Taylor & Francis
Taylor & Francis Group

Biotechnology & Biotechnological Equipment

ISSN: 1310-2818 (Print) 1314-3530 (Online) Journal homepage: http://www.tandfonline.com/loi/tbeq20

TMO: time and memory optimized algorithm applicable for more accurate alignment of trinucleotide repeat disorders associated genes

Done Stojanov, Ana Madevska Bogdanova & Tomasz Marcin Orzechowski

To cite this article: Done Stojanov, Ana Madevska Bogdanova & Tomasz Marcin Orzechowski (2016) TMO: time and memory optimized algorithm applicable for more accurate alignment of trinucleotide repeat disorders associated genes, Biotechnology & Biotechnological Equipment, 30:2, 388-403, DOI: 10.1080/13102818.2015.1114428

To link to this article: http://dx.doi.org/10.1080/13102818.2015.1114428

Published online: 08 Feb 2016.

BIOTECHNOLOGY & BIOTECHNOLOGICAL EQUIPMENT, 2016
VOL. 30, NO. 2, 388–403
http://dx.doi.org/10.1080/13102818.2015.1114428

Taylor & Francis
Taylor & Francis Group

ARTICLE; BIOINFORMATICS

OPEN ACCESS

TMO: time and memory optimized algorithm applicable for more accurate alignment of trinucleotide repeat disorders associated genes

Done Stojanov[a], Ana Madevska Bogdanova[b] and Tomasz Marcin Orzechowski[c]

[a]Faculty of Computer Science, Department of Computer Technologies and Intelligent Systems, University "Goce Delcev", Štip, Republic of Macedonia; [b]Faculty of Computer Sciences and Engineering, Department of Intelligent Systems, University "Sts Cyril and Methodius", Skopje, Republic of Macedonia; [c]Faculty of Computer Science, Electronics and Telecommunications, Department of Telecommunications, AGH University of Science and Technology, Krakow, Poland

ABSTRACT

In this study, time and memory optimized (TMO) algorithm is presented. Compared with Smith–Waterman's algorithm, TMO is applicable for a more accurate detection of continuous insertion/deletions (indels) in genes' fragments, associated with disorders caused by over-repetition of a certain codon. The improvement comes from the tendency to pinpoint indels in the least preserved nucleotide pairs. All nucleotide pairs that occur less frequently are classified as less preserved and they are considered as mutated codons whose mid-nucleotides were deleted. Other benefit of the proposed algorithm is its general tendency to maximize the number of matching nucleotides included per alignment, regardless of any specific alignment metrics. Since the structure of the solution, when applying Smith–Waterman, depends on the adjustment of the alignment parameters and, therefore, an incomplete (shortened) solution may be derived, our algorithm does not reject any of the consistent matching nucleotides that can be included in the final solution. In terms of computational aspects, our algorithm runs faster than Smith–Waterman for very similar DNA and requires less memory than the most memory efficient dynamic programming algorithms. The speed up comes from the reduced number of nucleotide comparisons that have to be performed, without having to imperil the completeness of the solution. Due to the fact that four integers (16 Bytes) are required for tracking matching fragment, regardless its length, our algorithm requires less memory than Huang's algorithm.

ARTICLE HISTORY
Received 29 June 2015
Accepted 27 October 2015

KEYWORDS
Huntington's disease; genes; computational; improved; algorithm

Introduction

In order to detect mutual similarities (differences) between two or more biological sequences (DNA, RNA or protein), the sequences have to be aligned. By aligning two or more biological sequences, a model that reveals not only the elements' matching and mismatching positions, but also the positions of elements' deletions (insertions) is generated. From an applicational viewpoint, the alignment is used for phylogenetic analysis, identification of functional motifs, profiling genetic diseases, prediction of proteins' tertiary structure upon similarities in proteins' primary structures, etc. As a computer process, each alignment requires time and memory. In general, the time and memory requirements for performing an alignment depend of the number and length of the sequences that are aligned. In some cases, when aligning many and long sequences, such as eukaryotic chromosomes or eukaryotic complete genomes, the time and memory complexity of an algorithm may be decisive factor whether the alignment of the sequences can be performed or not on a specific computer with limited processor speed and memory. Therefore, improvements of time and memory complexity are always welcomed, but not at the cost of generating partial solutions that do not completely depict the structural, functional or evolutionary relationship between the aligned sequences.

Depending of the number of sequences that are being aligned, there are algorithms for pairwise and multiple sequence alignment. Pairwise alignment algorithms align two sequences, whereas multiple alignment algorithms align more than two sequences. In terms of the scope of the generated solution, sequence alignment algorithms can be classified as local or global. Local alignment algorithms search for the most conserved (similar) DNA fragments between two or more sequences, whereas global alignment algorithms produce solutions that incorporate each element. In terms of the conceptual framework, if we look back in history, since

CONTACT Done Stojanov done.stojanov@ugd.edu.mk

the pioneering work of Needleman and Wunsch [1] in 1970, all algorithms for DNA pairwise alignment can be classified either as a dynamic programming based or they use some type of heuristic in order to obtain the solution (heuristic algorithms). Both groups of algorithms have their own pros and cons. Dynamic programming-based algorithms always produce a solution that maximizes the alignment score or minimizes the distance (dissimilarity) for specific metrics. This solution is referred to as an optimal one and it requires $O(n \times m)$ time and memory, where n and m are lengths of the sequences that are being aligned. The unfavourable quadratic time and memory complexity sometimes may limit the application of these algorithms, especially if long sequences are aligned on a standard PC configuration. In contrast to dynamic programming-based algorithms, heuristic algorithms are faster and they can be applied to long sequences, such as complete chromosomes or complete genomes. They gain on the speed of execution due to the significant reduction of the problem domain, by applying some type of filtering or prior indexing. As a result, sometimes partial or incomplete solution may be produced that does not fully represent the real structural, functional or evolutionary relationship between the sequences. A compromise between the opposite groups of algorithms is always welcomed.

The oldest, but commonly used DNA pairwise alignment algorithms, are Needleman–Wunsch [1] and Smith–Waterman.[2] They are dynamic-programming based implementations that maximize the score of an alignment for a specific metrics. Needleman–Wunsch [1] generates global alignment, whereas Smith–Waterman [2] generates local alignment in quadratic $O(n \times m)$ time and memory. Instead of maximizing the alignment's score, Sellers [3,4] minimizes the distance (dissimilarity) between two sequences. Ulam [5] gave the formal definition for distance (dissimilarity) between two sequences, as the minimal number of edit operations (element deletion, element substitution or element insertion) that have to be performed in order to transform one of the sequences into the other. Smith et al. [6] proved that no matter whether the alignment's score is maximized [1,2] or the distance (dissimilarity) between two sequences is minimized,[3,4] these two procedures are equivalent in terms of the structure of the generated solution under certain conditions. Rather than searching for one optimal solution, done in previous studies,[1–4] algorithms of Goad and Kanehisa [7] and Waterman and Eggert [8] generate a list of significant local alignments. For instance, Waterman and Eggert [8] identified a set of k best local alignments between two sequences. This is done by searching a dynamic programming matrix k times for k highest-scoring alignments. It is worth to mention that Fitch and Smith,[9] as well as Gotoh,[10,11] made a distinction between the penalty that is assigned for gap opening and the penalty that is assigned for extending an already opened gap.

The first tries for linearization of the memory complexity of dynamic programming algorithms were applications of divide and conquer approach. The idea is to find in each step an interception point (u, v) such as the alignment of the sequences: $a: a_1 \ldots a_n$ and $b: b_1 \ldots b_m$ is generated by joining the optimal alignments of the subsequences: $(a_1 \ldots a_u; b_1 \ldots b_v)$ and $(a_{u+1} \ldots a_n; b_{v+1} \ldots b_m)$. This is repeated until base is aligned to base or gap. Since the path of the optimal alignment cannot pass through the north-east and south-west quadrant in dynamic programming matrix, these quadrants can be rejected in each step of the procedure. This idea was theoretically discussed by Hirschberg [12] and it served as a basis for the memory linear implementations that were proposed afterwards by Mayers and Miller,[13] Huang et al. [14] and Huang and Miller.[15]

Apart from the efforts for memory linearization, there were also attempts to improve the time complexity. The first approach is known as alignment within specific diagonal band or diagonal alignment. The idea is to calculate only those cells in the dynamic programming matrix that are located near by the main diagonal, since the path of the optimal alignment converges to it. If the length of the diagonal band equals $k, k < m$ then the time complexity can be improved up to $O(k \times n)$. This approach was discussed by Sankoff and Kruskal,[16] whereas solutions of the problem were proposed by Ficket,[17] Ukkonen [18] and Chao et al.[19] However, drawback is its application that is limited only to very similar sequences.

Heuristic algorithms were introduced to meet the necessity for significant improvement of the time complexity of DNA alignment process. They can speed up the process up to hundred times, but there is no guarantee for optimality of the generated solution. The increased speed of execution makes these algorithms suitable for comparing query sequence to database, rather than comparing only two sequences.

For instance, FASTA, the DNA and protein sequence alignment software package,[20] looks for common hits between two sequences which are afterwards extended. Only 10 most significant extensions are included in the solution. Basic local alignment search tool (BLAST) [21] compiles an extended list of words that resemble the words extracted from one of the sequences. These words are searched in the other sequence in order to find the perfect hits. Each hit is extended in both directions until the score drops below certain threshold. Unlike BLAST, BLAST-like alignment tool (BLAT) [22] is

less sensitive approach that looks for perfect or near-perfect hits, due to what can be applied only for comparison of evolutionary closely related sequences. PatternHunter [23] improves the BLAST sensitivity, due to the fact that PatternHunter looks for common hits that include at least k common nucleotides. Alike BLAST, these hits are afterwards extended in both directions. Fast length adjustment of short reads (FLASH) [24] is also seed-based approach that looks for common words between two sequences. After calculating the differences between the starting positions of common hits, the longest list of equal differences determines the most similar fragments between the sequences. Yet Another Similarity Searcher (YASS), a pairwise sequence alignment software,[25] looks for groups of close hits. The maximum distance between the hits in the groups should not exceed d where d is calculated in context to the frequencies of base substitutions and base insertions (deletions).

The most efficient approach to identify common hits between two sequences is to employ suffix tree. This data structure uses maximum unique matches-mer (MUMmer) [26] and AVID, a global alignment method. [27] MUMmer looks for maximum unique matches (MUMs) upon which an alignment is derived. Maximum unique match is defined as a common hit with maximum length that is not part of any other hit. By rejecting hits, whose length is less than half of the length of the longest common hit, AVID looks for a subset of non-crossing and parallel hits, which are identified by traversing suffix tree that is constructed for a sequence which is obtained by joining together the sequences being aligned. Unlike MUMmer and AVID, LAGAN, tool for large-scale multiple alignment of genomic DNA,[28] detects ungapped local alignments between two sequences, where, except of nucleotide hits, mismatches are also allowed. Local ungapped alignments are identified by CHAOS algorithm, a fast database search tool that creates a list of local sequence similarities,[29] which can detect short and partial hits between two sequences. LAGAN selects subset of consecutive ungapped alignments with maximum alignment score, which are afterwards connected. Super pairwise alignment (SPA) [30] exploits methods based on probability and combinatorics to find the positions where gaps should be inserted. SPA calculates the percentage of local similarity within shifting window of fixed size. The shifting window includes fragments of both sequences for which the percentage of similarity is calculated as a rate between the number of matching elements and the length of the shifting window. Gaps are inserted if minimum shift of nucleotides drastically increases the percentage of local similarity.

In terms of computational aspects (time and memory complexity) and comprehensiveness of the generated solution, a compromise between dynamic programming-based algorithms and heuristic algorithms is presented. The proposed algorithm has several distinctions from the existing ones. First, the proposed algorithm does not reject any of the hits that can be included in the solution. By applying dynamic programming algorithms, some of the hits may be rejected due to the decrease of the overall alignment score for a specific metrics, in spite of the fact that these hits can be included in the alignment. The same also happens if heuristic algorithms are applied, due to the filtering in the preprocessing phase when usually shorter hits are rejected on what these algorithms gain on the speed of execution. By generating a solution that always includes the maximum possible numbers of base-to-base hits, the proposed algorithm is applicable for comprehensive structural, functional and evolutionary analysis and it can be applied for distant homology detection. Even for dissimilar sequences, the proposed algorithm is superior to dynamic programming and heuristic algorithms in terms of accurate and comprehensive detection of matching elements between two DNA samples. Second, rather than inserting gaps randomly, they are always inserted in base pairs XY with minimal frequency of occurrence, or in other words, we take into consideration the preservation of base pairs when determining the position where an indel is localized. Most importantly, we found that this model allows a more accurate detection of continuous indels in polyglutamine tracts of spliced mRNA of huntingtin gene than the dynamic programming. Third, unlike dynamic programming algorithms, which require $O(n \times m)$ base comparisons in order to identify the set of common hits, the proposed algorithm performs less than $n \times m$ comparisons for the same purpose. This can be interpreted as a gain in the speed of execution, but unlike heuristic algorithms, there is no loss of data. Furthermore, if similar sequences are aligned, the time complexity of the proposed algorithm approximates to $O(n \times k \times \log_2 m)$, where k is the number of common hits ($k < m$), m is the length of the shorter sequence and n is the length of the longer sequence. When dissimilar sequences are being compared, the time performance of our algorithm equals Smith−Waterman or $O(n \times m)$ time is required. Fourth, due to the memory efficient representation of each hit with data tuple that tracks the hit's starting and final positions, the memory requirement of the proposed algorithm is linear. In all tests that we performed, our algorithm required less memory than the length of the shorter sequence, regardless the identity of the sequences that were compared. This makes our algorithm one

of the most memory efficient approaches proposed so far.

Materials and methods

The time and memory optimized (TMO) algorithm runs in two phases. In the first phase, we perform a search for 'common' and 'consistent' hits of two DNA sequences, upon which an alignment in the second phase is constructed.

Phase 1: searching and representing common hits

A 'common' hit for two DNA sequences is a character or word found in both sequences. Each hit that does not form crossing diagonal with any other hit can be considered as 'consistent'. To introduce the concept of consistent and inconsistent common hits, the following samples are considered: a = ACACAATGGGGGCTCTACA and b = ACACTGTTTGGGGGACA. If the first occurrence of ACA in a(b) is paired with the last occurrence of ACA in b(a), two inconsistent hits are obtained, due to the crossing diagonal that they form (Figure 1(a)). Consistent hits are formed by pairing first-to-first and last-to-last occurrence of ACA in a and b (Figure 1(b)).

To optimize the memory requirement, a data tuple of four integers: $(p_{a,s}, p_{a,f}, p_{b,s}, p_{b,f})$ per identified consistent hit is tracked in the memory, where $p_{a,s}$ is the strating position of the hit in sequence a, $p_{a,f}$ is the final position of the hit in sequence a, $p_{b,s}$ is the strating position of the hit in sequence b and $p_{b,f}$ is the final position of the hit in sequence b. Under this approach, tuple $(1, 4, 1, 4)$ represents the first consistent hit ACAC in the samples a and b and tuples: $(7, 12, 9, 14)$ and $(17, 19, 15, 17)$ represent the second (TGGGGG) and the third (ACA) consistent hit, respectively (Figure 2).

The set of consistent hits: $\{$ACAC: $(1, 4, 1, 4)$, TGGGGG: $(7, 12, 9, 14)$, ACA: $(17, 19, 15, 17)\}$ is not the only one which can be derived for the samples that we consider. There are many other sets of consistent hits that mutually differ in the number of hits and in the number of matching nucleotides per hit, but the algorithm always tries to find the set of consistent hits which includes maximum possible number of matching elements, i.e. the set: $\{$ACAC: $(1, 4, 1, 4)$, TGGGGG: $(7, 12, 9, 14)$, ACA: $(17, 19, 15, 17)\}$ is reported, because there is no other set that can be defined for the samples a and b that includes more matching nucleotides than the one being detected.

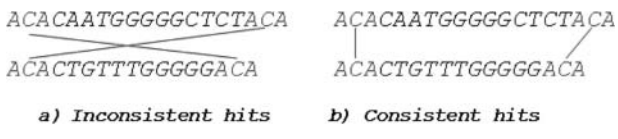

Figure 1. Examples of inconsistent (a) and consistent (b) common hits.

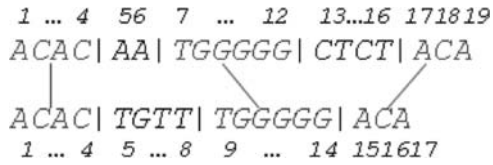

Figure 2. Set of common and consistent hits.

During the search for consistent hits, our algorithm exploits an idea which is based on partial comparison of mismatching DNA words, instead to comparing them entirely, or nucleotides at same positions in DNA words that mismatch are mutually compared, while the first pair of mismatching nucleotides is not detected. By employing this approach, only one base comparison is required to detect the mismatch of two DNA words in the best case, whereas in the worst case, all nucleotides have to be compared.

For instance, to detect the mismatch of the words: AGCT and GCTA, only the starting nucleotides have to be compared (A to G). The remaining nucleotides are not compared, since regardless if they match or not, the words AGCT and GCTA cannot perfectly match. The worst case scenario is when all nucleotides have to be compared, in order to detect the mismatch. That is the case when all nucleotides match, except the last ones. To detect the mismatch of the words: GCTA and GCTG, four comparisons have to be performed (the number of base comparisons in this case equals the words' length), since the last nucleotides (A and G) in these words constitute the first found pair of mismatching nucleotides. However, the number of base comparisons may be greater than one and less than the length of the words, depending of the position where the first pair of mismatching nucleotides is identified. For instance, when comparing the words TCTG and TACT, two base comparisons have to be performed in order to confirm the mismatch, since the first pair of mismatching nucleotides is found at position 2 (C to A).

Our algorithm looks for consistent hits in 'consistent domains'. 'Consistent domain' is defined as a pair of sequences or subsequences in a and b which are located out of identifed hits and they do not form crossing diagonal to any of the hits, Figure 3.

At the beginning, there is only one consistent domain which is made of the sequences being compared (Figure 3(a)). The search there results in one consistent hit, which derives two new consistent domains where two new hits may be identified (Figure 3(b)). These two

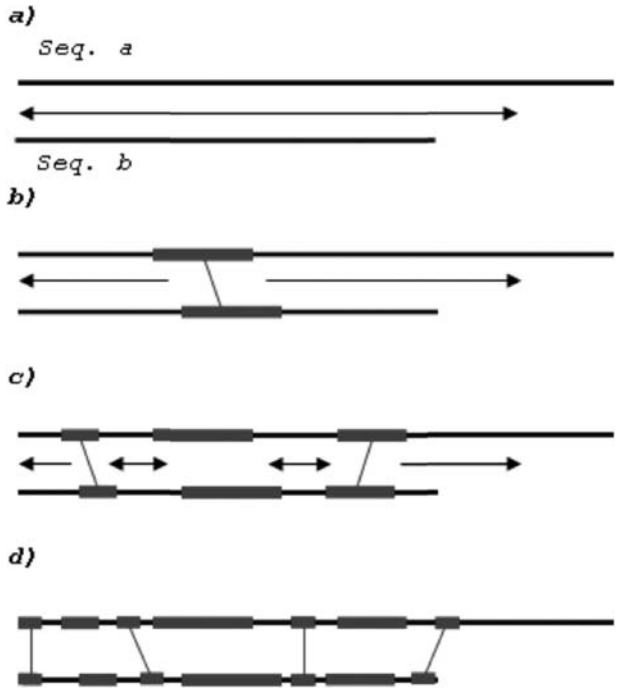

Figure 3. Searching for consistent hits: one consistent domain (a), one hit (two new consistent domains) (b), three hits (four new consistent domains) (c) and all nucleotides in consistent domains are compared (d).
Note: arrows represent consistent domains; thickened and connected segments represent identified hits in each step; sequences (Seq)

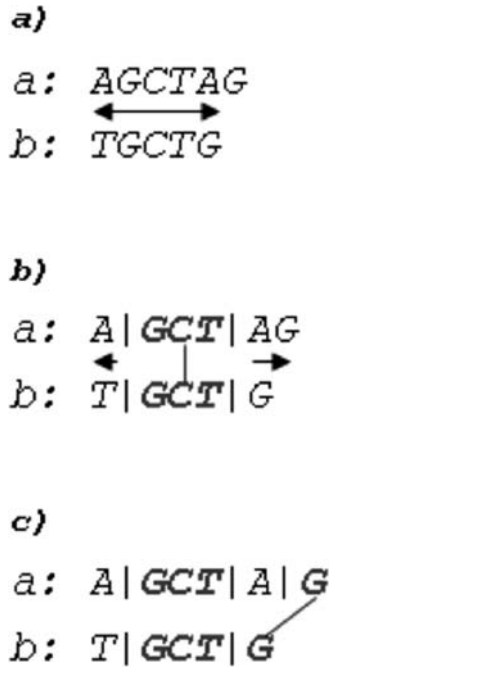

Figure 4. Identifying consistent hits for the samples AGCTAG and TGCTG: short samples (a), the first identified hit GCT derives two new consistent domains (b) and a single Guanine match is found in the consistent domain that follows GCT (**c**).

hits together with the first identified match derive four new consistent domains, where we can also search for consistent hits (Figure 3(c)) and so on until the length of at least one consistent fragment equals 0 or all nucleotides in the consistent domains are compared (Figure 3(d)).

In each consistent domain, the algorithm detects the 'longest' and the 'leftmost' common hit by performing partial comparisons on mismatching DNA words of the shorter sequence/subsequence to the longer sequence/subsequence until the hit is identified.

We will discuss this approach on the short samples a: AGCTAG and b: TGCTG (Figure 4(a)). At the beginning, there is only one consistent domain which is made of the samples a and b and this is the space where the algorithm tries to find the first consistent hit. Table 1 shows the order in which the algorithm compares mismatching DNA words and which nucleotides of them (highlighted) are mutually compared.

Words of 5, 4 and 3 nucleotides from both sequences are partially compared until the first hit GCT is identifed. The search requires totally 20 base comparisons (Table 1). The number of performed comparisons per pair of DNA words depends of the position of the first identified pair of mismatching nucleotides. For instance, one base comparison is required to detect the mismatch: (AGCTA, TGCTG) since the word's starting nucleotides mismatch. Two comparisons have to be performed to detect the mismatch: (TAG, TGC), because the first mismatch of nucleotides is found at position 2 and four base comparisons are required in order to detect the mismatch of the words GCTA and GCTG since all nucleotides, except the last ones, match (Table 1).

The first identified hit GCT derives two new consistent domains. The first consistent domain is made of the nucleotides that precede GCT (A in a and T in b), whereas the second is made of the nucleotides that follow GCT in a and b (AG in a and G in b) (Figure 4(b)). There is no common hit in the domain that precedes GCT, whereas a single guanine match is found in the consistent domain that follows GCT (Figure 4(c)). The algorithm requires totally 23 comparisons to derive the set of consistent hits for the samples a : AGCTAG and b: TGCTG, $\{\text{GCT} : (2,4,2,4),\ \text{G}(6,6,5,5)\}$ (Table 1).

Since tuples are appended into a set as hits are identified, before moving to the second phase, the set of tuples needs to be sorted according to hits' occurrence in the analysed sequences. This means that if we have obtained the set of tuples: $\{(7,12,9,14),\ (1,4,1,4),\ (17,19,15,17)\}$ for the samples a = ACACAATGGGG GCTCTACA and b = ACACTGTTTGGGGGACA, because TGGGGG : $(7,12,9,14)$ is identified at first, then ACAC: $(1,4,1,4)$ and finally, ACA : $(17,19,15,17)$, before

Table 1. Compared words and nucleotides, until consistent hits in the samples a : AGCTAG and b : TGCTG are identified.

Searching for hit in the consistent domain: a: AGCTAG and b: TGCTG					
Compared words	$a_1 \ldots a_5$:AGCTA	$a_2 \ldots a_6$:GCTAG			Comparisons
$b_1 \ldots b_5$:TGCTG	AGCTA	GCTAG			2
	TGCTG	TGCTG			
	$a_1 \ldots a_4$:AGCT	$a_2 \ldots a_5$:GCTA	$a_3 \ldots a_6$:CTAG		
$b_1 \ldots b_4$:TGCT	AGCT	GCTA	CTAG		3
	TGCT	TGCT	TGCT		
$b_2 \ldots b_5$:GCTG	AGCT	GCTA	CTAG		6
	GCTG	GCTG	GCTG		
	$a_1 \ldots a_3$:AGC	$a_2 \ldots a_4$:GCT	$a_3 \ldots a_5$:CTA	$a_4 \ldots a_6$:TAG	
$b_1 \ldots b_3$:TGC	AGC	GCT	CTA	TAG	5
	TGC	TGC	TGC	TGC	
$b_2 \ldots b_4$:GCT	AGC	GCT	-	-	4
	GCT	GCT (hit)			
Searching for hit in the consistent domain: a_1: A and b_1: T prior to GCT hit					
Compared words	a_1:A				
b_1:T	A				1
	T				
Searching for hit in the consistent domain: a_5a_6: AG and b_5: G after GCT hit					
Compared words	a_5:A	a_6:G			
b_5:G	A	G			2
	G	G(hit)			
Total number of performed base comparisons:					23

moving to the second phase, the algorithm will swap the first two tuples, because $(1,4,1,4)$ corresponds to hit ACAC that precedes TGGGGG, which is tracked by $(7,12,9,14)$. As a result, we get the set of tuples: $\{(1,4,1,4),\ (7,12,9,14),(17,19,15,17)\}$, which tracks the natural order of appearance of the consistent hits found in the samples a and b (ACAC, followed by TGGGGG and then ACA) (Figure 2).

We can use the previous samples to analyse the gain in terms of memory reduction, compared to Huang's algorithm [14] and the gain in saved nucleotide comparisons regarding Smith−Waterman.[2] For instance, if Huang is applied to a = ACACAATGGGGGCTCTACA and b = ACACTGTTTGGGGGACA, the memory requirement would be proportional to the sum of the lengths of the sequences being aligned, i.e. $n+m=19+17=36$ integers would have to be stored in the memory, in contrast to number of hits $\times 4 = 3\times 4 = 12$ integers that store the proposed algorithm for tracking hits' positions. To derive a set of consistent hits for the samples a : AGCTAG and b : TGCTG, Smith−Waterman requires lengthOf(a)×lengthOf $(b)=6\times5=30$ comparisons, whereas our algorithm, for the same purpose, requires 23 comparisons, without losing data for any consistent hit that may happen in heuristic algorithms for local DNA alignment, such as: FASTA,[20] BLAST,[21] BLAT,[22] PatternHunter,[23] FLASH [24] and YASS.[25]

Phase 2: generating local pairwise alignment

At the beginning of the second phase, the number of indels between successive hits is precisely calculated. Assuming that a is the longer sequence and b is the shorter sequence, then for each two successive hits, k'th and $(k+1)$'th the number of gaps n.g which have to be inserted between them is calculated according to Equation (1), where $(p_{a,s\,|\,k}, p_{a,f\,|\,k}, p_{b,s\,|\,k}, p_{b,f\,|\,k})$ and $(p_{a,s\,|\,k+1}, p_{a,f\,|\,k+1}, p_{b,s\,|\,k+1}, p_{b,f\,|\,k+1})$ are tuples that track their positions in a and b, respectively. The sign of n.g (>, < or 0) determines whether gaps should be inserted or not and if yes, in which mismatching fragment they have to be inserted, whether in sequence a or in sequence b.

$$\text{n.g} = p_{a,s\,|\,k+1} - p_{a,f\,|\,k} - (p_{b,s\,|\,k+1} - p_{b,f\,|\,k}) \qquad (1)$$

If n.g > 0 then n.g gaps have to be inserted in b. If n.g < 0 then |n.g| gaps have to be inserted in a. If n.g = 0 then no indel is found between successive hits.

If we apply Equation (1) to the set of consistent hits $\{\text{ACAC}: (1,4,1,4), \text{TGGGGG}: (7,12,9,14), \text{ACA}: (17,19,15,17)\}$, which was derived for the samples a = ACACAATGGGGGCTCTACA and b = ACACTGTTTGGGGGACA, as a result, we obtain that two gaps have to be inserted between ACAC and TGGGGG in a and four gaps have to inserted between TGGGGG and ACA in b, i.e. $\text{n.g} = p_{a,s\,|\,k+1} - p_{a,f\,|\,k} - (p_{b,s\,|\,k+1} - p_{b,f\,|\,k}) = 7-4-(9-4) = -2$ for $\text{ACAC}: (1,4,1,4)$ and $\text{TGGGGG}: (7,12,9,14)$, whereas $\text{n.g} = p_{a,s\,|\,k+1} - p_{a,f\,|\,k} - (p_{b,s\,|\,k+1} - p_{b,f\,|\,k}) = 17-12-(15-14) = 4$ for $\text{TGGGGG}: (7,12,9,14)$ and $\text{ACA}: (17,19,15,17)$.

Indels/inserted gaps shift consistent hits towards their alignment. This means that $\text{TGGGGG}: (7,12,9,14)$ in sample a needs to be shifted for two positions (Figure 5(a))

Figure 5. Indels $(-)$ or DNA shifts enable perfect alignment of matching fragments: TGGGGG needs to be shifted for two positions in sample a (a), ACA needs to be shifted for four positions in sample b (**b**).

and ACA: $(17, 19, 15, 17)$ in sample b needs to be shifted for four positions (Figure 5(b)), if we want these hits to match perfectly within an alignment.

Before localizing indels, two matrices $[A]_{4\times4}$ and $[B]_{4\times4}$ that track the numbers of occurrence of all nucleotide pairs XY, X, Y $\in \{A, C, T, G\}$ in a and b are computed. Having marked each row and each column in A and B with one of the letters: A, C, T, G, which correspond to the four DNA nucleotides, $A[X, Y](B[X, Y])$ represents the number of occurrence of nucleotide pair XY in $a(b)$. These matrices for the samples: a = ACACAATGGGGGCTCTACA and b = ACACTGTTTGGGGGACA are shown in Figure 6. For instance: $A[C, A] = 3$ and $B\ [C, A] = 2$, because CA occurs three times in a and two times in b (Figure 6). Some base pairs may not occur at all, if short sequences are analysed. Sample a conatins one AA and one AT nucleotide pair, but they do not occur at all in the other sample $(B[A, A] = 0, B[A, T] = 0)$ (Figure 6).

The algorithm pinpoints indels in mismatching fragments which are obtained by appending the last and the first nucleotide from surrounding hits. Extending is required, because except inside mismatching fragments, gaps can also be inserted between hits and mismatching DNA. For instance, two gaps except inside AA mismatch (Figure 7(a)) can be also inserted between the first hit ACAC and AA in sample a (Figure 7(b)), as well as between AA and the second hit TGGGGG (Figure 7(c)).

The CAAT-gapped modifications shown in Figure 7 are based on continuous insertion of gaps and they are

Figure 7. Indels in extended mismatching fragments (marked with rectangles). Two gaps inserted: inside AA mismatch (a), between the first hit ACAC and AA (b) and between AA and the second hit TGGGGG (c).

three of six possible two indel transformations of the extension, since gapped transformations C−A−AT, C−AA−T and CA−A−T of CAAT, which are based on interrupted localization of indels, are also possible.

The whole process of identifying indels step by step, according to the proposed methodology, is shown in Figure 8. Our algorithm reports CA−A−T modification, because AA and AT are considered for mutated triplets whose mid-elements were deleted ($A - A \rightarrow AA$, $A - T \rightarrow AT$), due to the fact that CA nucleotide pair tends to be more preserved than AA/AT in a (CA occurs three times, whereas AA and AT occur only once). Therefore, rather than modifying a frequently occurring nucleotide pair (CA), our algorithm will choose to pinpoint indels in rarely occurring nucleotide pairs (AA/AT) (Figure 8(b) and 8(c)).

In general, for each extension, a list of all nucleotide pairs found there and data regarding their occurrence in $a(b)$, is compiled. The number of occurrences of each base pair is read from matrixA (matrixB), depending on which sequence the extension belongs to. Data from the list is used to identify the rarest occurring nucleotide pair in each step, where, in fact, an indel is identified.

[A]	A	C	T	G
A	1	3	1	0
C	3	0	2	0
T	1	1	0	1
G	0	1	0	4

[B]	A	C	T	G
A	0	3	0	0
C	2	0	1	0
T	0	0	2	2
G	1	0	1	4

Figure 6. Occurence of nucleotide pairs XY, X, Y $\in \{A, C, T, G\}$ in a = ACACAATGGGGGCTCTACA and b = ACACTGTTTGGGGGACA.

The rarest occurring nucleotide pair XY in a(b) of all nucleotide pairs in the list is the one where the first gap is inserted: $XY \rightarrow X-Y$. This nucleotide pair, in which an indel is identified, has to be removed from the list before repeating the search, if another gap has to be inserted. The same is repeated until all gaps are inserted (their number is calculated according Equation(1)). If insertion has to be proceeded, but the list is empty (in all nucleotide pairs in the extension gaps were inserted), the proposed algorithm chooses to insert all remaining gaps next to the first nucleotide in the extension.

For the samples that we consider, CAAT is an extension of the mismatching AA fragment in sample a which is obtained by appending the last nucleotide C from the first hit (ACAC) and the first nucleotide T of the following hit (TGGGGG). This extension contains three nucleotide pairs: CA, AA and AT (Figure 8(a)). The numbers of occurrence of these nucleotide pairs are read from matrix A (Figure 6(a)) and they are stored in list $L=\{A\ (CA)=3, A\ (AA)=1, A\ (AT)=1\}$.

Searching the list for the rarest occurring nucleotide pair in a of all nucleotide pairs in the list, the algorithm pinpoints the location of the first indel. The first found nucleotide pair is always reported and that is the nucleotide pair where the gap is inserted.

The corresponding nucleotide pair in this case is AA in the extension CAAT, which is transformed in (CA−AT) (Figure 8(b)). According to the previous discussion, the nucleotide pair AA, where the gap is inserted, $A\ (AA)=1$ is removed from the list $L=\{A(CA)=3, A\ (AT)=1\}$. Once again the algorithm looks for the rarest occurring nucleotide pair in $L=\{A(CA)=3, A(AT)=1\}$, but now to identify the position where the second gap has to be inserted. The nucleotide pair AT is reported, since AT appears once and CA three times. The second gap insertion transforms (CA−AT) in (CA−A−T) (Figure 8(c)). Due to the absence of mismatching DNA content between TGGGGG and ACA in b, all four gaps are inserted next to the last nucleotide G of the second hit (TGGGGG) (Figure 8(d)).

The algorithm pinpoints indels in 'the least preserved nucleotide pairs', because it assumes that they 'originate from codons whose mid nucleotides were deleted' when working with coding DNA. This property does not restrict the application of our algorithm only to coding DNA. It is a general approach that can also be applied to junk DNA (tandem repeats and interspersed repeats), where instead of codon mutations, mutations in non-coding triplets may be considered for accurate prediction of indels within satellite DNA or precise localization of Alu (LINE) repetitions. By applying our algorithm to eukaryotic chromosomes, we expect to retain Alu repetitions intact, due to its high occurrence rate, which can facilitate the identification of a newly evolved gene. On the other hand, Smith−Waterman will pinpoint indels in one or more Alu repetitions, if that results in a better score alignment.

Figure 8(d) shows the alignment of the samples a and b according to the proposed model. As it can be noticed, all matching nucleotides are included in the solution. Shortened alignment, which partially depicts the structural relationship between the samples, is obtained if Smith−Waterman is applied, based on metrics which penalizes mismatch alignments and gap insertions with −1 and awards +1 for each nucleotide match (Figure 9). For this metrics, consistent hits ACAC and ACA, are not detected (included in the alignment) (Figure 9). The overall alignment score would not be increased if the first consistent hit ACAC is included in the solution. On the other hand, the overall alignment score would have been decreased for 1, if the third consistent hit ACA is included, since four gaps have to be inserted to align three matching nucleotides: A, C and A.

Obviously, when applying Smith−Waterman, the output depends on the selected metrics. As a consequence, some local similarities 'remain unidentified' ('excluded of

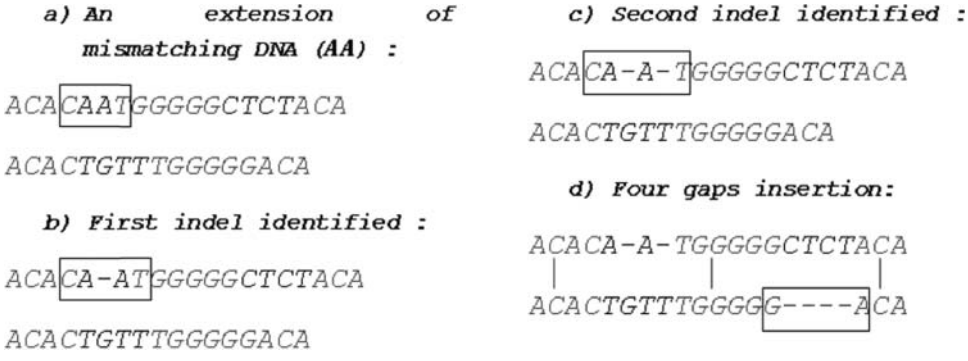

Figure 8. Steps for generating a solution: (a) CAAT is an extension of the mismatching AA fragment and contains three nucleotide pairs: CA, AA and AT; (b) gapped modification of AA; (c) gapped modification of AT; and (d) four gaps are inserted following the last nucleotide G in TGGGGG.

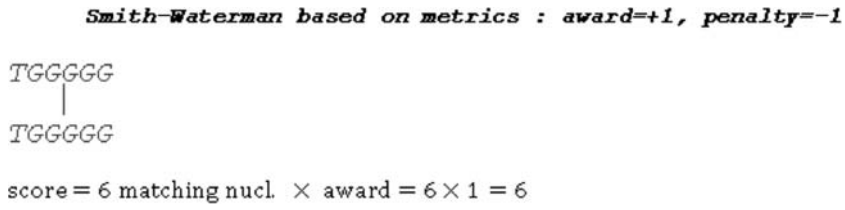

Figure 9. Smith−Waterman solution for specific metrics.
Note: nucleotide (nucl).

the alignment'). For instance, although three matching fragments (ACAC, TGGGGG, ACA) can be included in the alignment of the samples a and b, Smith−Waterman detects only one (TGGGGG) matching fragment.

Partial solutions derived from Smith−Waterman may imply incorrect identification, characterization and classification of DNA data, such as errors when predicting disease predisposition of a specific gene and incorrect taxonomic classification. Unlike Smith−Waterman, the proposed algorithm maximizes the number of matching nucleotides per alignment, regardless any specific metrics and rather than inserting gaps at arbitrary positions, they are inserted at positions in the least preserved nucleotide pairs.

Due to these properties, a more accurate model for the relationship of trinucleotide repeat disorders associated with spliced mRNA can be derived in comparison to Smith−Waterman. It is known that trinucleotide repeat disorders, such as Huntington's disease,[31,32] Spinocerebellar ataxia [33] and Dentatorubropallidoluysian atrophy [34] are genetic disorders which are associated with excessive expansion of trinucleotide repeats in certain genes' fragments. Usually, the number of uninterrupted trinucleotide repetitions determines whether an individual will be affected or not. For instance, Huntington's disease, which is associated with excessive number of uninterrupted CAG repetitions in Huntingtin gene, causes a progressive decline of brain cells. An individual is affected if the number of uninterrupted repetitions exceeds 35. We found that indels of CAG repetitions in spliced mRNA of huntingtin polyglutamine (polyQ) tract are more specifically identified by the proposed algorithm than Smith−Waterman.

Figure 10 shows the alignments of *Homo sapiens* isolate hunt1 huntingtin partial coding sequence (cds) (ENA ID: EU797016.1) and *Homo sapiens* isolate hunt3 huntingtin partial cds (ENA ID: EU797018.1) near polyQ coding strands by applying Smith−Waterman (Figure 10(a)) and the proposed algorithm (Figure 10 (b)). Our algorithm detects double deletion of glutamine (CAG) in polyQ tract of hunt3 huntingtin spliced mRNA regarding polyQ tract of hunt1 huntingtin spliced mRNA (Figure 10(b)). On the other hand, glutamine's double deletion is not confirmed by Smith−Waterman (Figure 10(a)), since non-glutamine nucleotide (cytosine) in hunt3 huntingtin partial cds is aligned to glutamine nucleotide (cytosine) in polyQ tract of hunt1 huntingtin partial cds.

Regardless if non-glutamine to glutamine or non-glutamine to non-glutamine cytosine is aligned, the overall Smith−Waterman alignment score is not affected and that is the only parameter, which Smith−Waterman considers. The proposed algorithm does not separate cytosine in hunt3 huntingtin partial cds from … CCTCAAGTCCTT matching fragment, because its length is increased for one, if non-glutamine cytosines are

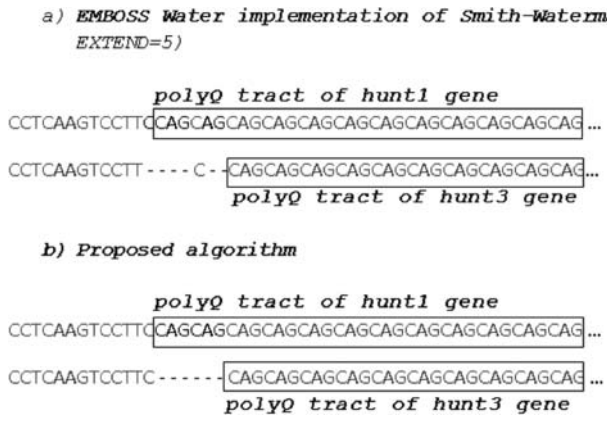

Figure 10. Detection of glutamine (CAG) indels in spliced mRNA of *Homo sapiens* huntingtin gene by applying Smith−Waterman (a) and the proposed algorithm (b).
Note: European Molecular Biology Open Software Suite (EMBOSS); polyglutamine (polyQ).

aligned (Figure 10(b)). Moreover, the nucleotide pair TC is considered to be unsplittable due to its frequent occurrence in hunt3 huntingtin partial cds (Figure 10(b)).

By knowing the number of uninterrupted glutamine repetitions in hunt1 huntingtin spliced mRNA, it is easy to find out whether the carrier of spliced mRNA, associated with hunt3 huntingtin, is affected by the disease or not.

In addition, we summarize the instructions in which our algorithm runs. Following these instructions, we developed a desktop application in C# which was tested on Acer Aspire 5570Z computer with Genuine Intel CPU 2080 at 1.73 GHz and 2 GB RAM memory.

Input: DNA sequence **a,** DNA sequence **b**
Output: Local alignment **A**
10. Partly comparing mismatching DNA words, **IDENTIFY ALL CONSISTENT HITS**.
20. **REPRESENT** each hit with data tuple: $(p_{a,s}, p_{a,f}, p_{b,s}, p_{b,f})$ in the memory.
30. **SORT** the set of tuples according to their appearance in a and b.
40. **CALCULATE** the number of indels between successive hits by applying Equation (1).
50. **TRACK** the number of occurrences of each nucleotide pair XY in a(b) in matrix $[A]_{4\times4}$ ($[B]_{4\times4}$).
60. **EXTEND** each mismatching fragment (where at least one indel is identified) with the last nucleotide from the preceding hit and the first nucleotide from the following hit.
70. **FOR EACH** nucleotide pair in extension, read the number of occurrence from matrix A(B) and store it in list L.
80. **UNTIL** all gaps are inserted.
90. **SEARCH** L for the rarest occuring nucleotide pair XY and **TRANSFORM** XY in $\mathbf{X-Y}$ in the extension.
100. **PRINT** alignment **A**.

Figure 11 is a screen capture of the application output, when we compared partial cds of homosapiens hunt1 huntingtin (ENA ID: EU797016.1) with hunt3 huntingtin (ENA ID: EU797018.1) near polyQ tracts, where indels / gaps are marked with '**N**'.

In total, there were nine tests that were performed on very similar huntingtin partial cds (ENA IDs: EU7970**16**−**24**.1), which were retrieved from the European Nucleotide Archive,[35] as well as a test, which was performed on dissimilar coding DNA of human tetratricopeptide repeat protein 25 (UniProt ID: Q96NG3) to tetratricopeptide repeat protein 23 (UniProt ID: Q5W5×9). The same samples were also aligned using European Molecular Biology Open Software Suite (EMBOSS) Water implementation of Smith−Waterman.[36]

Both algorithms were compared in terms of computational performance and hit detection rate. Additionally, we analyzed the number of detected glutamine deletions in polyglutamine tracts and the number of detected matching residues in tetratricopeptide repeats (TPR), which are considered to be similar, but different. TPR is usually made of 34 amino acids and in spite of the fact that $W_4 - L_7 - G_8 - Y_{11} - A_{20} - F_{24} - A_{27} - P_{32}$ consensus has been derived, all residues, except alanine at positions 20 and 27 (A_{20}, A_{27}), are likely to change.

Results and discussion

We got better computational results when we analysed partial cds of huntingtin gene than when we analysed the coding DNA of rather than similar tetratricopeptide repeat proteins. Even for tetratricopeptide repeat proteins, for which the percentage of identity is approximately 24%, our algorithm requires less memory than Huang, but the time complexity seems to approximate Smith−Waterman, i.e. in both cases, six seconds were required for deriving a solution. In all cases, regardless if similar or dissimilar sequences were analysed, our algorithm detected more consistent hits than Smith−Waterman that has resulted in better detection of matching residues in tetratricopeptide repeats. Moreover, our algorithm pinpointed indels in polyglutamine tracts of partial cds of huntingtin gene more accurately than Smith−Waterman's due to the fact that it considers the relative frequency of mismatching DNA, when it pinpoints an indel.

Figure 11. Output of the application near polyQ tracts of *Homo sapiens* huntingtin hunt1 to *Homo sapiens* huntingtin hunt3 partial cds alignment.

Table 2. Detected glutamine deletions in polyQ tract by applying the proposed algorithm.

Aligned sequences	Number of detected glutamine deletions in polyQ tract which were not identified by Smith−Waterman
(hunt3; hunt6) partial cds	3
(hunt6; hunt9) partial cds	3
(hunt1; hunt3) partial cds	2
(hunt2; hunt3) partial cds	2
(hunt4; hunt5) partial cds	2
(hunt2; hunt7) partial cds	2
(hunt7; hunt6) partial cds	1
(hunt6; hunt8) partial cds	1
(hunt1; hunt6) partial cds	1
(hunt9; hunt8) partial cds	0

Note: polyglutamine (polyQ); huntingtin (hunt); coding sequence (cds).

We chose to compare the performance of our algorithm to Smith−Waterman and Huang, because Smith−Waterman is considered to be the most accurate method for local pairwise alignment, whereas Huang has the same accuracy of Smith−Waterman, but instead of $O(n \times m)$ memory, it requires linear $O(n+m)$ space for deriving a solution. There are computationally better algorithms than Smith-Waterman/Huang, such as: FLASH, YASS and PatternHunter, but in most of the cases they generate partial (incomplete) solutions.

Glutamine deletions in polyQ tracts were analysed by applying the proposed algorithm and Smith−Waterman based on gap open $= +5$ and gap extend $= +5$ metrics. We found that a total of 17 missing glutamines in polyQ tracts were more specifically identified by the proposed algorithm compared to Smith−Waterman (Table 2). The number of precisely identified deletions in polyQ tracts applying the proposed algorithm ranges between 0 and 3 per alignment Table 2. Due to the key algorithmic feature to preserve highly conserved CA and AG nucleotide pairs in uninterrupted CAG repetitions in polyQ tracts, in 9 out of 10 cases our algorithm detects at least one more glutamine deletion more specifically than Smith−Waterman. Equal algorithmic outputs in the same context are obtained only once (Table 2).

In the cases of spliced hungtingtin mRNA alignments, our algorithm runs faster than Smith−Waterman, due to the fewer numbers of comparisons that were performed when compared with Smith−Waterman. The number of performed comparisons and the overall time required per alignment, when applying our algorithm and Smith−Waterman, are given in Table 3. Less nucleotide comparisons are performed, because rather than comparing mismatching DNA entirely, our algorithm escapes performing redundant comparisons, starting from the first found pair of mismatching nucleotides. According to the data in Table 3, the speed up of the proposed algorithm over Smith−Waterman depends on the number of matching fragments (hits). The speed up is greater, if fewer, but longer hits are detected (Table 3). In the best case, our algorithm runs 23 to 27 times faster than Smith−Waterman. These numbers correspond to alignments which include only two matching fragments. In the worst case, the seed up is approximately five times faster and that is the case when 8(9) hits are detected (Table 3). In spite of the fact that most of the time is spent for comparing nucleotides, minor part of the total time is spent for computing the frequencies of mismatching DNA and tracking/sorting the set of consistent hits. For hunt 4 (EU797019.1) and hunt 5 (EU797020.1) partial cds, it took only 0.1%, while 99.9% of the total time was required for comparing nucleotides.

On the other hand, six seconds were required for computing the alignment of human tetratricopeptide repeat protein 25 (UniProt ID: Q96NG3) to human tetratricopeptide repeat protein 23 (UniProt ID: Q5W5×9), when applying our algorithm and Smith−Waterman. In this case, there was no speed up over Smith−Waterman, because instead of few, but long common hits, our algorithm detected 162 short identities, which were not longer than four residues. The presence of many, but short common hits, as in this case, has a negative impact on

Table 3. Time performance analysis.

		Number of base comparisons (Total time)			
Aligned sequences	Hits	Smith−Waterman	Proposed algorithm	$f = n \times \text{hits} \times \log_2 m$	Speed up
(hunt4; hunt5) partial cds	2	234,247 (0.5198 s)	8,678 (0.01925 s)	8,678	27
(hunt2; hunt3) partial cds	2	162,400 (0.3604 s)	7,019 (0.0156 s)	7,019	23
(hunt7; hunt6) partial cds	2	161,200 (0.3578 s)	6,967 (0.0155 s)	6,967	23
(hunt1; hunt3) partial cds	3	166,800 (0.3702 s)	10,848 (0.0241 s)	10,813	15
(hunt6; hunt9) partial cds	3	156,400 (0.3471 s)	10,363 (0.0230 s)	10,333	15
(hunt2; hunt7) partial cds	4	163,618 (0.3631 s)	14,130 (0.0313 s)	14,055	12
(hunt3; hunt6) partial cds	4	160,000 (0.3551 s)	13,895 (0.0308 s)	13,830	11
(hunt1; hunt6) partial cds	4	166,800 (0.3702 s)	14,508 (0.0322 s)	14,418	11
(hunt9; hunt8) partial cds	8	150,535 (0.3341 s)	27,865 (0.0630 s)	26,865	5
(hunt6; hunt8) partial cds	9	154,000 (0.3418 s)	32,169 (0.0728 s)	30,919	5

Note: huntingtin (hunt); coding sequence (cds).

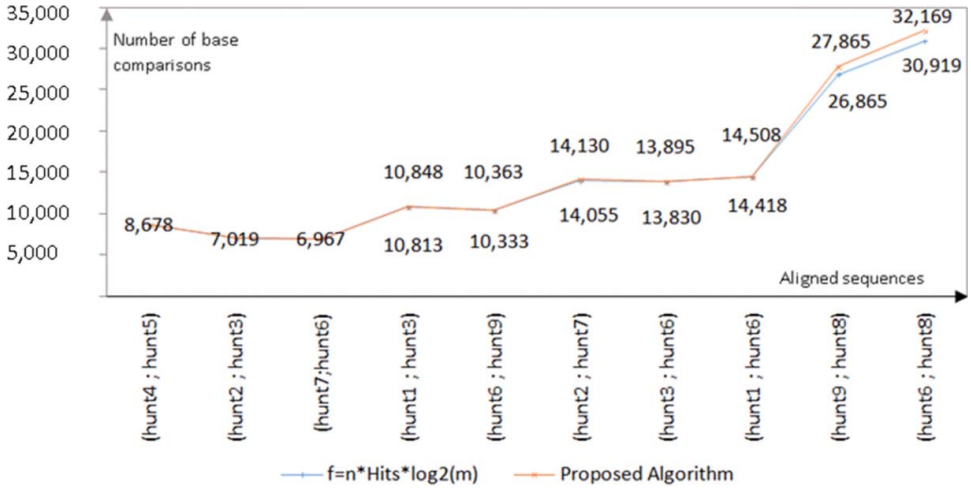

Figure 12. Best fitting function for the number of nucleotide comparisons.
Note: huntingtin (hunt).

the overall time performance, mainly due to the sizeable number of nucleotide comparisons that had to be performed. In this case, we estimated that 0.8% of the total time was spent for computing the frequencies of mismatching DNA and tracking/sorting the set of consistent hits that was likely to happen due to the high portion of mismatching DNA and the increased length of the consistent hits set.

We got better results than Smith−Waterman for the pairwise alignments of partial cds of huntingtin gene, because there were only few, but long common hits (2−9). This is specific only for very similar DNA data (the percentage of identity for partial cds of huntingtin gene is more than 90%), for which the obtained results confirm that our algorithm runs several times faster than Smith−Waterman. We found that $(n,\ m, k) = n \times k \times \log_2 m$ is the best fitting function for the time samples of spliced huntingtin mRNA alignments and, therefore, the time complexity of our algorithm when it works with very similar DNA is $O(n \times k \times \log_2 m)$, where n is the length of the longer sequence, m is the length of the shorter sequence and k is the number of hits (matching fragments) in alignment (Figure 12).

On the other hand, if sequences with poor identity are analysed, such as the cds of human tetratricopeptide repeat protein 23 to human tetratricopeptide repeat protein 25, for which the percentage of identity is 24%, our algorithm is not worse than Smith−Waterman, but there is no time improvement ($O(n \times m)$ time is required).

The results of the memory analysis for partial huntingtin cds are shown in Table 4. Space requirements of the proposed algorithm and Huangs' memory linear algorithm are compared. The memory requirement of the proposed algorithm was calculated according to Equation (2) and it depends on the Number of hits and the size of the list L $|L|$ which tracks the numbers of occurrence of all nucleotide pairs in mismatching extensions. The constant 32 in Equation (2) represents the

Table 4. Memory analysis.

		Memory (int)			
			Proposed algorithm		
Aligned sequences	Hits	Huang	Equation (1)	Equation (3)	Memory reduction-ratio applying Equation (1)
(hunt4; hunt5) partial cds	2	968	47	43	20.6
(hunt7; hunt6) partial cds	2	803	44	40	18.3
(hunt2; hunt3) partial cds	2	806	47	43	17.1
(hunt6; hunt9) partial cds	3	791	56	50	14.1
(hunt1; hunt3) partial cds	3	817	64	58	12.8
(hunt2; hunt7) partial cds	4	809	67	59	12.1
(hunt3; hunt6) partial cds	4	800	70	62	11.4
(hunt1; hunt6) partial cds	4	817	75	67	10.9
(hunt9; hunt8) partial cds	8	776	80	64	9.7
(hunt6; hunt8) partial cds	9	785	91	73	8.6

Note: integer (int); huntingtin (hunt); coding sequence (cds).

cost for storing two matrices: $[A]_{4\times4}$ and $[B]_{4\times4}$ in the main memory.

$$\text{Memory requirement} = 4\times\text{number of hits} + |L| + 32 \tag{2}$$

According to the results in Table 4, our algorithm requires between 8 and 20 times less memory than Huang. The same relationship, in terms of the number of hits, can be derived again, i.e. fewer hits are detected and the memory saving is greater (Table 4). The alignment which includes nine hits is the worst case of memory improvement, whereas the best one corresponds to the alignment of only two hits Table 4**.**

The proposed TMO algorithm required 2,592 B (648 integers) for tracking 162 residual common hits of human tetratricopeptide repeat protein 23 to human tetratricopeptide repeat protein 25. There were 1 four-residue hit, 3 three-residue hits, 25 two-residue hits and 133 one-residue hits detected. Regardless of the common hit length, our algorithm requires 16 B (four integers) per match representation, which means that the cost for tracking four-residues common hit equals the cost for tracking one-residue common hit or in both cases, 16 B (four integers) are required for tracking the hits' locations. For the same purpose, Huang required five times more memory, or 13,428 B. This confirms the space efficiency of the proposed methodology, even when sequences with poor percentage of identity are analysed.

Further improvement of the memory complexity for 25% for tracking data tuples is possible, if instead of four integers per data tuple, we use data tuples of three integers, since each common hit can be tracked with only three integers $(p_{a,s}, p_{b,s}, l)$, where $p_{a,s}$ is the starting position of the common hit in sequence a, $p_{b,s}$ is the starting position of the common hit in the sequence b and l is the length of the common hit, which is the same in both sequences. This is the maximum possible cut per data tuple without losing any tracking information, but compared to the first representation, which is based on four integers $(p_{a,s}, p_{a,f}, p_{b,s}, p_{b,f})$, this results in an additional memory saving of 25% for tracking data tuples.

If we exclude the length of the hits l from the data tuples, the number of indels between successive hits can also be precisely calculated, but this will result in memory saving of 50% for tracking tuples, compared to the first representation. This requires rearrangement of Equation (1), such as the final positions of the k'th hit in $a(b)$, $p_{a,f\,|\,k}(p_{b,f\,|\,k})$ have to be written as $p_{a,f\,|\,k} = p_{a,s\,|\,k} + l_k - 1$ $(p_{b,f\,|\,k} = p_{b,s\,|\,k} + l_k - 1)$, where $p_{a,s\,|\,k}$ $(p_{b,s\,|\,k})$ is the starting position of the k'th hit in $a(b)$ and l_k is the length of the hit. With a substitution of $p_{a,f\,|\,k}(p_{b,f\,|\,k})$ with $p_{a,s\,|\,k} + l_k - 1$ $(p_{b,s\,|\,k} + l_k - 1)$ in Equation (1), we get Equation (3), or the starting positions of hit k and hit k+1 in a and b: $p_{a,s\,|\,k}$, $p_{b,s\,|\,k}$ and $p_{a,s\,|\,k+1}$, $p_{b,s\,|\,k+1}$ are enough to calculate the number of indels.

$$\begin{aligned} \text{n.g} &= p_{a,s\,|\,k+1} - p_{a,f\,|\,k} - \left(p_{b,s\,|\,k+1} - p_{b,f\,|\,k}\right) \\ &= p_{a,s\,|\,k+1} - (p_{a,s\,|\,k} + l_k - 1) - (p_{b,s\,|\,k+1} - (p_{b,s\,|\,k} + l_k - 1)) \\ &= p_{a,s\,|\,k+1} - p_{a,s\,|\,k} - p_{b,s\,|\,k+1} + p_{b,s\,|\,k} \end{aligned} \tag{3}$$

By using Equation (3), instead of Equation (1), we reduced the overall memory requirement with up to 20% for huntingtin spliced mRNA (Table 4**)** and with 35% in the case of human tetratricopeptide repeat protein 25 to tetratricopeptide repeat protein 23 alignment. We got better memory reduction ratio for human tetratricopeptide repeat proteins, because there were more hits that were identified (162) than in the cases of huntingtin spliced mRNA alignments (2−9). The overall memory complexity was not reduced for 50%, because according to Equation (2), the algorithm also needs memory for tracking the mismatching DNA (|L|) and 32 integers for two matrices $[A]_{4\times4}$ and $[B]_{4\times4}$.

An equal number of matching nucleotides in spliced mRNA samples of huntingtin gene is identified when applying the proposed algorithm and EMBOSS Water implementation of Smith−Waterman based on gap open $= +5$ and gap extend $= +5$ metrics. But when we run tests based on increased gap penalty (gap open $= +50$, gap extend $= +10$), some of the matching nucleotides which were detected by the proposed algorithm were not detected by applying Smith−Waterman (Table 5). Regardless of any specific alignment metrics, in 9 out of 10 cases, our algorithm detected more matching nucleotides in partial cds of huntingtin gene than Smith−Waterman (Table 5). Their number ranges between 8 and 19 and they are excluded by Smith−Waterman, due to the decrease the overall alignment score.

When we compared human tetratricopeptide repeat protein 23 to human tetratricopeptide repeat protein 25, 59 matching amino acids were found by Smith−Waterman, whereas our algorithm reported 192. The analysis of the number of matching residues in tetratricopeptide repeats: TPR1, TPR2, TPR3 and TPR4 of human tetratricopeptide repeat protein 23 to human tetratricopeptide repeat protein 25 shows that our algorithm detected 46 matching amino acids more in tetratricopeptide repeats than Smith−Waterman (Table 6).

Moreover, Smith−Waterman completely excluded TPR4 repeat from the solution, due to the fact that it could not detect even a single matching residue,

Table 5. Number of matching nucleotides per alignment.

Aligned sequences	Number of matching nucleotides: EMBOS Water (Smith−Waterman gap open = 50, gap extend = 10)	Number of matching nucleotides: Proposed algorithm	Additionally identified matching nucleotides
(hunt6; hunt8) partial cds	366	385	19
(hunt1; hunt6) partial cds	378	396	18
(hunt1; hunt3) partial cds	385	400	15
(hunt2; hunt3) partial cds	385	400	15
(hunt9; hunt8) partial cds	367	382	15
(hunt4; hunt5) partial cds	466	481	15
(hunt2; hunt7) partial cds	383	396	13
(hunt3; hunt6) partial cds	382	390	8
(hunt6; hunt9) partial cds	382	390	8
(hunt7; hunt6) partial cds	400	400	0

Note: European Molecular Biology Open Software Suite (EMBOSS); huntingtin (hunt); coding sequence (cds).

whereas our algorithm detected 14 matching amino acids there (Table 6). Our algorithm detected between 11 and 12 matching residues more in TPR1, TPR2 and TPR3 than Smith−Waterman and a total of four consensus elements were confirmed.

Even for sequences with small percentage of identity, such as human tetratricopeptide repeat protein 23(25), our algorithm was able to detect more common hits than Smith−Waterman, regardless of any specific alignment metrics what makes it suitable for comprehensive analysis of structural, functional and evolutionary relationship between coding DNA data.

The problem of identifying common hits between two DNA sequences has been widely explored in the science and different solutions have been proposed. Index-based solutions, such as MUMmer, AVID and FLASH, can discover common hits faster than our algorithm, but they are expected to be worse in terms of memory complexity, due to the indexed data structure that they employ. For instance, MUMmer and AVID build suffix tree, whose size is proportional to the sum of the sequences' lengths and FLASH builds an index of all words found in both sequences, regardless if they are matching or not. There is also a preprocessing phase which precedes the actual hits' searching phase, which may be considered an additional computational cost.

Recent methods based on DNA hashing enable fast comparison on DNA words and efficient data storage. Instead of comparing words, their hash representations are mutually compared and the cost for storing hash of a DNA word is only 4 B (1 integer). Sequence Search and Alignment by Hashing Algorithm (SSAHA) [37] is one of the hash-based implementations which can be applied for discovering common hits based on their hash-representations. It assigns a unique number to each DNA nucleotide $f(b_j) \in \{0, 1, 2 \text{ or } 3\}$, $b_j \in \{A, C, T, G\}$ and calculates the hash of DNA word of k nucleotides $w : b_1 \ldots b_k$ by applying the equation $f(w : b_1 \ldots b_k) = \sum_{j=1}^{k} f(b_j) \times 4^{j-1}$. SSAHA requires 4 B (1 integer) for storing DNA hash $f(w)$, 4 B (1 integer) for hash's starting position p and 4 B (1 integer) for the index i of the sequence where the hash occurs, or 12 B (3 integers) per hashed/tracked DNA word are required. This means that for hashing/tracking 12 Kbp common hit $12,000/5 \times 12 = 28,800$ B (7,200 integers) would be required, given that $k = 5$. For tracking the same hit, our algorithm requires 16 B (four integers), which results in memory saving of $28,800 - 16 = 28,784$ B.

In general, indexing and hashing allow faster comparison of DNA words, but there is an additional memory cost for storing word's index/hash. For instance, it would be faster to compare hashes f(GCTAA) and f(GCTAG) than performing partial comparison based on linear walk on GCTAA and GCTAG, because instead of comparing five characters (5 B), integer would have to be compared with integer, for what is a required comparison of 4 B. Faster comparison, based on indexing/hashing, is only possible at the cost of increased memory requirement, because 8 B (two integers) that track the hashes of GCTAA and GCTAG would have to be stored in the memory. There is also an additional computational cost for

Table 6. Detailed analysis of matching residues in TPR1−TPR4 repeats.

Repeat	Smith−Waterman: Matching amino acids	Smith−Waterman: Longest common hit/hits	Smith−Waterman: Confirmed TPR consensus elements	Our algorithm: Matching amino acids	Our algorithm: Longest common hit/hits	Our algorithm: Confirmed TPR consensus elements
TPR1	8: K,N,E,K,V,L,C,I	K, N…I	/	18: L,L,C,E,K,A,S,S,N,Y,K,L,V,R,C,L,R,C	LC, SN, LV	/
TPR2	12: E,L,F,H,G,R,L,L,A,N,K,E	LFH	/	23: E,L,T,M,G,L,L,L,Q,K,K,E,A,E,L,T,K,A,E,L,K,E,L	KAE	G8-A20-A27
TPR3	8: A,A,Q,V,A,A,L,E	AQV, ALE	/	19: A,R,I,R,F,A,V,Q,Q,K,K,E,L,H,Y,Q,A,A,Y	FA, QA	A27
TPR4	0	/	/	14: E,G,A,D,L,A,G,S,G,R,K,L,L,I	DL	/

Note: tetratricopeptide repeat (TPR).

Table 7. Summary of the proposed algorithm's key features.

Factors	Advantages	Applications
Detection of continuous glutamine deletions in polyQ tracts of Huntington's disease associated genes	More precise detection than Smith−Waterman	Suitable for accurate detection of trinucleotide repeat associated diseases
Time complexity	Requires less nucleotide comparisons than Smith−Waterman.Requires $O(n \times k \times \log_2 m)$ time for very similar sequences, whereas there is no time improvement over Smith−Waterman for dissimilar DNA, i.e. $O(n \times m)$ time is requested	It can be applied on medium-size DNA sequences (long genes, chromosomes)
Memory complexity	Requires less memory than Huang's algorithm.	Fixed memory requirement per hit: 4 int = 16 B, ragrdless its length
Detection rate of matching nucleotides	Higher than Smith−Waterman, if it is based on metrics that assigns high penalties	Suitable for the detection of complete homology between two DNA samples, regardless of any specific alignment metrics

computing the hashes of the words, which is proportional to the length of words that are indexed. Our algorithm is a compromise between the time and memory requirement, i.e. it is expected to run slower than hash/index-based methods, but there is no cost for storing DNA index/hash and no preprocessing phase that precedes the actual hits' searching phase.

Theoretically, different sets of consistent hits can be defined if at least one of the common hits is found as a repeating sequence. For instance, if the trinucleotide GCT in the sample a : AGCTAG is extended in **GCTGCTGCT** trinucleotide microsatellite (a : A**GCTGCTGCT**AG), then, instead of one, three different sets of consistent hits, which are equal in terms of the matching content:{GCT : $(2, 4, 2, 4)$, G$(12, 12, 5, 5)$}, {GCT : $(5, 7, 2, 4)$, G$(12, 12, 5, 5)$} and {GCT : $(8, 10, 2, 4)$, G$(12, 12, 5, 5)$} can be defined for the samples a : AGCTGCTGCTAG and b : TGCTG. So, which one of the three candidates does the algorithm report?

Our algorithm works accurately for coding and noncoding DNA, due to the fact that the leftmost positioned consistent hits in minisatellite (microsatellite) DNA are always reported. For the samples a and b, the first set:{GCT : $(2, 4, 2, 4)$, G$(12, 12, 5, 5)$} is the one which is reported, because GCT : $(2, 4)$ is the leftmost positioned microsatellite triplet of all three (GCT : $(2, 4)$,GCT : $(5, 7)$ and GCT : $(8, 10)$) in the samplea : A**GCTGCTGCT**AG (Figure 13(a)). This approach enables accurate indel's prediction, since the alignment is built starting from the leftmost positioned match in the set of consistent hits. Our model is based on the set: {GCT : $(2, 4, 2, 4)$, G$(12, 12, 5, 5)$} where two GCT indels are pinpointed (Figure 13(a)). Incorrect solutions would have been derived if the second (third) candidate set of consistent hits were used, where instead of 2, 1(0) GCT local indels are predicted (Figure 13(b) and 13(c)). Table 7 summarizes the key features of our algorithm.

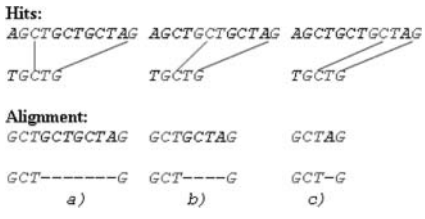

Figure 13. Aligning sequences with repeats. Hits in the microsatellite DNA: GCT (2,4) − being reported (a), GCT (5,7) (b) and GCT (8,10) (c).

Conclusions

In the present study, a TMO and more accurate algorithm than the well-known and often used dynamic programming algorithms is presented. Four crucial aspects outline the benefit of the proposed algorithm. First, our algorithm allows precise detection of missing glutamine-coding triplets in polyglutamine tracts of partial cds of huntingtin gene, due to its predisposition to pinpoint indels in less frequently occurring nucleotide pairs. This property makes our algorithm suitable for accurate detection of Huntington's disease and other diseases, which are associated with excessive trinucleotide repetitions, such as: Spinocerebellar ataxia and Dentatorubropallidoluysian atrophy. Second, our algorithm runs faster than Smith−Waterman for very similar DNA, but not at the cost of generating partial or shortened alignments, which may happen in recent developments, such as FLASH, YASS, PatternHunter and other related approaches that usually cut in the search space in order to improve the time performance. It is also very important to note that the time improvement (the reduced number of comparisons) does not affect the completeness of the solution. Third, our algorithm requires less memory than the most memory efficient dynamic programming algorithm, the Huang's algorithm, regardless if similar or dissimilar DNA is analysed. The memory complexity of the proposed algorithm is linear and it is proportional to the number of hits being detected. Fourth,

unlike Smith–Waterman, our algorithm does not reject any of the matching nucleotides which can be included in the solution and, therefore, is suitable for a complete detection of the homology between two DNA samples.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

[1] Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol. 1970;48(3):443–453.

[2] Smith T, Waterman M. Identification of common molecular subsequences. J Mol Biol. 1981;147(1):195–197.

[3] Sellers PH. An algorithm for the distance between two finite sequences. J Combin Theory Ser A. 1974;16(2):253–258.

[4] Sellers PH. The theory and computation of evolutionary distances: pattern recognition. J Algorithms. 1980;1(4):359–373.

[5] Ulam SM. Some combinatorial problems studied experimentally on computing machines. In: Zaremba SK, editor. Applications of number theory to numerical analysis. San Diego, CA, Academic Press; 1972. p. 1–3.

[6] Smith TF, Waterman MS, Fitch WM. Comparative biosequence metrics. J Mol Evol. 1981;18(1):38–46.

[7] Goad WB, Kanehisa MI. Pattern recognition in nucleic acid sequences. I. A general method for finding local homologies and symmetries. Nucleic Acids Res. 1982;10(1):247–263.

[8] Waterman MS, Eggert M. A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. J Mol Biol. 1987;197(4):723–728.

[9] Fitch WM, Smith TF. Optimal sequence alignments. Proc Natl Acad Sci U S A. 1983;80(5):1382–1386.

[10] Gotoh O. An improved algorithm for matching biological sequences. J Mol Biol. 1982;162(3):705–708.

[11] Gotoh O. Pattern matching of biological sequences with limited storage. CABIOS. 1987;3(1):17–20.

[12] Hirschberg DS. A linear space algorithm for computing maximal common subsequences. Commun ACM. 1975;18(6):341–343.

[13] Myers EW, Miller W. Optimal alignments in linear space. CABIOS. 1988;4(1):11–17.

[14] Huang X, Hardison RC, Miller W. A space-efficient algorithm for local similarities. CABIOS. 1990;6(4):373–381.

[15] Huang X, Miller W. A time-efficient, linear-space local similarity algorithm. Adv Appl Math. 1991;12(3):337–357.

[16] Sankoff D, Kruskal JB. Time warps, string edits, and macromolecules: the theory and practice of sequence comparison. 1st ed. Boston, MA: Addison-Wesley; 1983. p. 382.

[17] Fickett JW. Fast optimal alignment. Nucleic Acids Res. 1984;12(1):175–179.

[18] Ukkonen E. Algorithms for approximate string matching. Inf Control. 1985;64(1):100–118.

[19] Chao KM, Pearson WR, Miller W. Aligning two sequences within a specified diagonal band. CABIOS. 1992;8(5):481–487.

[20] Lipman DJ, Pearson WR. Rapid and sensitive protein similarity searches. Science. 1985;227(4693):1435–1441.

[21] Altschul SF, Gish W, Miller W, et al. Basic local alignment search tool. J Mol Biol. 1990;215(3):403–410.

[22] Kent WJ. BLAT—the BLAST-like alignment tool. Genome Res. 2002;12(4):656–664.

[23] Ma B, Tromp J, Li M. PatternHunter: faster and more sensitive homology search. Bioinformatics. 2002;18(3):440–445.

[24] Califano A, Rigoutsos I. FLASH: A fast look-up algorithm for string homology. In: IEEE, editor. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition; 1993 Jun 15–17; New York, NY: IEEE Computer Society; 1993.

[25] Noé L, Kucherov G. YASS: enhancing the sensitivity of DNA similarity search. Nucleic Acids Res. 2005;33(suppl 2):W540–W543.

[26] Delcher AL, Kasif S, Fleischmann RD, et al. Alignment of whole genomes. Nucleic Acids Res. 1999;27(11):2369–2376.

[27] Bray N, Dubchak I, Pachter L. AVID: A global alignment program. Genome Res. 2003;13(1):97–102.

[28] Brudno M, Do CB, Cooper GM, et al. NISC comparative sequencing program. LAGAN and multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. Genome Res. 2003;13(4):721–731.

[29] Brudno M, Morgenstern B. Fast and sensitive alignment of large genomic sequences. In: IEEE, editor. Bioinformatics Conference; 2002 Aug 14–16; Stanford, CA: IEEE Computer Society; 2002.

[30] Shen SY, Yang J, Yao A, et al. Super pairwise alignment (SPA): an efficient approach to global alignment for homologous sequences. J Comput Biol. 2002;9(3):477–486.

[31] Walker FO. Huntington's disease. The Lancet. 2007;369(9557):218–228.

[32] Craufurd D, Thompson JC, Snowden JS. Behavioral changes in Huntington disease. Cogn Behav Neurol. 2001;14(4):219–226.

[33] Orr HT, Chung MY, Banfi S, et al. Expansion of an unstable trinucleotide CAG repeat in spinocerebellar ataxia type 1. Nat Genet. 1993;4(3):221–226.

[34] Nagafuchi S, Yanagisawa H, Sato K, et al. Dentatorubral and pallidoluysian atrophy expansion of an unstable CAG trinucleotide on chromosome 12p. Nat Genet. 1994;6(1):14–18.

[35] The European Nucleotide Archive [Internet]. Heidelberg: The European Molecular Biology; [cited 2015 June 20]. Available from: http://www.ebi.ac.uk/ena/.

[36] EMBOSS Water online tool [Internet]. Heidelberg: The European Molecular Biology; [cited 2015 June 20]. Available from: http://www.ebi.ac.uk/Tools/psa/emboss_water/.

[37] Ning Z, Cox AJ, Mullikin JC. SSAHA: a fast search method for large DNA databases. Genome Res. 2001;11(10):1725–1729.