Bioinformatics, 2017, 1–7
doi: 10.1093/bioinformatics/btx413
Advance Access Publication Date: 4 July 2017
Original Paper

OXFORD

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.

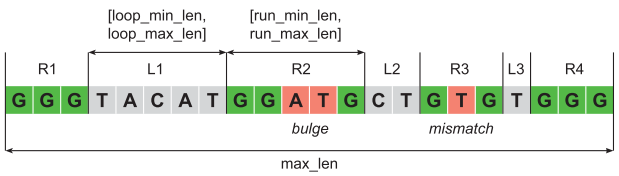

Fig. 1. PQS constraints. Every PQS consists of two types of elements: G runs (R1–4) and loops (L1–3). The minimal and maximal length of each element type is constrained by the corresponding options depicted in the picture as well as the overall PQS length. All these options can be freely customized when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

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

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.

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

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.

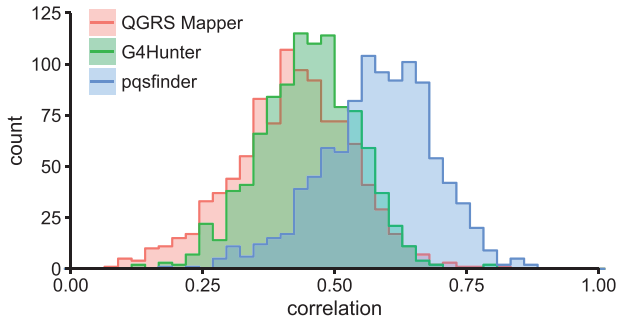

Fig. 2. Histogram of correlation coefficients for QGRS Mapper, G4Hunter and *pqsfinder* on the G4-seq test set fragments. The correlation was measured between the averaged level of mismatches of the G4-seq test set fragments (see Section 4.2) and the vector of maximal scores predicted by each tool. While 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.