Analysis of imperfect palindromes

Georgii A. Khaziev[1]

[1]PhD student at IITP RAS, Moscow, 1.5.8 Mathematical biology and bioinformatics, physical-mathemtical sciences

December 17, 2025

Today is a special day

I would like to formaly congratulate my scientific supervisor Vassily Alexandrovich Lyubetsky with his 80-th birthday!!!

Palindromes in Bioinformatics.

Let us consider nucleotide sequences. The nucleotides $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ form two complementary pairs:

$$\mathrm{c}(\mathtt{A}) = \mathtt{T},\ \mathrm{c}(\mathtt{T}) = \mathtt{A},\ \mathrm{c}(\mathtt{C}) = \mathtt{G},\ \text{and}\ \mathrm{c}(\mathtt{G}) = \mathtt{C}.$$

Next, $\mathrm{c}()$ denotes the reverse complement, i. e., $\mathrm{c}(xy) = \mathrm{c}(y)\mathrm{c}(x)$. For example, $\mathrm{c}(\mathtt{AACG}) = \mathtt{CGTT}$. In DNA, a perfect palindrome is an inverted sequence repeat, i. e., reverse complement of itself. Let us omit the concatenation symbol. So, all perfect palindromes are of the type $x\mathrm{c}(x)$, where x denotes a sequence. In particular, a sequence of odd length cannot be any perfect palindrome.

Needleman-Wunsch algorithm

let x and y be nucleotide sequences. Then to find edit distance between x and y we compute

$$f(j,k) = \begin{cases} j, \ j \geq 0, k = 0 \\ k, \ j = 0, k \geq 0 \\ \min\{f(j,k-1)+1, f(j-1,k)+1, f(j-1,k-1)+s(j,k)\}, \ j > 0, k > 0 \end{cases}$$

where $s(j,k) \in \{0,1\}$ depends on elements of x and y:

$$s(j,k) = \begin{cases} 0, & x_j = y_k \\ 1, & x_j \neq y_k \end{cases}$$

Needleman-Wunsch example

Let $x =$ ACCAT and $y =$ AGGA, then we compute alignment matrix.

		A	C	C	A	T
	0	1	2	3	4	5
A	1	0	1	2	3	4
G	2	1	1	2	3	4
G	3	2	2	2	2	4
A	4	3	3	3	3	3

Edit distance between x and y is at bottom-right corner and equals 3.

Theorem 1

Theorem (Zverkov et al., 2024)

There exists an algorithm that for input sequences x and y computes optimal partition of y to such concatenation $y = wz$ that edit distance between x and a palindrome $w\mathrm{c}(w)$ is minimal. In the same time edit distance between x and optimal palindrome $w\mathrm{c}(w)$ is also computed. Overall algebraic complexity is $O(|x||y|)$.

Proof of Theorem 1

To compute edit distance between x and y it is enough to compute $f(|x|, |y|)$ of function f, recurrently defined as

$$f(j,k) = \begin{cases} j, \ j \geq 0, k = 0 \\ k, \ j = 0, k \geq 0 \\ \min\{f(j,k-1)+1, f(j-1,k)+1, f(j-1,k-1)+s(j,k)\}, \ j > 0, k > 0 \end{cases}$$

where $s(j,k) \in \{0,1\}$ depends on elements of x and y:

$$s(j,k) = \begin{cases} 0, & x_j = y_k \\ 1, & x_j \neq y_k \end{cases}$$

$f(j,k)$ allow us to compute edit distance between prefixes of x and y with lengths j and k.

Proof of Theorem 1

Now let's compute edit distance between x and $\mathrm{c}(y)$. Let us define recursive function $g(j,k)$ as follows

$$g(j,k) = \begin{cases} j, \ j \geq 0, k = 0 \\ k, \ j = 0, k \geq 0 \\ \min\{g(j,k-1)+1, g(j-1,k)+1, \\ g(j-1,k-1)+r(j,k)\}, \ j > 0, k > 0, \end{cases}$$

where $r(j,k) \in \{0,1\}$ depends on elements of x and y:

$$r(j,k) = \begin{cases} 0, & x_{m-j+1} = \mathrm{c}(y_k) \\ 1, & x_{m-j+1} \neq \mathrm{c}(y_k) \end{cases}$$

Edit distance between x and $\mathrm{c}(y)$ equals $g(|x|,|y|)$.

Proof of Theorem 1

Let h be defined as

$$h(j,k) = f(j,k) + g(|x| - j, k).$$

A value of $h(j,k)$ equals sum of two edit distances. First distance is between prefixes of x and y, and a second one is between corresponding suffixes of x and $\mathrm{c}(y)$. yuEdit distance between x and $w\mathrm{c}(w)$ equals minimal value of h for $0 \le j \le |x|$ and $0 \le k \le |y|$. At the same time corresponding value of k, defines the partition $y = wz$.

imp function

Let us denote by $|x|$ the length of x. Let us denote by $\mathrm{imp}(x)$ the ratio of the minimum edit distance between sequence x and optimal palindrome to the length of the sequence:

$$\mathrm{imp}(x) = \frac{\min\{\mathrm{dist}(x, wc(w)) | x = wz\}}{|x|}.$$

Example

For $x = \mathrm{ATATGT}$, $\mathrm{imp}(x) = \frac{1}{6}$

The ratio shows how imperfect the palindrome is. $\mathrm{imp}(x)$ could be easily computed using algorithm from Theorem 1[1].

[1] If $y = x$, then $f(j, k)$ could be easily obtained as $f(j, k) = |j - k|$.

Theorem 2

Theorem

$\mathrm{imp}(x) = \mathrm{imp}(\mathrm{c}(x))$.

It is enough to show that for any $x = wz$ equalities

$$\mathrm{dist}(x, w\mathrm{c}(w)) = \mathrm{dist}(x, \mathrm{c}(z)z) = \mathrm{dist}(\mathrm{c}(x), \mathrm{c}(z)z)$$

are satisfied. Indeed,

$$\mathrm{dist}(x, w\mathrm{c}(w)) = \mathrm{dist}(z, \mathrm{c}(w)),$$

since prefixes are matching and

$$\mathrm{dist}(x, \mathrm{c}(z)z) = \mathrm{dist}(w, \mathrm{c}(z)),$$

since suffixes are also matched. Also, since dist is symmetrical function it is true that

$$\mathrm{dist}(z, \mathrm{c}(w)) = \mathrm{dist}(\mathrm{c}(w), z).$$

Proof of Theorem 2

And because of invariance of dist under simultaneous replacement of both arguments to their reverse complements, it is true that

$$\mathrm{dist}(\mathrm{c}(w), z) = \mathrm{dist}(w, \mathrm{c}(z)).$$

Thus, first equality is satisfied

$$\mathrm{dist}(x, w\mathrm{c}(w)) = \mathrm{dist}(z, \mathrm{c}(w)) = \mathrm{dist}(\mathrm{c}(w), z) = \mathrm{dist}(w, \mathrm{c}(z)) = \mathrm{dist}(x, \mathrm{c}(z)z).$$

The second equality

$$\mathrm{dist}(x, \mathrm{c}(z)z) = \mathrm{dist}(\mathrm{c}(x), \mathrm{c}(z)z)$$

is true for any z, since $\mathrm{c}(z)z$ is a palindrome (i.e. $\mathrm{c}(\mathrm{c}(z)z) = \mathrm{c}(z)z$).

Theorem 3

Theorem

For all even-length sequences it is true that $\mathrm{imp}(x) \leq 1/2$. *For all odd-length sequences it is true that* $0 < \mathrm{imp}(x) \leq (1 + 1/|x|)/2$.

For x of even length and prefix w which is equal half of x,

$$\mathrm{dist}(x, w) = \frac{|x|}{2}.$$

For odd-length sequence x and prefix w, which is limited by half of x, but don't contain the middle symbol,

$$\mathrm{dist}(x, w) = \frac{|x| + 1}{2}.$$

In any of the cases it's true that

$$\mathrm{dist}(x, w\mathrm{c}(w)) \leq \mathrm{dist}(x, w).$$

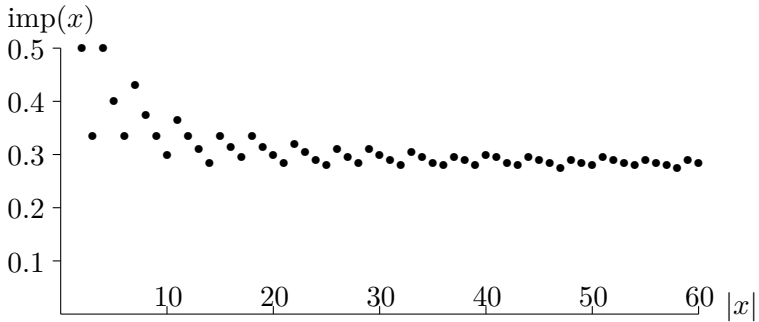


Figure: Empirical dependency between median of $\mathrm{imp}(x)$ and $|x|$ for random nucleotide sequences for $gc = 0.5$.

median of imp

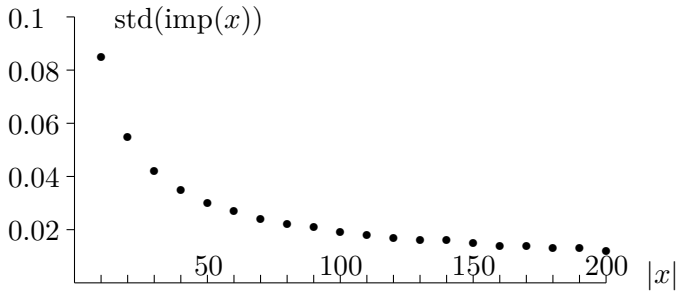


Figure: Empirical dependency between standard deviation of $\mathrm{imp}(x)$ and $|x|$ for random nucleotide sequences.

trimmers

Substring is a contiguous sequence of characters within a string. The main idea behind the algorithm to select an imperfect palindrome is checking whether one of the optimal lengths of the prefix w of the input sequence x differs significantly from $|x|/2$. If such case occurs, then the algorithm deletes either prefix or suffix by difference between $|w|$ and $|x|/2$.

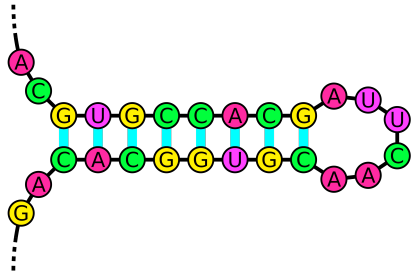


Figure: Hairpin example.

trimmers

All three functions `pref_trimmer`, `suff_trimmer`, and `double_trimmer` take as input nucleotide sequence x, sorted list `optimal_lengths` of optimal prefix lengths $|w|$, and floating-point number `cutoff_condition`. Note that $\min(\texttt{optimal\_lengths})$ and $\max(\texttt{optimal\_lengths})$ are the first and last elements of `optimal_lengths`, respectively.

The `pref_trimmer` function trims first rd symbols of x, where

$$rd = \max(\texttt{optimal\_lengths}) - \left\lfloor \frac{|x|}{2} \right\rfloor,$$

when $rd \geq \text{length}(x) \cdot \texttt{cutoff\_condition}$ is satisfied.
The `suff_trimmer` function trims last ld symbols of x, where

$$ld = \left\lfloor \frac{|x|}{2} \right\rfloor - \min(\texttt{optimal\_lengths}),$$

when $ld \geq |x| \cdot \texttt{cutoff\_condition}$ is satisfied.

double_trimmer

The `double_trimmer` function initially computes

$$rd = \max(\texttt{optimal\_lengths}) - \left\lfloor \frac{|x|}{2} \right\rfloor$$

and

$$ld = \left\lfloor \frac{|x|}{2} \right\rfloor - \min(\texttt{optimal\_lengths}).$$

Subsequently it checks if

$$\begin{cases} rd \geq |x| \cdot \texttt{cutoff\_condition} \\ ld \geq |x| \cdot \texttt{cutoff\_condition} \end{cases}$$

If the first inequality is satisfied, the function trims the first rd symbols from the string x. Similarly, if the second inequality is satisfied, the function trims the last ld symbols from x.

Theorem

For perfect palindrome x for any `cutoff_condition` > 0 *no trimming would be performed.*

Let x be a perfect palindrome. Then its only optimal partition would be exactly at

$$\left\lfloor \frac{|x|}{2} \right\rfloor$$

Then

$$ld = \left\lfloor \frac{|x|}{2} \right\rfloor - \left\lfloor \frac{|x|}{2} \right\rfloor = 0$$

and

$$rd = \left\lfloor \frac{|x|}{2} \right\rfloor - \left\lfloor \frac{|x|}{2} \right\rfloor = 0.$$

For a non-zero `cutoff_condition` both rd and ld would be less than $|x| \cdot$ `cutoff_condition` so no cutoff condition would be met.

Theorem (Theorem 5)

For any $n \geq 3$, there exist `cutoff_condition` > 0 *and x with length of at least n, which satisfies both prefix and suffix trimming conditions such that*

$$\texttt{pref\_trimmer}(x_{suff}) \neq \texttt{suff\_trimmer}(x_{pref}),$$

where x_{suff} and x_{pref} are results of suffix and prefix trimming of x, respectively.

Let

$$x = \underbrace{\texttt{ATA...ATA}}_{\alpha},$$

where $\alpha = 2\left\lfloor \frac{n}{2} \right\rfloor + 1$, i.e. minimal odd number, not smaller than n. For such x optimal partitions would be precisely at $\left\lceil \frac{n}{2} \right\rceil - 1$ and $\left\lceil \frac{n}{2} \right\rceil + 1$. Minimal index of optimal partition would correspond to the prefect palindrome

$$\underbrace{\texttt{AT...AT}}_{\alpha-1}$$

Proof of Theorem 5

and maximal index would correspond to perfect palindrome

$$\underbrace{\texttt{AT...AT}}_{\alpha+1}.$$

Let $\texttt{cutoff\_condition} = \frac{1}{10\alpha}$. Since this value satisfies both suffix and prefix trimming conditions, x_{suff} and x_{pref} could be computed and would be equal to

$$\underbrace{\texttt{AT...AT}}_{\alpha-1}$$

and

$$\underbrace{\texttt{T...ATA}}_{\alpha-1}$$

respectively. Both of those sequences are perfect palindromes and would not be trimmed due to the previous theorem.

Another trimmers

We've also developed trimmer algorithms based on another approach. `pref_GRT`, `suff_GRT`, and `double_GRT` recursively trim `cutoff_value` $\cdot\, |x|$ symbols from x for a given `recursion_depth` number of iterations. On each iteration they check, whether value of imp decreased, if not, they decrease `cutoff_value` by half and return substring x' with minimal value of imp that was obtained.

de_shapker

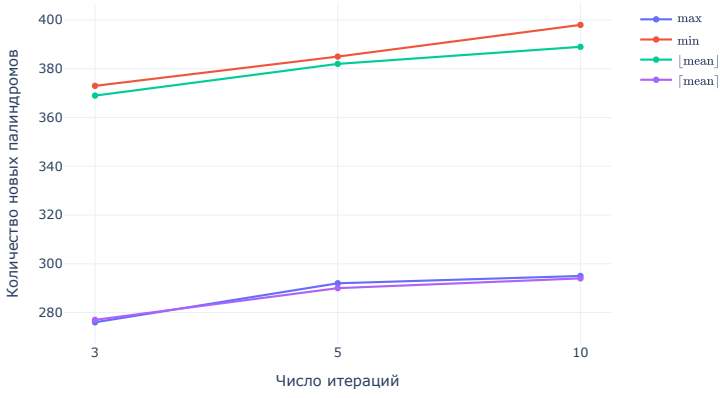
To find a noncomplementary substring(loop) inside x we've developed new algorithm called de_shapker. It takes as an input a string x, a floating point value imp_given, which equals $\mathrm{imp}(x)$, matrix H of values of function $h(j,k)$ from palindrome_self_alignment(x) algorithm, integers iteration_counter, window_delta and window_size, and a Python function strategy.
Let $x_{result} = x$. On each iteration de_shapker algorithm computes l_1 norms of all continuous submatrices $S \in \mathbb{R}^{\texttt{window\_size} \times \texttt{window\_size}}$ of H defined as

$$l_1(S) = \sum_{i=1}^{\texttt{window\_size}} \sum_{i=j}^{\texttt{window\_size}} (S)_{ij} \tag{1}$$

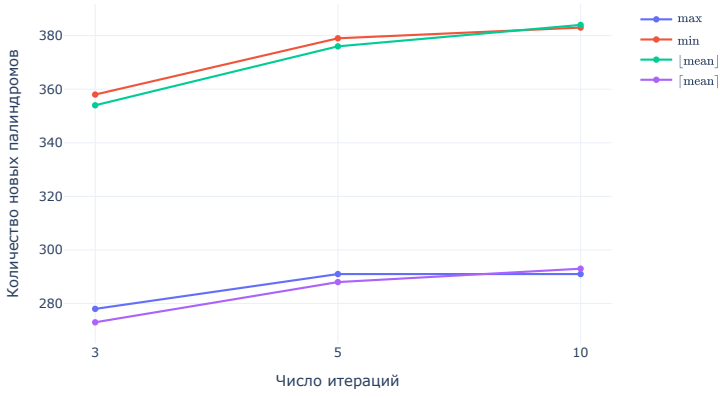
de_shapker

After that, the submatrix $S_{\min}$ with minimal norm is chosen. Then, the index i of the beginning of potential loop in x is computed as $i = \texttt{strategy}(u, v)$, where u and v are coordinates of top-left element of $S_{\min}$ in H. We recommend to choose as a `strategy` a function biased towards minimal value between u and v (e.g. $\min(u, v)$ or $\lfloor \mathrm{mean(u, v)} \rfloor$) as they tend to capture loops more precisely. Subsequently, x_b is computed as x with cutted symbols from i to $i + \texttt{window\_size}$ positions. If $\mathrm{imp}(x_b) < \texttt{imp\_given}$, then x_{result} will be overwritten with x_b, and `imp_given` will be overwritten with $\mathrm{imp}(x_b)$. Next, the `window_size` is increased by `window_delta` and the next iteration starts. If $\mathrm{imp}(x_b) \geq \texttt{imp\_given}$, then algorithm returns x_{result}.After all iterations, algorithm returns x_{result}.

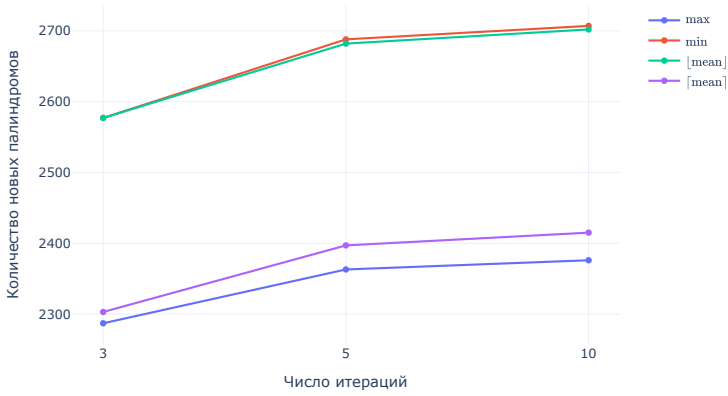
On real data


400
380
360
340
320
300
280
Количество новых палиндромов
3
5
10
Число итераций
max
min
⌊mean⌋
⌈mean⌉

On real data

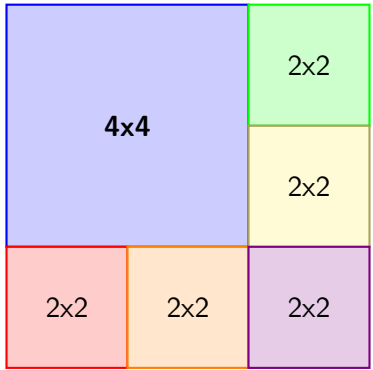

Количество новых палиндромов
380
360
340
320
300
280
3
5
10
Число итераций
max
min
⌊mean⌋
⌈mean⌉

On real data


2700
2600
2500
2400
2300
Количество новых палиндромов
3
5
10
Число итераций
max
min
⌊mean⌋
⌈mean⌉

Boosting de_shapker

Note that on each iteration we recompute $l_1(S)$ increasing size of S each time by `window_delta`. But if we will use `window_size` = `window_delta` for the first iteration, then new l_1 norm could be computed from previous with addition of l_1 norms of the first iteration.



Boosting `de_shapker`

To implement this we need to store 3 square matrices of length $n - \texttt{window\_size} + 1$, where $|x| = n$ and by `window_size` we mean input value of `window_size`. The first matrix would be used to store initial norms, second and third matrices would be previous and next step norms by rotation. After computing new norms matrix that stores previous ones would be multiplied by infinity.

Wobbly palindromes

In RNA hairpin is sometimes formed with the use of non-complementary pair GU[2]. Such bond is called wobbly. To address this me and my student S. Mankevich implemented wobbly version of algorithm from theorem 1 with changed value of function

$$r(j,k) = \begin{cases} 0, & x_{m-j+1} = \mathrm{c}(y_k) \vee (\mathrm{c}(y_k), x_{m-j+1}) \in \{(\mathtt{C},\mathtt{U}), (\mathtt{U},\mathtt{C}), (\mathtt{A},\mathtt{G}), (\mathtt{G},\mathtt{A})\} \\ 1, & otherwise \end{cases}$$

Such modification allows us to compute more weak distance and $\mathrm{imp}_{\mathrm{w}}$ functions, but with accounting of wobbly pair.
We've tested this algorithm on mirBase dataset and from 38.5K RNA hairpins for less than 30 we couldn't find substrings with value $\mathrm{imp}_{\mathrm{w}} \leq 0.2$ after trimming. While on the same dataset for $\mathrm{imp} \leq 0.2$ couldn't be found for about 8K of sequences.

[2] in RNA T is replaced by U

Conclusion

1. New algorithms for picking a close-to-palindrome subsequence of x are introduced.
2. Modification of algorithm for finding distance to perfect palindrome with respect to RNA wobbly pairs were made.
3. Those algorithms were tested on synthetic and real DNA and RNA data.

References

- Zhao X.X., Wang Z.X., Tang D. et al. *The expectation and the variance of the weights of de Bruijn sequences*, Des. Codes Cryptogr. 2025. https://doi.org/10.1007/s10623-025-01729-2
- Nazipova N.N. *Application of suffix arrays to detect repeats in genomic sequences*, Mathematical Biology and Bioinformatics, 2025. V. 20 I. 2. P. 348-362. https://doi.org/10.17537/2025.20.348
- Zverkov O., Seliverstov A., Shilovsky G. *Alignment of a Hidden Palindrome*, Mathematical biology and bioinformatics. 2024. V.19. I.2. P. 427-438. https://doi.org/10.17537/2024.19.427
- Khaziev G.A., Seliverstov A.V., Zverkov O.A. *Searching for an imperfect palindrome*, Computer algebra: 6th International Conference Materials, Moscow, Russia, June 23–25 2025, RUDN University, 2025, P. 62–65.