity} = \text{TP}/(\text{TP}+\text{FN})$$
$$\text{Specificity} = \text{TN}/(\text{FP}+\text{TN})$$
$$\text{Accuracy} = (\text{TP}+\text{TN})/(\text{TP}+\text{TN}+\text{FP}+\text{FN})$$
$$\text{Precision} = \text{TP}/(\text{TP}+\text{FP})$$

where true positive (TP) stands for matches of coordinates of TE found by tested tools and corresponding element given by TE-generator within tolerance ±5% of reference TE length at the start and end positions. Correspondingly, false positive (FP) is a TE with no match with any TEs in the generated sequences, a false negative (FN) are TEs which were present in generated sequence and absent in output GFFs from given tools, true negative (TN) is estimated TE count from number of bases which are without TE in both GFF from TE-generator and also from tested tools.

3.4 Speed and RAM performance

Resource utilization of the four different software tools was compared on all the datasets used in this article. For evaluations, the programs were run on a dedicated Linux machine running no other job, using the GNU time command to obtain ‘real time’ and ‘maximum resident set size’ as an average of three runs. The machine had 12 GB physical RAM, a 4 core Intel i5 CPU. The running process was monitored for signs of memory swapping and process pruning.

4 Results

We developed the TE-greedy-nester software for finding nested LTR retrotransposons using a combination of a greedy algorithm and identification of full-length TEs. In contrast to comparable software relying mostly on sequence similarity, TE-greedy-nester is based on identification of structural features of LTR retrotransposons. We compared the performance of TE-greedy-nester with four other related software tools. The number of features detected by TE-greedy-nester on different types of data is reported in Table 1. We found that TE-greedy-nester detected the highest number of TEs in all tested sequences in comparison with other examined tools. Moreover, despite the higher rate of false-positive identification, TE-greedy-nester also retrieved the highest count of TEs matched with elements present in annotated sequences (number of TEs matched, Table 1). To better evaluate the performance of TE-greedy-nester, we carried out a deeper analysis using both synthetic and biological data.

4.1 Annotated synthetic data (TE-generator and matryoshka)

To test TE-greedy-nester on synthetic data, we generated artificial sequences using TE-generator. The TE-generator sequences had medium levels of nesting. We also prepared sequences with extreme depth of nesting, only inserting new sequences into previously inserted ones and call them ‘matryoshka’. After running TE-greedy-nester on these sequences, we evaluated sensitivity, selectivity, precision and accuracy (see Section 3). Sensitivity and precision measure the ability to detect sequences, selectivity measures the ability to reject false positives and accuracy is a combination of both. Calculated comparative performance values are shown in Figure 3. While TE-greedy-nester showed higher sensitivity for all available data, its comparative accuracy gave mixed results, with lower specificity (higher false-positive rate) on synthetic data with high TE density (90%) (Fig. 3B). On the other hand, TE-greedy-nester was superior to all existing software on synthetic data with deep nesting (matryoshka). While TEnest could compete with TE-greedy-nester when provided with the proper TE database (maize) (Fig. 3C), TE-greedy-nester was the only software that could detect deep nesting correctly on mixed-origin TE data (Fig. 3D), showing an even higher accuracy on mixed data than maize data. TE-greedy-nester was also one to two orders of magnitude faster than TEnest on all datasets.

Because the above performance evaluations on annotated data partly depend on the definition of successful TE identification, we examined the effect of position tolerance on performance measures in the same four annotated synthetic datasets (Fig. 4). It can be seen that low copy datasets, such as matryoshka, produce the same number of TEs at the lowest used tolerance of 1% (Fig. 4C and D). TE rich data from TE-generator show an increased TE discovery at higher tolerance level, most likely as a result of increased false-positive rate, this was more apparent in sequences with 90% TE coverage (Fig. 4B) than in those with 10% TE coverage (Fig. 4A).

Table 1. Number of detected or expected full-length LTR retrotransposons by TE-greedy-nester, TEnest, LTRtype and REannotate on natural (rows 1–2) and artificial (rows 3–6) testing sequences

Sequence name	Seq. length (bp)	No. of reference TEs	No. of TEs found				No. of TEs matched (tolerance = 0.01)			
			TE-g-nester	TEnest	LTRtype	REannotate	TE-g-nester	TEnest	LTRtype	REannotate
Zm_adh1	302987	21	15	11	7	6	—	—	—	—
Zm_Chr10_2Mb	2000001	—	157	46	60	21	—	—	—	—
Zm_synth_10	10141285	260	78	77	42	30	35	36	24	14
Zm_synth_90	10271500	2329	774	177	250	193	72	15	36	16
Zm_matryoshka_20	120219	20	16	14	2	2	13	11	0	0
Os_synth_10	10190963	100	90	16	6	3	78	6	6	2
Os_synth_90	9562586	900	729	71	44	24	291	1	19	8
Os_matryoshka_20	198560	20	14	4	0	0	1	4	0	0

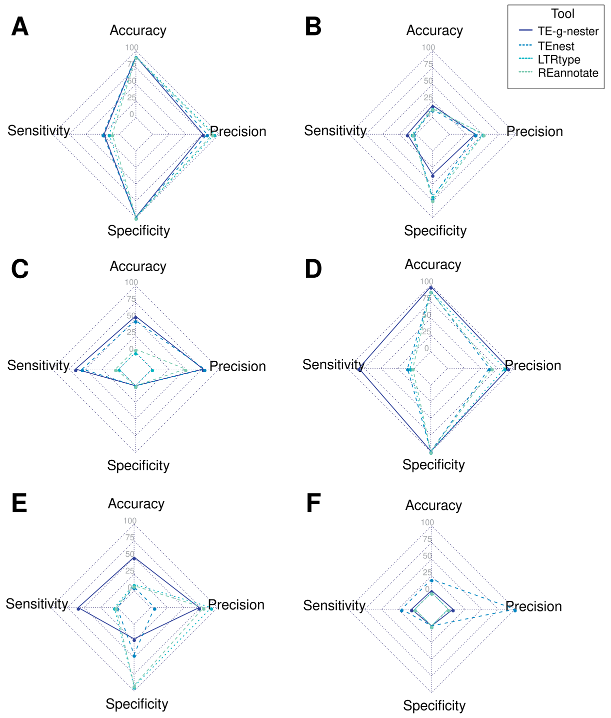


Fig. 3. Sensitivity, specificity, accuracy and precision of TE-greedy-nester and comparable software on synthetic and biological data; (**A**) zea_10%; (**B**) zea_90%; (**C**) zea_matryoshka; (**D**) oryza_10%, (**E**) oryza_90% and (**F**) oryza_matryoshka

While Figure 4 shows the overall performance values of the different tools tested, their abilities to discover nested TEs remain partly hidden in the lump sum numbers. We therefore divided the counts of identified TEs based on their nesting status, including their nesting level (how many successive nesting events occurred within the given TE internal region) (Fig. 5). The most striking finding is that all tools overestimate the number of non-nested elements and underestimate the number of nested ones (Fig. 5A and B). TEnest was less prone to this error in high-density TE data (90% TEs; Fig. 5B), in accordance with its higher specificity on the same data (Fig. 3B). LTRtype and REannotate missed more TEs than the other two tools. Only TEnest and TE-greedy-nester were able to resolve the majority of the deeply nested matryoshka dataset (Fig. 5C and D). The same tendency could be seen at different nesting levels in 90% TE-generator data. TEnest and TE-greedy-nester always had much higher counts than the other tools at nesting levels II and deeper. TEnest gave higher counts than TE-greedy-nester at nesting levels IV and higher, however, most of those above nesting level VI were false positives (Fig. 5B). The best performance of TE-greedy-nester on mixed origin matryoshka data (generated with TE sequences from multiple plant species) observed in Figure 5D can be seen here as well.

For a better perspective of individual tool performance, we also show the data from this analysis in the Integrated Genome Browser (Robinson *et al.*, 2011) (Supplementary Fig. S1). While both TE-greedy-nester and to a limited extent also TEnest had a tendency to overestimate certain types of TEs (false positives), in the overall visualization, TE-greedy-nester results render best the overall density and nesting depth distribution of TEs along the sequence.

4.2 Biological data

After testing on synthetic data, we applied the compared tools to biological data, namely (i) the well-studied *adh1* region from maize and (ii) a 2-MB region from maize chromosome 10 used in a previous comparative study by Zeng *et al.* (2017).

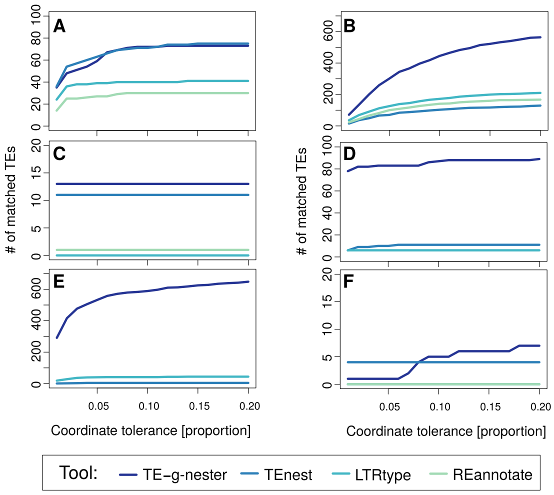


Fig. 4. Number of correctly identified TEs as a function of length tolerance by the four software tools; (**A**) zea_10%; (**B**) zea_90%; (**C**) zea_matryoshka; (**D**) oryza_10%, (**E**) oryza_90% and (**F**) oryza_matryoshka

4.2.1 *Adh1* region

TE-greedy-nester as well as the other three compared tools was tested on the *adh1* region in which 21 full-length LTR retrotransposons (of which 12 are nested) were found by SanMiguel *et al.* (1998). TE-greedy-nester detected 15 (7 nested), TEnest detected 11 (1 nested), LTRtype 7 (0 nested) and REannotate 6 (1 nested) (Fig. 6). Only four of these full-length LTR retrotransposons were identified by all tools, while six were common for TEnest and TE-greedy-nester results, as can be seen in Supplementary Figure S2A.

To compare *adh1* outputs from TE-greedy-nester with the original SanMiguel report on family level, TEs from TE-greedy-nester were additionally annotated using maize-specific TE database http://people.oregonstate.edu/~fowlerjo/MaizeRepeatDatabases/uniqueTEDB_1526.fasta.txt and a locally installed BLASTN (Altschul *et al.*, 1990) tool (Supplementary Fig. S3). Although the TE annotations from TE-greedy-nester and Fedoroff (2012) do not fully match, we counted 12 families that were correctly recognized and placed within the segment.

4.2.2 2 MB region of maize Chr10. We used all compared software tools for an analysis of LTR retrotransposons in the initial 2 MB region of maize chromosomes 10. The maize genome was chosen because of its high content of LTR retrotransposons that are often nested and that it was previously tested by other authors on the same sequence. The results for this maize segment (Supplementary Fig. S2B) were in line with results on 90% TE-generator data. TE-greedy-nester found the most TEs, TEnest was the most conservative in the number of non-nested TEs and LTRtype and REannotate were not able to find full-length TEs beyond nesting level I.

4.3 Performance tests

To compare the four evaluated software tools also by computational performance and requirements, we recorded computation times and peak physical memory usage on the data described above (Table 2). TEnest, which performed very well in nesting accuracy tests above, was the slowest and most memory hungry of the four (worst case 8719 s, 12GB RAM). With the largest datasets, it caused the system to swap memory and kill processes (shown as >12GB RAM in 2) resulting also in a steep increase of computation time. Our TE-greedy-nester was comparable with LTR Type and REannotate in terms of speed and memory usage (worst case 886 s, 628 MB). While it performed better with RAM on extremely small datasets. LTR Type used slightly more RAM (752 MB versus 500–600 MB) and was also slower to compute the results on small datasets (36s versus 13–21 s). It should be noted that TEnest was set to use four

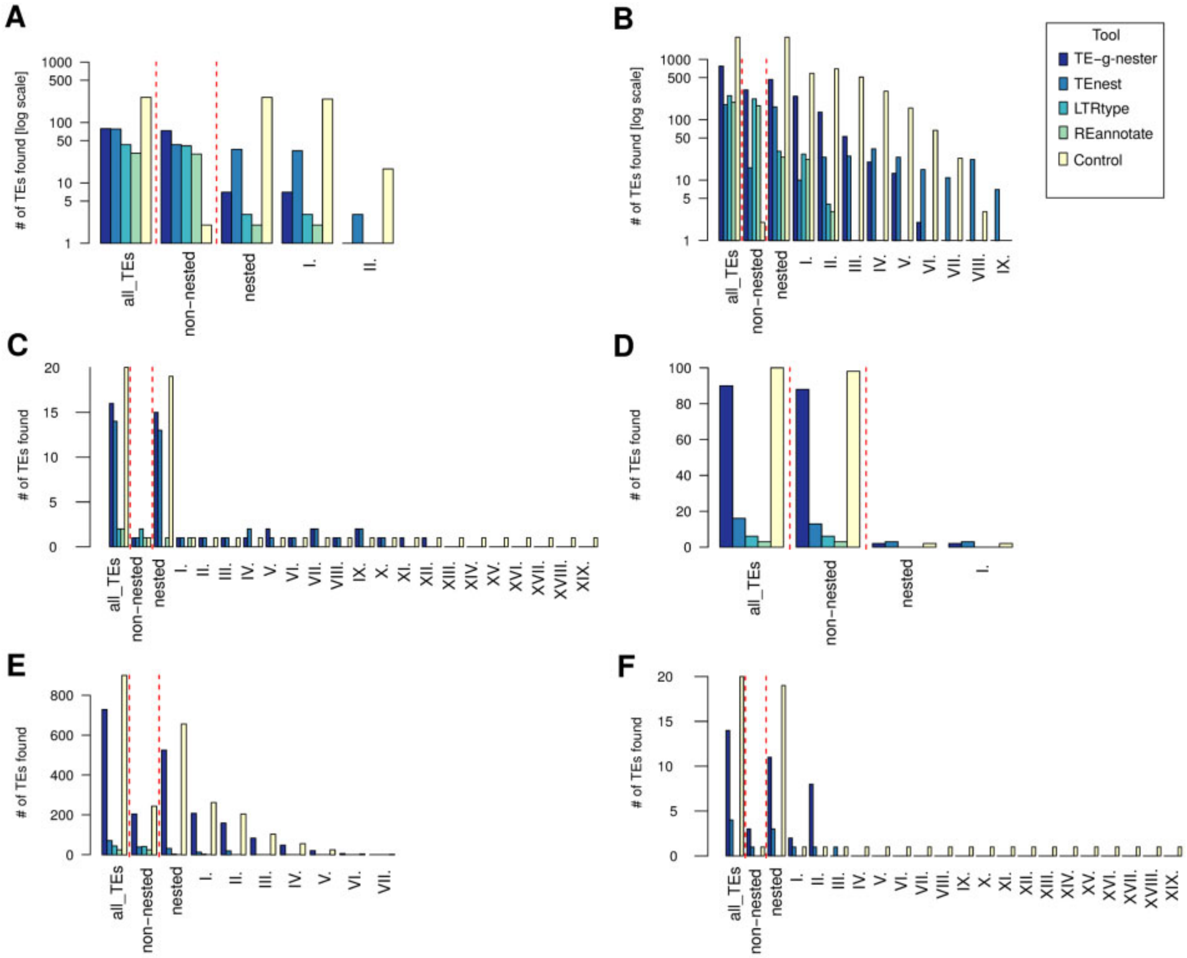


Fig. 5. Number of correctly identified TEs at different nesting levels by the four software tools; (A) zea_10%; (B) zea_90%; (C) zea_matryoshka; (D) oryza_10%, (E) oryza_90% and (F) oryza_matryoshka

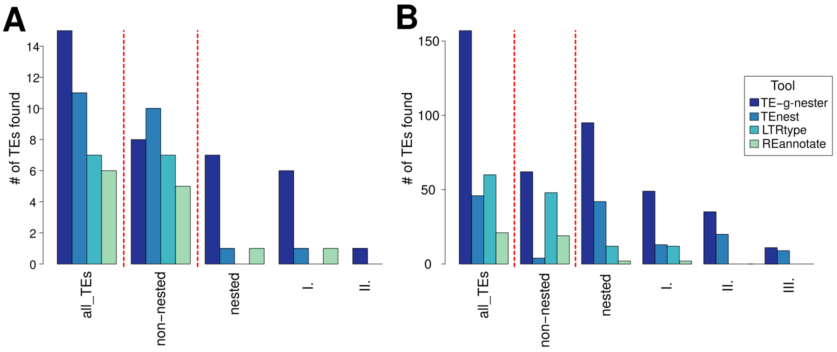


Fig. 6. Number of correctly identified TEs at different nesting levels by the four software tools in biological data; (A) zea_adh1 and (B) zea_2MB

processors, while TE-greedy-nester used multithread BLASTX searches (3 processors). LTR Type and REannotate relied on RepeatMasker output, however, the time to generate that output was included and tabulated as well. For further details, please, see Section 3.

4.4 Plant genomes

Finally, we applied TE-greedy-nester to 18 plant genomes available mostly from Phytozome (see Section 3) and provide the results in GFF files (http://hedron.fi.muni.cz/TEgnester/). In Table 3, we show the classification and the main characteristics of TE nesting in these 18 species. We demonstrated that 96.4% of detected TEs (i.e. 98 187 out of 10 1887), have LTR identity 80% or higher (Supplementary Fig. S4). TE-greedy-nester also found at least one protein domain in 29.8–88.4% of identified LTR retrotransposons in vascular plants but only 0–1.9% in non-vascular plants. The percentage of TEs found in nested configuration was between 19.6 and 54% in vascular plants and 0 and 17.6% in non-vascular plants. The highest nesting rates were observed in *S.bicolor* and *G.max*. The proportion of solo LTRs was higher in eudicots and algae (e.g. *Solanum*, *Gossypium* and *Dunaliella*) than in monocots.

Table 2. Time and memory requirements of tested software when analyzing input data of different types

Species	Sequence	TE-g-nester	TEnest	LTRtype	REannotate
Process time (s)					
Z.mays	synt10	174	83	297	338
	synt90	650	8917	740	657
	matryoshka	16	544	26	11
	adh1	14	1052	36	21
	Chr10_2Mbp	331	7866	136	122
O.sativa	synt10	187	6562	322	330
	synt90	886	3072	145	525
	matryoshka	13	57	36	15
Maximum memory usage (Mbytes)					
Z.mays	synt10	591.5	146.4	740.5	489.0
	synt90	628.1	>12000	740.7	536.3
	matryoshka	74.8	18.2	739.5	482.4
	adh1	77.1	351.5	741.2	482.2
	Chr10_2Mbp	152.7	11800.0	752.2	482.1
O.sativa	synt10	595.7	>12000	740.4	488.0
	synt90	603.7	>12000	740.4	517.9
	matryoshka	80.2	18.6	740.7	482.2

Note: Time and memory usage were measured as 'real time' and 'maximum resident set size' using the Linux time command on a dedicated machine with 12 GB RAM, Intel i5 processor with four cores, Fedora Linux installed and no other jobs running.

5 Discussion

Here, we present a new bioinformatic tool TE-greedy-nester for the detection of nested LTR retrotransposons that is faster and more efficient in finding deeply nested elements than existing tools. These advantages are due to the combination of structure-based retrotransposon detection with recursive sequence removal. This results in comparatively low memory usage and high computation speed, as seen in performance tests presented herein. With the default settings our tool is competitively sensitive although can produce higher rates of false positives in some instances, as seen in both synthetic and real biological data. We are exploring ways to reduce the number of false-positive calls where several directions of action are possible, such as (i) optimizing the search parametrization and scoring of candidate TEs; (ii) improving TE annotation, especially by accounting for TSD sequences that should be present in bonafide full-length TEs and (iii) abandoning the greedy approach by introducing extra passes that would pre-score elements or explore multiple sequence fragmentation scenarios.

Another advantage of TE-greedy-nester is its ability to identify nesting in different species without the need for species specific TE databases. This is in sharp contrast to the other three compared tools that lack performance in cross-species applications. Their results also differ significantly with the size and sparsity of the used TE database.

While the focus now is on the nesting of LTR retrotransposons, the approach is modular and can be expanded to other classes of repetitive sequences by simply employing additional TE detection tools alongside LTR Finder. However, even without expansion to other TE classes, the algorithm in TE-greedy-nester can detect short foreign insertions in full-length LTR retrotransposons. This is the advantage of using structural information where the most important signal is the order of required TE components, while distance and sequence similarity is secondary.

Both synthetic and biological data were chosen to represent different TE densities and levels of nesting to identify the strengths and weaknesses of the tested tools. In this respect, we found that TE-greedy-nester had the highest sensitivity across the board of different tests. As expected, high sensitivity typically comes with a higher proportion of false positives, and so it is important to look at accuracy and precision for an objective comparison of different tools. TE-greedy-nester showed markedly lower precision with 90% TE-generator data, suggesting that it could benefit from parameter fine-tuning depending on the TE density of the data being analyzed. Identifying the best parameter combinations for different types of data is beyond the scope of this article, where only fixed or default settings were used. TE-greedy-nester also found nested full-length TEs one to two orders of magnitude faster than TEnest, which gave the best results of the three compared tools across different tests. LTRtype was more conservative but still performed very well on TE-generator data of both densities. Compared to TEnest, it however failed to be competitive, together with REannotate, on simple data with deep nesting, such as the *adh1* locus or matryoshka data. It should be noted that LTRtype is a tool able to recognize composite LTR retroelements, something the other tools cannot do.

Interestingly, running TE-greedy-nester on several species uncovered remarkable differences to previously published nesting estimates in certain species. For example, in *P.patens* no nesting was observed by Gao *et al.* (2012), while we saw 32% nested LTR retrotransposons.

TE-greedy-nester development is a live ongoing project. While we were testing the software performance on nested full-length TEs, a new feature was added to the code base. TE-greedy-nester can now identify solo-LTRs in the analyzed sequence based on the sequences of the LTRs identified in all iterations.

TE nesting reconstruction is important in genome evolution and TE life cycle studies. We hope it will help users excavate older full-length TE copies, differentiate between complex TEs transcribed as a single unit and nested TEs originating from many independent insertions. Based on the testing results, it may be useful in sequences with deeply nested structures, such as those in centromeric and pericentromeric regions of plant chromosomes. For such use, it might be useful to expand its abilities towards tandem repeat detection. In our observations, tandem repeats are one of the things that can confuse LTR Finder into interpreting some of their subsequences as pairs of LTRs. The situation becomes even more complicated when such tandem repeats originate from fragments of TE sequences containing fragments of canonical domain coding sequences (Ahmed and Liang, 2012) or LTRs. TE-greedy-nester should also come handy in whole genome annotations where speed could be as important as precision.

Our tool is similar to software mentioned by Stitzer *et al.* (2019) in two aspects. We also create a graph data structure to find the best TE candidates, the two structures, however, carry different types of data and are used for slightly different purposes.

6 Conclusion

In this work, we present a new software tool for the recovery of full-length LTR retrotransposons fragmented by the nesting of other elements. We used a recursive approach in combination with structure-based detection of TEs with LTR Finder and implemented it in a Python tool called TE-greedy-nester. We tested this computational approach on synthetic and natural DNA sequences. Testing showed that TE-greedy-nester gave high-quality results faster than existing tools and is superior in selected parameters, especially in its ability to recover full-length LTR retrotransposons in deeply nested regions. Moreover, we analyzed 18 plant genomes and showed that TE-greedy-nester could be used in studies of TE life cycle and genome evolution, especially in areas where relative insertion times are important.

Acknowledgements

The authors thank Christopher Johnson for critical reading of the manuscript.

Table 3. TE-greedy-nester result summary on genome sequences from organisms across the plant kingdom ordered by the detected rate of nesting

Species	Class	Family	Genome size (Mbp)	All TEs			TEs with domain[a]			Annotation rate (%)	Nesting rate (%)	Solo LTRs	Reported TEs
				Nested	Isolated	Sum	Nested	Isolated	Sum				
Selaginella moellendorffii	Isoetopsida	Selaginellaceae	212.7	6704	3187	9891	2489	1756	4245	42.9	58.6	3247	~941[b]
S.bicolor	Monocots	Poaceae	732.2	19662	10128	29790	9947	8479	18426	61.9	54	8706	23915[c]
G.max	Eudicots	Fabaceae	978.5	19325	11203	30528	7128	8610	15738	51.6	45.3	20266	32370[d]
M.acuminata	Monocots	Musaceae	390.6	3304	3599	6903	1270	1842	3112	45.1	40.8	4917	~14400[e]
P.trichocarpa	Eudicots	Salicaceae	422.9	4455	3467	7922	1039	2106	3145	39.7	33	2747	1479[f]
P.patens	Bryopsida	Funariaceae	473.2	3353	6017	9370	2685	5598	8283	88.4	32.4	24643	~24261[g]
Brachypodium	Monocots	Poaceae	271.2	1668	2803	4471	848	1847	2695	60.3	31.5	5174	569[h]
A.thaliana	Eudicots	Brassicaceae	119.1	677	831	1508	196	429	625	41.4	31.4	422	376[i]
S.lycopersicum	Eudicots	Solanaceae	823.9	6895	9494	16389	2770	7234	10004	61	27.7	22591	2086[j]
L.japonicus	Eudicots	Fabaceae	462.5	1923	2706	4629	490	1534	2024	43.7	24.2	1974	~8930[k]
A.lyrata	Eudicots	Brassicaceae	206.7	1746	2820	4566	547	1713	2260	49.5	24.2	1318	~940[l]
O.sativa	Monocots	Poaceae	374.5	3361	5163	8524	929	3206	4135	48.5	22.5	6668	7043[c]
M.truncatula	Eudicots	Fabaceae	411.8	2685	4578	7263	462	1703	2165	29.8	21.3	4569	12250[m]
G.raimondii	Eudicots	Malvaceae	761.4	5064	9835	14899	1611	6620	8231	55.2	19.6	18393	13297[n]
D.salina	Chlorophyceae	Dunaliellaceae	343.7	20002	4314	24316	81	378	459	1.9	17.6	5586	~12572[o]
C.reinhardtii	Chlorophyceae	Chlamy-domonadaceae	107.1	2360	1356	3716	0	47	47	1.3	0	181	~2230[o]
A.filiculoides	Polypodiopsida	Salviniaceae	750	238	270	508	0	0	0	0	0	459	17484[p]
M.pusilla	Mamiello-phyceae	Mamiellaceae	22	20	41	61	0	0	0	0	0	116	~65[o]

Note: ~number of TEs was estimated from TE genome coverage data assuming 10kbp per TE.

[a] Counted from TEs with annotated protein domain.

[b] Vanburen *et al.* (2018).

[c] Jiang and Ramachandran (2013).

[d] Du *et al.* (2010).

[e] Hribova *et al.* (2010).

[f] Cossu *et al.* (2012).

[g] Lang *et al.* (2018).

[h] Stritt *et al.* (2019).

[i] Peterson-Burch *et al.* (2004).

[j] Xu and Du (2014).

[k] Holligan *et al.* (2006).

[l] Civan *et al.* (2011).

[m] Wang and Liu (2008).

[n] Xu *et al.* (2017).

[o] Goodstein *et al.* (2012).

[p] Li *et al.* (2018).

Funding

This research was supported by the Czech Science Foundation [grant number 18-00258S to E.K. and M.L.].

Conflict of Interest: none declared.

References

Ahmed,M. and Liang,P. (2012) Transposable elements are a significant contributor to tandem repeats in the human genome. *Comp. Funct. Genomics*, **199**, 1–7.

Altschul,S.F. *et al.* (1990) Basic local alignment search tool. *J. Mol. Biol.*, **215**, 403–410.

Bergman,C.M. and Quesneville,H. (2007) Discovering and detecting transposable elements in genome sequences. *Brief. Bioinform.*, **8**, 382–392.

Civan,P. *et al.* (2011) On the coevolution of transposable elements and plant genomes. *J. Bot.*, **2011**, 893546.

Cossu,R.M. *et al.* (2012) A computational study of the dynamics of LTR retrotransposons in the *Populus trichocarpa* genome. *Tree Genet. Genomes*, **8**, 61–75.

Du,J. *et al.* (2010) SoyTEdb: a comprehensive database of transposable elements in the soybean genome. *BMC Genomics*, **11**, 113.

Ellinghaus,D. *et al.* (2008) LTRharvest, an efficient and flexible software for de novo detection of LTR retrotransposons. *BMC Bioinformatics*, **9**, 18.

Fedoroff,N.V. (2012) Presidential address. Transposable elements, epigenetics, and genome evolution. *Science*, **338**, 758–767.

Gao,C. *et al.* (2012) Characterization and functional annotation of nested transposable elements in eukaryotic genomes. *Genomics*, **100**, 222–230.

Goerner-Potvin,P. and Bourque,G. (2018) Computational tools to unmask transposable elements. *Nat. Rev. Genet.*, **19**, 688–704.

Goodstein,D.M. *et al.* (2012) Phytozome: a comparative platform for green plant genomics. *Nucleic Acids Res.*, **40**, D1178–1186.

Gremme,G. *et al.* (2013) GenomeTools: a comprehensive software library for efficient processing of structured genome annotations. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, **10**, 645–656.

Hirochika,H. (1997) Retrotransposons of rice: their regulation and use for genome analysis. *Plant Mol. Biol.*, **35**, 231–240.

Holligan,D. *et al.* (2006) The transposable element landscape of the model legume *Lotus japonicus*. *Genetics*, **174**, 2215–2228.

Hribova,E. *et al.* (2010) Repetitive part of the banana (*Musa acuminata*) genome investigated by low-depth 454 sequencing. *BMC Plant Biol.*, **10**, 204.

Jiang,S.-Y. and Ramachandran,S. (2013) Genome-wide survey and comparative analysis of LTR retrotransposons and their captured genes in rice and sorghum. *PLoS One*, **8**, e71118.

Kapitonov,V.V. and Jurka,J. (1999) Molecular paleontology of transposable elements from *Arabidopsis thaliana*. *Genetica*, **107**, 27–37.

Kronmiller,B.A. and Wise,R.P. (2008) TEnest: automated chronological annotation and visualization of nested plant transposable elements. *Plant Physiol.*, **146**, 45–59.

Kronmiller,B.A. and Wise,R.P. (2013) TEnest 2.0: computational annotation and visualization of nested transposable elements. *Methods Mol. Biol.*, **1057**, 305–319.

Lang,D. *et al.* (2018) The *Physcomitrella patens* chromosome-scale assembly reveals moss genome structure and evolution. *Plant J.*, **93**, 515–533.

Li,X. *et al.* (2008) A novel genome-scale repeat finder geared towards transposons. *Bioinformatics*, **24**, 468–476.

Li,F.W. *et al.* (2018) Fern genomes elucidate land plant evolution and cyanobacterial symbioses. *Nat. Plants*, **4**, 460–472.

McCarthy,E.M. and McDonald,J.F. (2003) LTR_STRUC: a novel search and identification program for LTR retrotransposons. *Bioinformatics*, **19**, 362–367.

Nussbaumer,T. *et al.* (2013) MIPS PlantsDB: a database framework for comparative plant genome research. *Nucleic Acids Res.*, **41**, D1144–D1151.

Ou,S. and Jiang,N. (2018) LTR_retriever: a highly accurate and sensitive program for identification of long terminal repeat retrotransposons. *Plant Physiol.*, **176**, 1410–1422.

Pereira,V. (2008) Automated paleontology of repetitive DNA with REannotate. *BMC Genomics*, **9**, 614.

Peterson-Burch,B.D. *et al.* (2004) Genomic neighborhoods for Arabidopsis retrotransposons: a role for targeted integration in the distribution of the Metaviridae. *Genome Biol.*, **5**, R78.

Quinlan,A.R. and Hall,I.M. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, **26**, 841–842.

Rice,P. *et al.* (2000) EMBOSS: the European Molecular Biology Open Software Suite. *Trends Genet.*, **16**, 276–277.

Robinson,J.T. *et al.* (2011) Integrative genomics viewer. *Nat. Biotechnol.*, **29**, 24–26.

Saha,S. *et al.* (2008) Computational approaches and tools used in identification of dispersed repetitive DNA sequences. *Trop. Plant Biol.*, **1**, 85–96.

SanMiguel,P. *et al.* (1998) The paleontology of intergene retrotransposons of maize. *Nat. Genet.*, **20**, 43–45.

Smit,A.F. (1999) Interspersed repeats and other mementos of transposable elements in mammalian genome. *Curr. Opin. Genet. Dev.*, **9**, 657–663.

Stitzer,M.C. *et al.* (2019) The genomic ecosystem of transposable elements in maize. **559922**, 1–48.doi: 10.1101/559922.

Stritt,C. *et al.* (2019) Diversity, dynamics and effects of long terminal repeat retrotransposons in the model grass *Brachypodium distachyon*. *N. Phytol*, **10**.1111/nph.**16308**, 1–13.

Thorvaldsdottir,H. *et al.* (2013) Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. *Brief. Bioinform.*, **14**, 178–192.

Valencia,J.D. and Girgis,H.Z. (2019) LtrDetector: a modern tool-suite for detecting long terminal repeat retrotransposons de-novo on the genomic scale. *BMC Genomics*, **20**, 450.

Vanburen,R. *et al.* (2018) Extreme haplotype variation in the desiccation-tolerant clubmoss Selaginella lepidophylla. *Nat. Commun.*, **9**, 8.

Vicient,C.M. and Casacuberta,J.M. (2017) Impact of transposable elements on polyploid plant genomes. *Ann. Bot. Lond.*, **120**, 195–207.

Wang,H. and Liu,J.S. (2008) LTR retrotransposon landscape in *Medicago truncatula*: more rapid removal than in rice. *BMC Genomics*, **9**, 382–382.

Xu,Y. and Du,J. (2014) Young but not relatively old retrotransposons are preferentially located in gene-rich euchromatic regions in tomato (*Solanum lycopersicum*) plants. *Plant J.*, **80**, 582–591.

Xu,Z. *et al.* (2017) GrTEdb: the first web-based database of transposable elements in cotton (*Gossypium raimondii*). *Database*, **2017**, 1–7.

Xu,Z. and Wang,H. (2007) LTR_FINDER: an efficient tool for the prediction of full-length LTR retrotransposons. *Nucleic Acids Res.*, **35**, W265–268.

Zeng,F. *et al.* (2017) LTRtype, an efficient tool to characterize structurally complex LTR retrotransposons and nested insertions on genomes. *Front. Plant Sci.*, **8**, 402.

F Lexa et al., 2022

Bioinformatics, 38(16), 2022, 4030–4032
https://doi.org/10.1093/bioinformatics/btac442
Advance Access Publication Date: 4 July 2022
Applications Note

Genome analysis

HiC-TE: a computational pipeline for Hi-C data analysis to study the role of repeat family interactions in the genome 3D organization

Matej Lexa [1,2,*], Monika Cechova[1], Son Hoang Nguyen[1], Pavel Jedlicka[2], Viktor Tokan[2], Zdenek Kubat[2], Roman Hobza[2] and Eduard Kejnovsky[2,*]

[1]Faculty of Informatics, Masaryk University, 60200 Brno, Czech Republic and [2]Department of Plant Developmental Genetics, Institute of Biophysics of the Czech Academy of Sciences, 61200 Brno, Czech Republic

*To whom correspondence should be addressed.

Associate Editor: Can Alkan

Received on February 15, 2022; revised on June 14, 2022; editorial decision on June 28, 2022; accepted on June 30, 2022

Abstract

Motivation: The role of repetitive DNA in the 3D organization of the interphase nucleus is a subject of intensive study. In studies of 3D nucleus organization, mutual contacts of various loci can be identified by Hi-C sequencing. Typical analyses use binning of read pairs by location to reduce noise. We use binning by repeat families instead to make similar conclusions about repeat regions.

Results: To achieve this, we combined Hi-C data, reference genome data and tools for repeat analysis into a Nextflow pipeline identifying and quantifying the contacts of specific repeat families. As an output, our pipeline produces heatmaps showing contact frequency and circular diagrams visualizing repeat contact localization. Using our pipeline with tomato data, we revealed the preferential homotypic interactions of ribosomal DNA, centromeric satellites and some LTR retrotransposon families and, as expected, little contact between organellar and nuclear DNA elements. While the pipeline can be applied to any eukaryotic genome, results in plants provide better coverage, since the built-in TE-greedy-nester software only detects tandems and LTR retrotransposons. Other repeats can be fed via GFF3 files. This pipeline represents a novel and reproducible way to analyze the role of repetitive elements in the 3D organization of genomes.

Availability and implementation: https://gitlab.fi.muni.cz/lexa/hic-te/.

Contact: lexa@fi.muni.cz

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

The eukaryotic genome is hierarchically packed in the nucleus allowing DNA replication and gene transcription to take place in a spatially and temporally regulated fashion. A significant part of eukaryotic genomes is made up of transposable elements (TEs) and satellite DNA, where e.g. LTR retrotransposons constitute up to 90% of genomes in some species (Liehr, 2021; Schnable *et al.*, 2009; Wicker *et al.*, 2018). TEs are often embedded in cellular regulatory networks (Feschotte, 2008) where they rewire the gene expression programs (Slotkin and Martienssen, 2007). Many examples of the domestication of TEs for specific cellular functions have been observed (Jangam *et al.*, 2017; Sinzelle *et al.*, 2009). Moreover, the 3D organization of the interphase nucleus is recently a subject of intensive study.

Methods of high-throughput mapping of DNA–DNA interactions, such as chromosome conformation capture (Hi-C), now allow the study of long-distance interactions in eukaryotic nuclei. Because of technical issues, these have mostly avoided repetitive parts of the genome. A better understanding of the interaction of the main repeat classes can help uncover their genomic role. A recent study demonstrated the role of TEs in organizing the human and mouse genomes (Lu *et al.*, 2021) and similar analysis in other organisms is hitherto missing.

Here, we present a new sequence processing pipeline to identify and quantify interactions of TEs, satellite DNA and rDNA in nuclei, especially those that participate in long-distance (≥1 Mb) or interchromosomal contacts with frequencies that differ from baseline expectations of randomness.

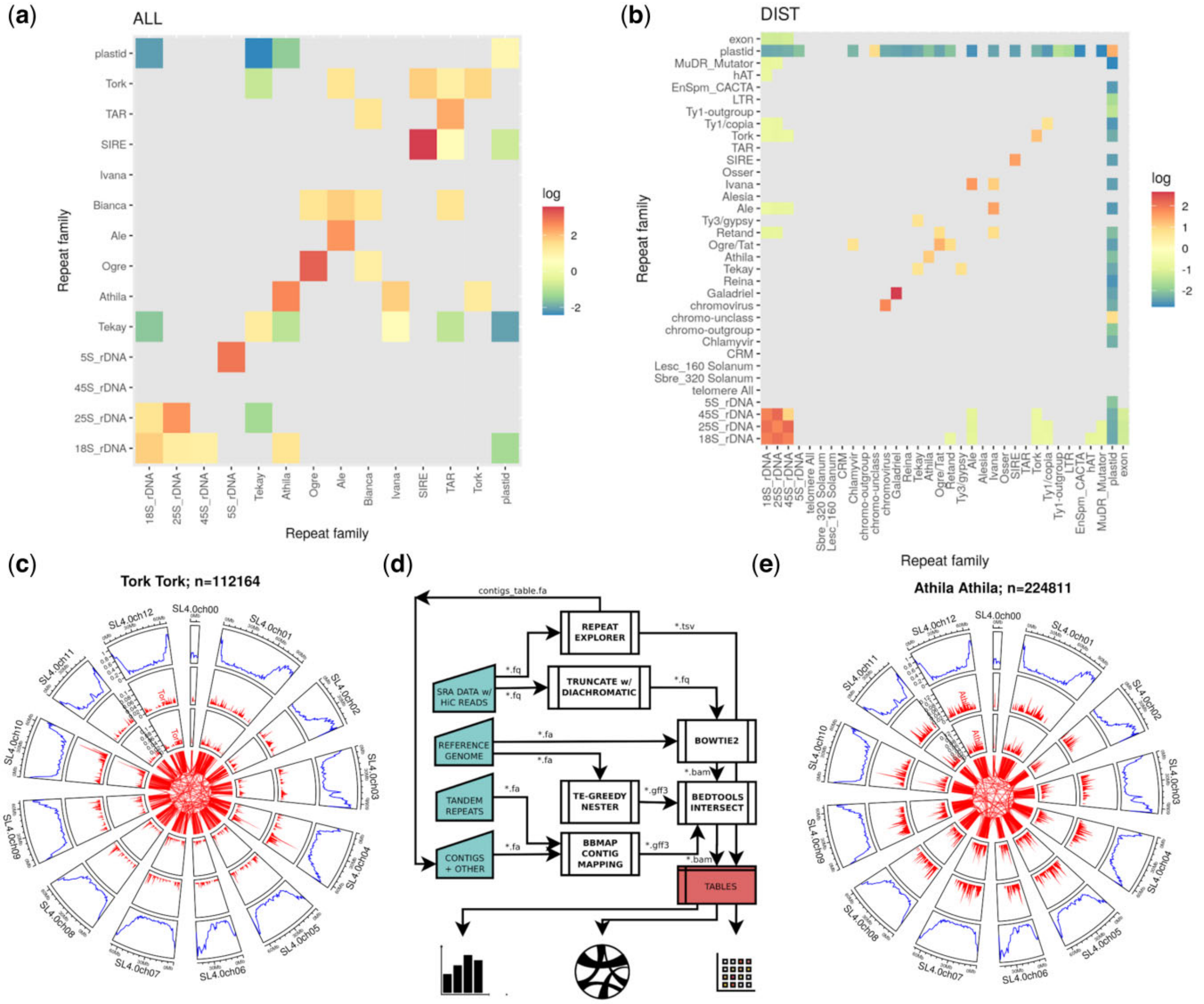


Fig. 1. HiC-TE pipeline. (**a**) Reference-free and (**b**) reference-based heatmaps generated by the pipeline for SRR5748725, showing repeat family pairs and their label-permutation normalized contact frequency; (**c**, **e**) Circos plots for the same showing chromosomal locations of representative contacts for the given family pair; (**d**) block diagram showing the overall data flow in the HiC-TE pipeline. Some details were omitted for clarity (the full graph produced by the pipeline is shown in Supplementary Fig. S1) [left—main data inputs; bottom—main data outputs; double edged rectangles—main processes running external tools; FASTA (*.fa), FASTQ (*.fq), BAM, GFF3—main sequence and annotation data formats passed between processes]

2 Nextflow pipeline description and testing

Our pipeline (Fig. 1d, Supplementary Fig. S1) integrates sequence analysis from several sources: assembled genome repeat annotation [TE-greedy-nester (Lexa *et al.*, 2020), PlantSat database (Macas *et al.*, 2002)], medium and long-distance contact information (Hi-C experiments) and repetitive NGS read clustering [Repeat Explorer (Novak *et al.*, 2013)]. TE-greedy-nester is our previously developed tool for structure-based detection of LTR retrotransposons that can even assign fragmented TEs to their families. However, a large number of tools in this area exist as recently reviewed by Rodriguez and Makałowski (2022). Currently, alternative annotations from such sources can be applied via a GFF3 file or consensus sequences that will be mapped to reference by the pipeline (see 'DATA' and 'OPTIONAL PARAMETERS' sections in source code README.md). Repeat Explorer is a popular and time-tested solution for graph-clustering repetitive sequencing reads without the need for a reference genome. We included these two tools to complement each other, as each method has different strengths. For example, many reference genomes lack repetitive regions that are hard to assemble, a situation in which reference-free approach might be advantageous, although the quality of reference genomes is gradually improving with T2T sequencing (Nurk *et al.*, 2022). The pipeline was implemented with Nextflow (Di Tommaso *et al.*, 2017) to allow for flexibility and scalability, using a recent installation of Ubuntu Linux with all dependencies included. In addition, we provide a tested containerized version allowing runs with Docker/Singularity deployment (see 'RUNNING THE PIPELINE' in source code README.md). As a result, all the figures and tables are fully reproducible and can be easily generated. We summarize the memory, disk and time requirements in Supplementary Tables S1 and S2 and Supplementary Figures S2–S4.

To test 'HiC-TE', we used a publicly available dataset on the tomato (*Solanum lycopersicum*) (Dong *et al.*, 2017) with two technical replicates for each of three plants. The minimal input dataset represents a HiC experiment (FASTQ) with the name of the restriction enzyme used in the protocol, a reference genome sequence (FASTA), exon annotations (GFF3) and a set of tandem repeats and satellite DNA to be mapped (PlantSat FASTA) with the telomeric repeat sequence array as a minimum. We verified that the pipeline produces consistent results and that the computational replicates are less variable than any other replicates. We analyzed Hi-C contacts from reads clustered with Repeat Explorer (Fig. 1a) and reference-mapped long-distance interactions (spanning ≥1 Mb or between sequences located on different chromosomes) (Fig. 1b). The main output is a series of heatmaps showing high/low values of normalized contacts in diverging colors, while fields (repeat family pairs) with missing values are shown in gray. The pipeline also relies on initial trimming of raw reads with Diachromatic (Hansen *et al.*, 2019), read mapping via Bowtie2 (Langmead and Salzberg, 2012) and BBmap, overlap/intersection analysis with bedtools (Quinlan and Hall, 2010) and data manipulation and visualization in R/Bioconductor with extra packages (Supplementary Note S1) (Gel and Serra, 2017; Gu *et al.*, 2014; Indahl *et al.*, 2018; Pedersen and Shemanarev, 2020; Wickham, 2007). Before visualization in heatmaps, the data are normalized using three different normalization techniques (Supplementary Note S2) to account for background HiC signal and the fact that repeat families have varying frequencies. Circos plots allow to understand chromosomal localization of the contacts, such as the chromosome number or whether it is in a gene-rich or gene-poor area (Fig. 1c and e). Normalized values that are too close to 1, or based on samples with a low number of reads (configurable by the user) are shown as gray fields in these heatmaps.

3 Discussion

The pipeline contains two modes of repeat annotation, reference-based and reference-free. While reference-based data contain chromosomal positions and allow the calculation of distances, the reference-free mode avoids the necessity to discern real and apparent read mapping, which is especially problematic when dealing with repeats and short reads. Our tool contrasts traditional methods of binning HiC contacts (Golicz *et al.*, 2020; Sun *et al.*, 2020; Zheng *et al.*, 2019) (for the discussion see Supplementary Note S3). It has a potential, based on frequency of interactions of specific centromeric or telomeric repeats, to reveal distinct local organizations of chromosomes, such as Rabl, Rosette or Bouquet arrangement (Tiang *et al.*, 2012) (see Supplementary Note S4 for further discussion of biological relevancy).

The focus of traditional HiC data analysis pipelines on unique mapping made us realize that multiple-mapping reads could still be assigned to a family of repeats. To this end, we built a HiC read mapping pipeline that explores this possibility in tandem with Repeat Explorer software, producing a report of likely interaction partners among repeat families. Another branch of our computations produces similar output relying on reads that Bowtie2 maps to annotated repeats. This Nextflow pipeline can identify and quantify the contacts of specific repeats in the 3D nucleus. Using real biological data from public databases we have shown that with proper normalization techniques, known (and possibly also unknown) interaction partners can be revealed among annotated repeat families.

Acknowledgements

We thank Christopher Johnson for critical reading of the manuscript, Jan Hoidekr for help with deployment of the pipeline in the MetaCentrum environment. Computational resources were supplied by the project ‘e-Infrastruktura CZ’ (e-INFRA CZ LM2018140) supported by the Ministry of Education, Youth and Sports of the Czech Republic. M.C. is the holder of Martina Roeselova Memorial Fellowships 2020.

Funding

This work was supported by the Czech Science Foundation [21-00580S].

Conflict of Interest: none declared.

Data availability

The data underlying this article are available in Zenodo, at: https://dx.doi.org/10.5281/zenodo.6628770 (pipeline info and output) and https://dx.doi.org/10.5281/zenodo.6628543 (source code freeze).

References

Di Tommaso,P. *et al.* (2017) Nextflow enables reproducible computational workflows. *Nat. Biotechnol.*, **35**, 316–319.

Dong,P. *et al.* (2017) 3D chromatin architecture of large plant genomes determined by local a/B compartments. *Mol. Plant.*, **10**, 1497–1509.

Feschotte,C. (2008) Transposable elements and the evolution of regulatory networks. *Nat. Rev. Genet.*, **9**, 397–405.

Gel,B. and Serra,E. (2017) KaryoploteR: an R/Bioconductor package to plot customizable genomes displaying arbitrary data. *Bioinformatics*, **33**, 3088–3090.

Golicz,A. *et al.* (2020) Rice 3D chromatin structure correlates with sequence variation and meiotic recombination rate. *Commun. Biol.*, **3**, 235.

Gu,Z. *et al.* (2014) Circlize implements and enhances circular visualization in R. *Bioinformatics*, **30**, 2811–2812.

Hansen,P. *et al.* (2019) Computational processing and quality control of Hi-C, capture Hi-C and Capture-C data. *Genes*, **10**, 548.

Indahl,U. *et al.* (2018) A similarity index for comparing coupled matrices. *J. Chemom.*, **32**, e3049.

Jangam,D. *et al.* (2017) Transposable element domestication as an adaptation to evolutionary conflicts. *Trends Genet.*, **33**, 817–831.

Langmead,B. and Salzberg,S. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**, 357–359.

Lexa,M. *et al.* (2020) TE-greedy-nester: structure-based detection of LTR retrotransposons and their nesting. *Bioinformatics*, **36**, 4991–4999.

Liehr,T. (2021) Repetitive elements in humans. *Int. J. Mol. Sci.*, **22**, 2072.

Lu,J. *et al.* (2021) Homotypic clustering of L1 and B1/Alu repeats compartmentalizes the 3D genome. *Cell Res.*, **31**, 613–630.

Macas,J. *et al.* (2002) PlantSat: a specialized database for plant satellite repeats. *Bioinformatics*, **18**, 28–35.

Novak,P. *et al.* (2013) RepeatExplorer: a Galaxy-based web server for genome-wide characterization of eukaryotic repetitive elements from next-generation sequence reads. *Bioinformatics*, **29**, 792–793.

Nurk,S. *et al.* (2022) The complete sequence of a human genome. *Science*, **376**, 44–53.

Pedersen,T. and Shemanarev,M. (2020) *ragg: graphic devices based on AGG. R package version 1.1.3*. https://ragg.r-lib.org, https://github.com/r-lib/ragg.

Quinlan,A. and Hall,I. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, **26**, 841–842.

Rodriguez,M. and Makałowski,W. (2022) Software evaluation for de novo detection of transposons. *Mob. DNA*, **13**, 14.

Schnable,P. *et al.* (2009) The B73 maize genome: complexity, diversity, and dynamics. *Science*, **326**, 1112–1115.

Sinzelle,L. *et al.* (2009) Molecular domestication of transposable elements: from detrimental parasites to useful host genes. *Cell. Mol. Life Sci.*, **66**, 1073–1093.

Slotkin,R. and Martienssen,R. (2007) Transposable elements and the epigenetic regulation of the genome. *Nat. Rev. Genet.*, **8**, 272–285.

Sun,L. *et al.* (2020) Heat stress-induced transposon activation correlates with 3D chromatin organization rearrangement in Arabidopsis. *Nat. Commun.*, **11**, 1886.

Tiang,C.-L. *et al.* (2012) Chromosome organization and dynamics during interphase, mitosis, and meiosis in plants. *Plant Physiol.*, **158**, 26–34.

Wicker,T. *et al.* (2018) Impact of transposable elements on genome structure and evolution in bread wheat. *Genome Biol.*, **19**, 103.

Wickham,H. (2007) Reshaping data with the reshape package. *J. Stat. Softw.*, **21**, 1–20.

Zheng,Y. *et al.* (2019) Generative modeling of multi-mapping reads with mhi-c advances analysis of hi-c studies. *eLife*, **8**, 1141–1156.