Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

PerFSeeB: Designing Long High-weight Single Spaced Seeds for Full Sensitivity Alignment with a Given Number of Mismatches

Valeriy Titarenko (valeriy.titarenko@manchester.ac.uk)
University of Manchester

Sofya Titarenko
University of Huddersfield

Research Article

Keywords: spaced seeds, lossless seed, full sensitivity, sequence alignment, mismatch, indexing

Posted Date: November 15th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-1051543/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

RESEARCH

PerFSeeB: designing long high-weight single spaced seeds for full sensitivity alignment with a given number of mismatches

Valeriy Titarenko[1*] and Sofya Titarenko[2]

*Correspondence:
valeriy.titarenko@manchester.ac.uk
[1]School of Biological Sciences,
University of Manchester,
M13 9PL, Manchester, UK
Full list of author information is
available at the end of the article

Abstract

Background: Technical progress in computational hardware allows researchers to use new approaches for sequence alignment problems. A standard procedure is usually based on pre-aligning of short subsequences followed by proper comparison of neighbouring parts. For this purpose index files are created that store all subsequences (or numbers associated with them) and their positions within a reference sequence. Index files designed on subsequences of 32–64 symbols for a human reference genome can now be easily stored without any compression even on a budget computer. The main goal now is to choose a combination of symbols (a spaced seed) that will tolerate various mismatches between reference and given sequences. An ideal spaced seed should allow us to find all such positions (full sensitivity). By increasing the seed's weight by one we usually reduce the number of candidate positions fourfold. At the same time longer seeds also reduce the number of signatures to be checked.

Results: Several algorithms to assist seed generation are presented. The first one allows us to find all permitted spaced seeds iteratively. The results obtained with the algorithm show specific patterns of the seeds of the highest weight. Among the best seeds, there are periodic seeds with a simple relation between the period of a seed, its length and the length of a read. The second algorithm generates blocks for periodic seeds. A list of blocks is found for blocks of up to 50 symbols and up to 9 mismatches. The third algorithm uses those lists to find spaced seeds for reads of an arbitrary length.

Conclusions: Lists of long high-weight spaced seeds are found and available in Supplementary Materials. The seeds are best in terms of weights compared to seeds from other papers and can usually be applied to shorter reads. Codes for all algorithms are available at https://github.com/vtman/PerFSeeB.

Keywords: spaced seeds; lossless seed; full sensitivity; sequence alignment; mismatch; indexing

Background

Sequence analysis is used to understand the features, structure, function of organisms. By comparing sequences obtained for well-known and unexplored organisms we may understand the biology of the unexplored ones. Currently, experimental techniques cannot provide us with whole sequences but only with many subsequences that need to be concatenated/merged in some way to form a long sequence. Thus, researchers have to solve a general sequence alignment problem before they

can produce any meaningful conclusions related to the biological properties of an organism.

Suppose two long sequences are similar in some way. One sequence (a *reference* sequence) is known, the other sequence is to be found. For example, as a reference sequence, we may use a human genome sequence (an "averaged" sequence based on data obtained from several persons). We want to know a genome sequence for another person, i.e. a patient with a specific disease. The current hardware for genome sequence allows us to find only chunks of the unknown genome of the "patient". Those chunks (called *reads*) are usually relatively short (hundreds of base pairs) and may contain errors. We assume that the reference and "patient" sequences are similar, so we may use the reference sequence to align a set of reads accounting for possible mismatches.

The standard procedure is to consider each read and find its possible positions within the reference sequence such that the distance between a given read and a part of the reference sequence is minimal. The distance between two strings x and y was introduced in [1] and can be determined in several ways. We may define a list of elementary operations which convert a source string x into a target string y. Each operation may have a different cost, the distance is then a sum of all costs. If we cannot transform x into y, then the cost is ∞. According to [2], depending on permissible elementary operations the most common distances are: Levenshtein (or edit) distance (insertions, deletions, substitutions are allowed at equal cost of 1; symmetric, i.e. $d(x,y) = d(y,x)$) [3], Hamming distance (only substitutions, symmetric) [4], episode distance (only insertions, is not symmetric) [5], longest common subsequence distance (insertions and deletions, all costing 1, symmetric) [6].

The similarity between two strings was originally measured using dynamic programming algorithms, see [7, 8, 9]. However, due to time complexity, the use of these algorithms became impractical for the increased size of data available. While it is natural to position a read in a way to achieve the smallest distance, the found location may not always be the best one. With progress related to sequencing hardware and the amount of data provided the original problem can be simplified as many reads may overlap, so for each position within a reference sequence, there are tens/hundreds of reads that have regions similar to a chosen reference chunk. So, ideally, we need to align reads in such a way that there are fewer discrepancies between each other when aligned. In any case, the initial problem is to find possible candidate positions for each read.

In the late 90s several new algorithms, e.g. BLAST [10, 11], appeared based on ideas of filtration and indexing. Short sequence fragments were used. To align a read we require fragments from the read to be also found in a reference sequence. The search is sped up using various indexing, e.g. hash-tables, which may provide a researcher with "false-matching" positions. The fragments were considered as contiguous segments. Once candidate positions based on full matching of contiguous regions are found, algorithms extend regions around the common parts to calculate a similarity score (seed-and-extend approach).

Originally only contiguous fragments were considered, however, in [12] and [13] spaced seeds were introduced. By a *seed* we mean a vector with all elements from a

binary alphabet. We may use a bit-wise representation (1 when a symbol at a given position is taken into account and 0 when it is ignored). We position three sequences with respect to each other (a reference sequence, a read and a seed). To find the distance we consider only those elements of the reference sequence and the read that have 1-element of the seed at the same position. When there is 0-element of the seed, the distance between the corresponding elements of the reference sequence and the read does not contribute to the total distance.

The number of 1s within a seed is called its *weight*, while the number of all symbols (0s or 1s) gives its *length*/*span*. In [12] the most sensitive seed was `111010010100110111`. Some spaced seeds for different weights can be found in [14]. The idea of spaced seeds was extended for other problems: vector [15], indel [16], neighbour [17] seeds. In ZOOM software [18] spaced seeds are generated to perform alignment with at most 2 mismatches. In PerM software [19] so-called periodic spaced seeds are used to improve the efficiency of mapping. Fast alignment-free string comparison based on spaced seeds (spaced-words) is discussed in [20].

Later approaches based on multiple seeds were used, e.g. [21, 22] (a good review can be found in [23]). Seeds in a multiple seed environment are designed to have less overlap between each other and thus are increasing chances to hit common regions.

Seed design may also deal with a non-binary alphabet. For example, in YASS [24] a three-letter alphabet (#, @, −) is used, where # stands for a nucleotide match, @ is for a match or transition ($A \leftrightarrow G$ or $C \leftrightarrow T$ mutations) and − is used when we ignore corresponding symbols.

The sensitivity of seeds is usually measured using probabilistic approaches. In this paper, we consider an extreme case of *full sensitivity* seeds under the assumption that a read has at most n_m mismatches. This means that these seeds should allow us to find all candidate positions. Full sensitivity seeds are also known as *lossless seeds* [25] and were applied to filtration problems. A lossless filtration means that all fragments will be found (in case of *lossy* filtration we may miss some of them).

For a given length of reads, there are a lot of seeds that can be used for our task. In the general case, a seed of higher weight should provide us with less candidate positions for alignment. As there are 4 symbols for genome alignment problems, then by increasing the weight of seed by one we reduce the number of candidate positions by 4 times. This factor 4 is valid only in the perfect case of discrete uniform distribution when all four symbols A, C, G, T have the same chance to appear at a given position in a sequence and all such events for any two positions are independent. In practice, the factor is smaller, however, this does not change the fact that seeds of higher weight are preferable for alignment problems.

There may be several seeds of the maximum weight. Longer seeds require us to generate and check fewer index/hash values. We observed that most long high-weight seeds have a periodic structure. However, this structure is not a perfect one like discussed in [25] where a seed is made of several repetitions of structure A separated by structure B. We have an integer number of repetitions of structure A and a "remainder" of this structure (which is a number of first elements of structure A). The size of a periodic block has a simple relation with lengths of reads and the seed. It is possible to generate those periodic blocks within a reasonable time frame (from a fraction of a second for blocks less than 30 symbols to several hours for cases of 50 symbols).

Methods

Problem setup

Suppose there is a *reference* sequence of length L and we are given a set of shorter sequences (*reads*) of length n_r. For human genome problems $L \approx 3.2 \cdot 10^9$ and $n_r \approx 100$–300. Let there be a seed s of length n_s and weight w. Each symbol of sequences is usually either a letter A, C, G, T or a symbol N when the symbol is unknown or specific masking has been applied. For simplicity, we assume that only 4 letters are present. Therefore each symbol can be coded with 2 bits. When a seed is positioned at given elements of the reference sequence or a read, corresponding 1-elements select w elements within the sequences. Thus we may form $2w$-bit numbers. For a read we get $(n_r - n_s + 1)$ numbers, for the reference sequence there are $(L - n_s + 1) \approx L$ numbers. Therefore we get a list of pairs ("position", "signature"). We may sort the list by "signature" values. In this way for each $2w$-bit "signature" value found of a read, we may obtain a set of positions within the reference sequence having the same "signatures" values.

For human genome problems all positions can be coded with a 32-bit number. Therefore storing the whole list in memory requires $L(32 + 2w)$ bits. If $w = 32$, then the memory required is 39 GB; for $w = 64$ we need 64 GB. After the list is sorted by "signature" values we may split it on shorter vectors having the same number of identical bits which can be ignored for the whole vector. For example, a spaced seed of weight 24 provides us with a 24-vector of letters A, C, G, T for a given position within the reference sequence. We may combine all pairs ("position", "signature") in groups having the same 8 letters for "signature", e.g. the first 8 letters. Therefore we may omit those 8 letters in a chosen group and store only the other $24 - 8 = 16$ letters. Of course, by varying the number of the same letters in groups, we may increase or decrease the number of such groups. However, it is safe to assume that 8 letters are used, so we reduce storage requirements: $L(16 + 2w)$, i.e. 32 GB for $w = 32$ and 58 GB for $w = 64$. This means that index files created for a chosen seed can be easily stored even on a budget computer. In principle, we may create index files for multiple seeds of different weight, however in this paper we focus only on a single seed case.

In many other approaches based on probabilistic ideas quality of seeds generated is based on sensitivity, i.e. the chance that seed will allow us to find all candidate positions within a reference sequence. Here we consider the extreme case and require that a seed allows us to find *all* positions. On the other hand, we also restrict the matching of two sequences to a specific case. We set a number n_m of mismatches and consider full matching when two strings are of the same length and have not more than n_m mismatches. Cases of insertions/deletions, transitions/transversions are not considered in this paper and will be discussed in future papers. This approach seems to be a limited one. However, insertions/deletions are very small (but important) fractions among all other cases, thus we may successfully align almost all reads. So, for reads that are not aligned we may 1) use other approaches, e.g. those based on dynamic programming (see X-drop approach to improve performance [26]) or seeds designed to account for other mutations, 2) align the second sequence for paired-end data, 3) align with respect to already pre-aligned other reads not containing insertions/deletions. Thus in this paper, we deal with full sensitivity seeds that allowed us to have not more than n_m mismatches.

By increasing the seed's weight we reduce the number of positions within the reference sequence that have identical "signature" values. So, we aim to find seeds of the highest weight possible for reads of given lengths. Several seeds may exist. To choose the best one we pick up seeds of maximum length. In this way, we are reducing the number $(n_r - n_s + 1)$ of "signatures" generated for each read. Of course, each seed may provide us with a different number of candidate positions within a reference sequence. Therefore we cannot guarantee that a smaller number of "signature" values will provide us with an also smaller number of total candidate positions to be processed compared to a larger number of different "signatures" obtained with a different seed. However, in a general case of pure random sequences, this should be true, i.e. the total number of candidate positions should be proportional to $(n_r - n_s + 1)$.

The procedure to choose the best seeds is described below. We set a positive integer number w and for a given number of mismatches n_m and length n_r of reads we check if there are any seeds that have weight w. It is clear that by increasing lengths n_r we may increase maximum possible weight of seeds we can generate. However, the maximum weight of seeds may stay constant for a range of n_r values. If we have two lengths n_{r1} and n_{r2} of reads $(n_{r1} < n_{r2})$, then all seeds found for (n_m, w, n_{r1}) triple are also valid seeds for (n_m, w, n_{r2}). There may be seeds for (n_m, w, n_{r2}) that are not valid seeds for (n_m, w, n_{r1}). Therefore there is a higher chance that the longest seeds found for (n_m, w, n_{r1}) are shorter than the longest seeds found for (n_m, w, n_{r2}). While it may be good to choose the longest seeds valid for (n_m, w, n_{r2}), they may not be valid seeds for (n_m, w, n_{r1}). Therefore, once we have found seeds valid for (n_m, w) by increasing value n_r iteratively, we stop and choose the longest seeds among them.

Seed validation

Let there be a seed s of length n_s. We want to check if the seed meets the full sensitivity requirement for any read of length n_r and a given maximum number n_m of mismatches. Suppose n_m and n_r are set. Then we create n_m-vector of indices i_k, $k = 1, 2, \ldots, n_m$ such that $1 \le i_k \le n_r$. By definition, the length of a seed is not greater than the length of a read, $n_s \le n_r$. We may shift the first element of a seed by δ with respect to the first element of the read. As the last element of the seed should be within the read's elements, then δ can vary from 0 to $(n_r - n_s)$. For each value of δ we check that none of 1-elements of the seed shifted by δ has indices i_k, $k = 1, 2, \ldots, n_m$. If for any possible combination of indices i_k, there is at least one value of δ (depending on i_k, $k = 1, \ldots, n_m$) the requirement is met, then the seed is a *valid* seed for given n_m and n_r. See Figure 1.

Parameter δ has $(n_r - n_s + 1)$ values. By padding the seed vector with 0-elements from left and right we can form $(n_r - n_s + 1)$ vectors of length n_r. We pad from left and right and the total number of 0-elements to be used for each vector is $(n_r - n_s)$. Padding can be performed in any way, however it may be more convenient to have the first element of the seed aligned with the first element of the read for the first vector, the next vectors are just the previous vectors padded by one 0-element from left (see Figure 1). Or we may align the last elements of a seed and read and pad by 0-element from right (see Figure 4).

In any case for a given seed we form $(n_r - n_s + 1)$ vectors of length n_r. We generate all possible combinations of i_k indices. There are n_m indices and we want them all to be different. There are

$$C_{n_r}^{n_m} \equiv \frac{n_r!}{n_m!(n_r - n_m)!}, \qquad m! \equiv 1 \cdot 2 \cdot 3 \cdots m \tag{1}$$

such combinations. If for any of these combinations all $(n_r - n_m + 1)$ vectors/rows contain at least one 1-element for chosen indices i_k, then the seed cannot be used, otherwise, the seed meets the requirements.

We want seed validation to be performed as quickly as possible on a modern computer. Therefore we may store each n_r-vector as an n_r-bit number. It may often be reasonable to pad each vector with 1-elements to a 128-bit number (or several 128-bit numbers). Then SIMD instructions (single instruction, multiple data) can be used to perform comparison or various logical operations. For a given set of indices i_k, $k = 1, \ldots, n_m$ we may create a 128-bit vector V such that all its elements are 1 except i_k-th ones which are 0. Then for each row R_p, $p = 1, \ldots, (n_r - n_m + 1)$ we check if $R_p|V$ equals V (by "|" we mean bitwise logical OR operation). If it is true, then the seed can be used for the given combination of n_m indices. In Figure 2 we use examples for Figure 1. For convenience we pad all vectors to 24-vectors (by merging with four 1-elements). For example 1 ($i_1 = 7$, $i_2 = 12$) only one row (R_3) has met the requirement $R_p|V = V$, therefore for this combination of indices the seed can be used. However, for example 3 ($i_1 = 4$, $i_2 = 13$) no row meets the requirement. Therefore, the seed cannot be used for the given combination of indices and, as the result, it cannot provide us with full sensitivity. Only when the procedure is successful for all $C_{n_r}^{n_m}$ combinations, then the seed is valid.

There is an alternative approach to validate seeds. We create n_r binary vectors/columns U_t, $t = 1, \ldots, n_r$, of length $(n_r - n_m + 1)$, see Figure 3. As before we may also pad them with 1-elements to form numbers of a given length, e.g. 32-, 64- or 128-bit numbers. The task is to consider all $C_{n_r}^{n_m}$ combinations of columns U_t and perform logical OR operation, i.e.

$$U_{j_1}|U_{j_2}|\ldots|U_{j_{n_m}}, \tag{2}$$

and check if the resultant column has all 1-elements (let us call it the *saturated* vector).

Now we try to reduce the number of columns. We may identify the same columns and leave only different ones (10 out of 36 columns are removed in the example in Figure 4). So, as $n_m = 4$ and $n_r = 36$, then instead of $C_{36}^4 = 58905$ combinations, we should check only $C_{26}^4 = 14950$. The next step is to identify those columns that are subsets of other columns. We denote $U_k \subset U_m$ if $U_k|U_m$ is U_m. This means that all 1-elements of column U_k are also 1-elements of U_m (however, there may be positions q such that element q of U_m is 1-element but element q of U_k is 0-element). Removing columns that are subsets of other columns allows us to further decrease the number of combinations. For the example in Figure 4 we excluded extra 16 columns, so the total number of combinations is $C_{36-10-16}^4 = C_{10}^4 = 210$, i.e. the number of combinations we need to check is 280 times fewer compared to

the original case. Note that similar approaches can be used when we form resultant columns. For instance, we may generate i_k indices iteratively. We consider i_1 from 1 to $(n_r - n_m + 1)$ (as other $(n_m - 1)$ indices are to be generated, they all should be different, so i_1 can never be equal to n_r). Then i_2 is from $(i_1 + 1)$ to $n_r - n_m + 2$, similarly i_3 is from $(i_2 + 1)$ to $(n_r - n_m + 3)$, and so on. If $U_{i_3} \subset (U_{i_1}|U_{i_2})$, then U_{i_3} can be skipped.

Seed expansion

It is agreed that a seed should start and end with 1-element. So, there is only 1 seed of length 1 (`1`), there is also only 1 seed of length 2 (`11`), 2 seeds of length 3 (`101`, `111`), 4 seeds of length 4 (`1001`, `1011`, `1101`, 1111). Therefore for a read of length n_r there are

$$1 + 1 + 2 + 4 + \ldots + 2^{n_r - 2} = 2^{n_r - 1} \tag{3}$$

seeds in total. As validation of each seed is also quite time-consuming, we should identify ways to reduce the number of candidate seeds. For example, seed `10001110100100011101` is a valid seed for $n_r = 30$ and $n_m = 3$. Any subset of this seed is also a valid seed (of lower weight). So, `1110100100011101`, `100011101001000111`, `10001010100100010101` are also valid seeds. Therefore, we can implement the following procedure. Assume we have already found all valid seeds of length less or equal to k. We pad all these seeds with 0-elements from their right ends, so we get vectors of length k. Then we concatenate them with 1-element (also at the right ends of the padded seeds). Before applying the validation procedure we form a seed by removing the first 1-element and all 0-elements from the left end and check that the seed is in the list of valid seeds. For example, an original valid seed was `100111011` (of length 9) and we aim to find valid seeds of length 13, then the extended seed is `1001110110001` and we need to check if `1110110001` is already in the list of valid seeds. Clear that seed is valid if and only if the reverse seed (the order of elements is reversed) is valid. So, seeds `1001110110001` and `1000110111001` are valid (or are not valid) at the same time.

Periodic seeds

The above procedure allows reducing the number of candidate seeds. However, there are still a lot of seeds to be validated, so finding all possible seeds for a read's length of more than 45 is very slow. However, we can observe a very important property. There are almost always periodic seeds such that

$$n_r = n_s + T - 1, \tag{4}$$

where T is the size of the periodic block. Such seeds have a whole number n_b of these periodic blocks and the "remainder" (the first $n_d > 0$ elements of the block), so $n_s = n_b \cdot T + n_d$. There may be seeds of different period T for the same length n_r of a read. For example, in Figure 5 there are seeds found for $n_r = 43$, $n_m = 4$. The maximum weight is $w = 12$. We get two pairs of seeds for $T = 19$, three pairs for $T = 17$ and one for $T = 13$.

Note that when such seeds can be found there are often other (shorter) seeds available. There are also several exceptions from the observation. For some values of n_r and n_m we may have seeds such that (4) is valid for smaller values of n_r, e.g. $n_r - 1$, $n_r - 2$. However, there may be cases when the formula is not true for the best seeds. See Table 1. For 80% of read lengths formula (4) is valid for a given value of n_r, for 87% is true when smaller n_r can be used.

Suppose there is a periodic seed such that $n_s = n_b T + n_d$ and formula (4) is true. We want to check if a seed is a valid one for a given n_m, see Figure 6. We need to generate indices j_k, $k = 1, \ldots, n_r$ such that $1 \leq j_k \leq n_r$ and check if the resultant column $U_{j_1}|U_{j_2}|\ldots|U_{n_m}$ is the saturated vector. If the seed is valid, then its periodic block should also meet the same requirements. For this purpose, we just choose j_k such that $(1+T) \leq j_k \leq 2T$. Now we assume that the resultant column for the periodic block is not the saturated one for all possible combinations of indices. For any index j_k, $1 \leq j_k \leq n_r$, we may write down $j_k = j_k^* + m \cdot T$, where $(1+T) \leq j_k^* \leq 2T$ and m is an integer number. Therefore $U_{j_k} \subset U_{j_k^*}$ and instead of U_{j_k} we may consider $U_{j_k^*}$.

Periodic blocks

Validation of a periodic block is done in the way of validation of the whole seed. We just need to generate all possible combinations of indices j_k, $k = 1, \ldots, n_m$ such that $1 \leq j_k \leq T$ and check if the resultant column is not the saturated one. A periodic block is valid if and only if its reverse is valid. We may also perform a cyclic rotation of elements of the block, those new blocks are valid at the same time. As the result, we have groups of $2T$ periodic blocks of length T (some of them may be identical) that are valid/not valid at the same time. We call them *equivalent* blocks. Therefore it is enough to consider the validity of only one block. To reduce the number of blocks to be considered we require that the first element is always 0-element and the last element is always 1-element. See examples of equivalent blocks generated for a given periodic block in Figure 7.

We need a procedure to choose only one block instead of $2T$ blocks. Of course, for some blocks (e.g. `00101101`) there may be cases of identical blocks obtained via reversion/cyclic rotation. By varying indices j_k we create patterns of 0-elements. Those patterns should be within the periodic block (or one of the equivalent blocks obtained by cyclic rotation) since there should be a row such the corresponding elements of the chosen columns are all 0-elements. Therefore each periodic block must have a contiguous chunk of n_m 0-elements. Therefore when we generate various periodic blocks we may always assume that the first n_m elements are 0-elements. To choose one block we pick up the one with the highest number of 0-elements at the beginning. As among blocks obtained for the reverse block, there will also be a block with the maximum number of 0-elements, we need to perform a further comparison. We may count the number of 1-elements in contiguous blocks from the left/right side of the zero-block and choose the block with fewer 1-elements. If the contiguous are the same, then consider neighbouring blocks of 0-elements (and choose the smallest one). We repeat the procedure if it is needed.

As we want to find seeds of maximum weight, it is reasonable to also find periodic blocks of maximum weight. Suppose there are blocks found for weight w. By

replacing 1-elements with 0-elements we form periodic blocks of weight $(w-1)$. However, there may also be blocks of weight $(w-1)$ that cannot be formed by replacing 1-elements in blocks of weight w. For example, there are 5 blocks for $T=11$, $n_m=2$ and weight $w=7$: `00011011111`, `00101011111`, `00101110111`, `00101111011`, `00110101111`. Block `00010110111` of weight $w=6$ cannot be formed from those blocks. Seeds are formed of an integer number of periodic blocks and a "remainder". There is a possibility that the best seeds are formed of blocks of non-maximum weight. However, when we generated seeds of non-maximum weight, they never formed seeds of weight larger than seeds formed from maximum-weight blocks.

All ideas mentioned above allow us to reduce the number of blocks to be validated. It is also possible to parallelise processing, so each CPU thread validates only specific seeds from a pre-generated list. Together with SIMD instructions for validation steps generation of periodic blocks is sped up. It may take less than a second for $n_r<35$, however, finding periodic blocks for $n_r\approx 50$ may still take hours as trillions of blocks should be validated.

Forming periodic spaced seeds

The final step of PerFSeeB approach (periodic full sensitivity blocks) is to form spaced seeds of maximum weight. For this purpose, we consider all periodic blocks formed for a given number n_m of mismatches. As $n_s=n_bT+n_d$, $n_b\geq 1$, $1\leq n_d<T$, and $n_r=n_s+T-1$, then $n_r=n_bT+n_d+T-1=(n_b+1)T+(n_d-1)\geq 2T$ or $T\leq n_r/2$. For each periodic block we form all equivalent blocks and check if 1) the first symbol of the block is 1-element, 2) n_d-th element is also 1-element. In principle, there is no need to consider blocks obtained from the reverse periodic block as the final seed will be a reverse of seeds formed from the original block. Once the seed is formed we count its weight. By processing all periodic blocks we find seeds of maximum weight.

Results

We have written a code to generate the best periodic blocks of size T for a given number n_m of mismatches. A user can specify an initial number n_1 of 1-elements in those blocks. If no valid blocks are found for a given n_1, then the value of n_1 is incremented by one and the procedure restarts until at least one valid block is found. The code was applied for n_m values from 2 to 9, and for T values from 10 to 50 (and from 10 to 70 for $n_m=2$). Maximum weight n_1 and density of best blocks (defined as n_1/T) are shown in Table 2.

Note that while values n_1 and ρ tend to increase with the block size, they are not monotonic in general. See examples for $n_m=5$ and $n_m=8$ in Figure 8 (more figures are in Supplementary materials).

When all maximum-weight blocks are found for a given range of T, we may find periodic spaced seeds of the maximum weight for any given value of n_r. Of course, there may be cases of seeds composed of periodic blocks of different sizes. For example, if the length of reads is $n_r=35$, the maximum weight of possible seeds is $w=17$ and there are seeds formed for blocks of sizes 7, 10, 11, 13, e.g.

- $T=7$, $n_b=4$, $n_d=1$, $b=$ `1011100`, $s=$ `10111001011100101110010111001`;

- $T = 10, n_b = 2, n_d = 6, b = 1101111000, s = 1101111000110111000110111$;
- $T = 11, n_b = 2, n_d = 3, b = 11111000110, s = 11111000110111111000110111$;
- $T = 13, n_b = 1, n_d = 10, b = 1011111011100, s = 10111110111001011111011$.

For a given length n_r of reads, we plot sizes of corresponding periodic blocks, see Figure 9. We see that sizes of best blocks tend to work in a specific range of n_r values. Best blocks are often those having peak density values. For example, if $n_m = 8$, the most frequent block sizes are $T = 18, 21, 33, 36, 39$ (Figure 9), we see peak density values for them in Figure 8. Note that values of T also increase with the length of reads. So, while we can find seeds for any n_r, they may not be the densest as there we have not generated periodic blocks for large values of T.

The final goal is to set the weight w of seeds, then find the minimum length of reads that have valid seeds of this weight and choose the longest seeds when several seeds are available. In Table 3 we present examples of these seeds for weights w (multiples of 8). As seeds may be very long we present only periodic blocks, number n_b of these blocks and values of "remainders" n_d, see Tables 4 and 5. The full list of spaced seeds found for various values of n_r and w ($n_r \leq 400$, $w \leq 320$) is in Supplementary Materials. Users can generate their seeds using codes (https://github.com/vtman/PerFSeeB).

We plot weights of the best seeds as a function of reads' length in Figure 10. Based on lemmas in [27], it is possible to show that for a contiguous seed of weight/length q the minimum length of reads when the seeds are valid is $K = q(n_m + 1)$, so the ratio is $r = q/K \to r_\infty = \frac{1}{n_m+1}$ when $q \to \infty$. The corresponding values of r_∞ are also shown in Figure 10 (dashed lines). Depending on values of n_m and n_r the spaced seeds are more dense (per read's length) compared to contiguous seeds by 20–68% ($n_r = 50$), 40–95% ($n_r = 100$), 60–113% ($n_r = 150$), 70–128% ($n_r = 200$).

Discussion

To compare quality of spaced seed generated with PerFSeeB approach we present a list of most popular seeds in Table 6. These seeds are generated for given sensitivity levels. When there were several seeds presented in a paper, we choose a seed obtained for highest sensitivity levels. We may see that the seeds are usually relatively short and of a smaller weight. Unlike PerfSeeB approach, the other algorithms did not usually put any restrictions on the number of mismatches. Several seeds were generated for a multiple-seed approach.

Our goal is to have full sensitivity seeds. Therefore we aim to check how the other seeds work under such requirements. For a given seed and a number of mismatches n_m we vary lengths' n_r of reads in order to check when the seed is valid. These numbers are provided in Table 7. We may see that seeds generated directly with the extension procedure (usually $n_r < 45$) and seeds generated with PerFSeeB approach (they all called "best" seeds in Table 7) are valid for shorter reads: by approximately 15% for $w \leq 12$, by 20% for $w = 20, 22$ and by 32% for $w = 39$–46. We have also provided weights of PerFSeeB seeds for lengths n_r found for each seed. These weights are usually 30–40% higher compared to the given ones.

Conclusions

PerFSeeB approach proposed in this paper is based on the designing of periodic blocks. Resultant spaced seeds are guaranteed to find all positions within a reference

sequence when a given number of mismatches is set. Each periodic seed is an integer number of periodic blocks and a "remainder" (a number of the first symbols of the block). The size of a periodic block is the difference between read's and seed's lengths plus one. This relation is empirical and was observed for seeds generated by iterative extension procedure for reads of length less than 45.

Periodic blocks are found for the number mismatches n_m from 2 to 9 and block's length T up to 50 (or 70 for $n_m = 2$). Those blocks can be used to generate spaced seeds required for any given length of reads. The seeds found with PerFSeeB approach might not be of the highest weight for long reads (>200) as longer periodic blocks are needed to be found, however, they definitely meet the requirements. Codes used to generate periodic blocks and final seeds are designed to account for SIMD instructions, can work in a multithreading environment and are publicly available at (https://github.com/vtman/PerFSeeB). While the authors did their best to minimise the number of candidate blocks to be validated, there may be trillions of them to be checked for periods T around 50.

Declarations

Abbreviations
Not applicable.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Availability of data and materials
The codes to generate periodic blocks and seeds are publicly available at https://github.com/vtman/PerFSeeB.

Competing interests
The authors declare that they have no competing interests.

Funding
Not applicable.

Authors' contributions
VT and ST conceived the ideas for the study. VT wrote codes to generate the seeds. Both authors wrote the manuscript and approved its final version.

Acknowledgements
The authors would like to thank Dr Laurent Noé for fruitful discussion of the ideas and his comments.

Author details
[1]School of Biological Sciences, University of Manchester, M13 9PL, Manchester, UK. [2]School of Computing and Engineering, University of Huddersfield, HD1 3DH, Huddersfield, UK.

References

1. Hamming RW. Error detecting and error correcting codes. The Bell System Technical Journal. 1950;29(2):147–160.
2. Navarro G. A Guided Tour to Approximate String Matching. ACM Computing Surveys. 2001 3;33(1):31–88.
3. Levenshtein VI. Bounds for Codes Ensuring Error Correction and Synchronization. Problems Inform Transmission. 1969;5:1–10. Available from: mi.mathnet.ru/eng/ppi1793.
4. Kruskal JB. An Overview of Sequence Comparison: Time Warps, String Edits, and Macromolecules. SIAM Review. 1983;25(2):201–237. Available from: http://www.jstor.org/stable/2030214.
5. Das G, Fleischer R, Gasieniec L, Gunopulos D, Kärkkäinen J. Episode matching. In: Apostolico A, Hein J, editors. Combinatorial Pattern Matching. Berlin, Heidelberg: Springer Berlin Heidelberg; 1997. p. 12–27.
6. Apostolico A, Guerra C. The longest common subsequence problem revisited. Algorithmica. 1987 11;2(1):315–336.
7. Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology. 1970;48(3):443–453.
8. Smith TF, Waterman MS. Identification of common molecular subsequences. Journal of Molecular Biology. 1981;147(1):195–197.

9. Durbin R, Eddy SR, Krogh A, Mitchison G. Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge: Cambridge University Press; 1998.
10. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. Journal of Molecular Biology. 1990;215(3):403–410.
11. Altschul SF, Madden TL, Schäffer AA, Zhang J, Zhang Z, Miller W, et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Research. 1997 09;25(17):3389–3402.
12. Ma B, Tromp J, Li M. PatternHunter: faster and more sensitive homology search. Bioinformatics. 2002 3;18(3):440–445.
13. Burkhardt S, Kärkkäinen J. Better Filtering with Gapped q-Grams. In: Amir A, editor. Combinatorial Pattern Matching. Berlin, Heidelberg: Springer Berlin Heidelberg; 2001. p. 73–85.
14. Choi KP, Zeng F, Zhang L. Good spaced seeds for homology search. Bioinformatics. 2004 02;20(7):1053–1059.
15. Brejová B, Brown DG, Vinař T. Vector seeds: An extension to spaced seeds. Journal of Computer and System Sciences. 2005;70(3):364–380.
16. Mak D, Gelfand Y, Benson G. Indel seeds for homology search. Bioinformatics. 2006 07;22(14):e341–e349.
17. Csűrös M, Ma B. Rapid Homology Search with Neighbor Seeds. Algorithmica. 2007 6;48(2):187–202.
18. Lin H, Zhang Z, Zhang MQ, Ma B, Li M. ZOOM! Zillions of oligos mapped. Bioinformatics. 2008 08;24(21):2431–2437.
19. Chen Y, Souaiaia T, Chen T. PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds. Bioinformatics. 2009 08;25(19):2514–2521.
20. Leimeister CA, Boden M, Horwege S, Lindner S, Morgenstern B. Fast alignment-free sequence comparison using spaced-word frequencies. Bioinformatics. 2014 04;30(14):1991–1999.
21. Li M, Ma B, Kisman D, Tromp J. PatternHunter II: highly sensitive and fast homology search. Journal of Bioinformatics and Computational Biology. 2004;2(3):417–439.
22. Sun Y, Buhler J. Designing Multiple Simultaneous Seeds for DNA Similarity Search. Journal of Computational Biology. 2005;12(6):847–861.
23. Brown DG. 6. In: A Survey of Seeding for Sequence Alignment. John Wiley & Sons, Inc.; 2008. p. 117–142.
24. Noé L, Kucherov G. YASS: enhancing the sensitivity of DNA similarity search. Nucleic Acids Research. 2005 7;33(suppl_2):W540–W543.
25. Kucherov G, Noe L, Roytberg M. Multiseed lossless filtration. IEEE/ACM Transactions on Computational Biology and Bioinformatics. 2005;2(1):51–61.
26. Zhang Z, Schwartz S, Wagner L, Miller W. A Greedy Algorithm for Aligning DNA Sequences. Journal of Computational Biology. 2000;7(1-2):203–214.
27. Pevzner PA, Waterman MS. Multiple filtration and approximate pattern matching. Algorithmica. 1995 2;13(1):135–154.
28. Buchfink B, Xie C, Huson DH. Fast and sensitive protein alignment using DIAMOND. Nature Methods. 2015 1;12(1):59–60.
29. Homer N, Merriman B, Nelson SF. BFAST: An Alignment Tool for Large Scale Genome Resequencing. PLOS ONE. 2009 11;4(11):1–12.
30. Salmela L, Mukherjee K, Puglisi SJ, Muggli MD, Boucher C. Fast and accurate correction of optical mapping data via spaced seeds. Bioinformatics. 2019 09;36(3):682–689.
31. Hahn L, Leimeister CA, Ounit R, Lonardi S, Morgenstern B. rasbhari: Optimizing Spaced Seeds for Database Searching, Read Mapping and Alignment-Free Sequence Comparison. PLOS Computational Biology. 2016 10;12(10):1–18.
32. Egidi L, Manzini G. Better spaced seeds using Quadratic Residues. Journal of Computer and System Sciences. 2013;79(7):1144–1155.

Figure 1 Checking if a seed is a full sensitivity one for a given length of a read and number of mismatches.

Figure 2 Seed validation based on padded rows. When the requirement is met, the row is in green colour, otherwise, in red colour.

Figure 3 Seed validation based on columns. When the requirement is met, the row is in green colour, otherwise, in red colour.

Figure 4 Reducing the number of combinations when checking seed's validity.

Figure 5 All seeds of maximum weight (12) found for reads of length 43, number of mismatches is 4. Flipped seeds are not shown.

Figure 6 A seed is valid only when its periodic block is valid.

Figure 7 Equivalent periodic blocks generated for block 1011101001000100111000011010100010101. Only blocks with the first 0-element and last 1-element are shown. The bottom part is for the reverse block. The circled block is the one we choose from the group.

Figure 8 Maximum numbers of 1-elements per length of periodic blocks, $n_m = 5$ and $n_m = 8$.

Figure 9 Sizes of best periodic blocks for a given length of reads.

Figure 10 Weights of the best spaced seeds per reads' lengths (solid lines). Maximum ratios for contiguous seeds (dashed lines).

Table 1 Exceptions from the observed formula. If the formula is valid for shorter reads, then the corresponding n_r is in parentheses, otherwise (×).

n_m	n_r, min	n_r, max	Exceptions (n_r values)
2	12	42	no
3	11	42	11 (×), 13 (12), 39 (38)
4	20	52	no
5	13	50	13 (12), 16 (×), 17 (×), 19 (18), 20 (×), 47 (46)
6	14	53	15 (14), 17 (16), 19 (18), 23–26 (×), 34 (×), 38 (×), 39 (×), 47 (×), 48 (×), 52 (×)
7	16	46	17 (16), 19 (18), 21–23 (×), 25 (24), 26 (×), 27 (×), 29 (28), 43 (×), 45 (44)
8	18	46	19 (18), 21 (20), 23–26 (×), 28–32 (×), 34 (33), 42–44 (×)

Table 2 Maximum weight n_1 and density ρ of periodic blocks (period T, number of mismatches n_m).

T	$n_m=2$		$n_m=3$		$n_m=4$		$n_m=5$		$n_m=6$		$n_m=7$		$n_m=8$		$n_m=9$	
	n_1	$\rho,\%$	n_1	$\rho,\%$	n_1	$\rho,\%$	n_1	$\rho,\%$	n_1	$\rho,\%$	n_1	$\rho,\%$	n_1	$\rho,\%$	n_1	$\rho,\%$
10	6	60.0	4	40.0	3	30.0	2	20.0	1	10.0	1	10.0	1	10.0	1	10.0
11	7	63.6	5	45.5	3	27.3	2	18.2	1	09.1	1	9.1	1	9.1	1	9.1
12	8	66.7	5	41.7	3	25.0	3	25.0	2	16.7	2	16.7	1	8.3	1	8.3
13	9	69.2	6	46.2	4	30.8	3	23.1	2	15.4	1	7.7	1	7.7	1	7.7
14	9	64.3	6	42.9	4	28.6	4	28.6	3	21.4	2	14.3	1	7.1	1	7.1
15	10	66.7	8	53.3	5	33.3	4	26.7	2	13.3	2	13.3	2	13.3	2	13.3
16	11	68.8	8	50.0	5	31.3	4	25.0	3	18.8	3	18.8	1	6.3	1	6.3
17	12	70.6	8	47.1	6	35.3	4	23.5	3	17.6	2	11.8	2	11.8	1	5.9
18	13	72.2	9	50.0	6	33.3	5	27.8	3	16.7	3	16.7	3	16.7	2	11.1
19	14	73.7	10	52.6	7	36.8	5	26.3	4	21.1	3	15.8	2	10.5	2	10.5
20	14	70.0	10	50.0	8	40.0	6	30.0	4	20.0	4	20.0	3	15.0	3	15.0
21	16	76.2	11	52.4	8	38.1	6	28.6	5	23.8	4	19.0	4	19.0	2	9.5
22	16	72.7	12	54.5	8	36.4	7	31.8	5	22.7	5	22.7	3	13.6	3	13.6
23	17	73.9	12	52.2	9	39.1	7	30.4	5	21.7	4	17.4	3	13.0	3	13.0
24	18	75.0	14	58.3	9	37.5	8	33.3	5	20.8	5	20.8	4	16.7	3	12.5
25	19	76.0	14	56.0	10	40.0	7	28.0	6	24.0	5	20.0	4	16.0	4	16.0
26	20	76.9	14	53.8	10	38.5	9	34.6	6	23.1	6	23.1	4	15.4	4	15.4
27	21	77.8	15	55.6	11	40.7	9	33.3	6	22.2	5	18.5	5	18.5	3	11.1
28	22	78.6	16	57.1	11	39.3	9	32.1	8	28.6	6	21.4	4	14.3	4	14.3
29	22	75.9	16	55.2	12	41.4	9	31.0	7	24.1	6	20.7	5	17.2	4	13.8
30	23	76.7	17	56.7	13	43.3	10	33.3	8	26.7	8	26.7	6	20.0	5	16.7
31	25	80.6	18	58.1	16	51.6	10	32.3	8	25.8	6	19.4	5	16.1	4	12.9
32	25	78.1	19	59.4	14	43.8	11	34.4	8	25.0	8	25.0	5	15.6	5	15.6
33	26	78.8	20	60.6	14	42.4	11	33.3	8	24.2	7	21.2	7	21.2	5	15.2
34	27	79.4	20	58.8	15	44.1	12	35.3	9	26.5	8	23.5	6	17.6	6	17.6
35	28	80.0	21	60.0	15	42.9	12	34.3	10	28.6	8	22.9	6	17.1	6	17.1
36	29	80.6	22	61.1	16	44.4	13	36.1	10	27.8	9	25.0	8	22.2	6	16.7
37	30	81.1	22	59.5	17	45.9	13	35.1	10	27.0	8	21.6	6	16.2	6	16.2
38	30	78.9	23	60.5	18	47.4	14	36.8	10	26.3	10	26.3	7	18.4	7	18.4
39	32	82.1	24	61.5	18	46.2	14	35.9	10	25.6	9	23.1	9	23.1	6	15.4
40	32	80.0	27	67.5	18	45.0	14	35.0	11	27.5	10	25.0	8	20.0	8	20.0
41	33	80.5	25	61.0	19	46.3	14	34.1	11	26.8	9	22.0	8	19.5	6	14.6
42	34	81.0	26	61.9	19	45.2	16	38.1	13	31.0	11	26.2	10	23.8	8	19.0
43	35	81.4	26	60.5	20	46.5	15	34.9	12	27.9	10	23.3	8	18.6	7	16.3
44	36	81.8	28	63.6	21	47.7	16	36.4	12	27.3	11	25.0	9	20.5	8	18.2
45	37	82.2	28	62.2	21	46.7	17	37.8	13	28.9	11	24.4	10	22.2	9	20.0
46	38	82.6	29	63.0	22	47.8	17	37.0	13	28.3	12	26.1	9	19.6	9	19.6
47	39	83.0	30	63.8	23	48.9	17	36.2	13	27.7	11	23.4	9	19.1	8	17.0
48	40	83.3	31	64.6	23	47.9	18	37.5	14	29.2	14	29.2	10	20.8	9	18.8
49	41	83.7	31	63.3	26	53.1	18	36.7	15	30.6	12	24.5	10	20.4	9	18.4
50	42	84.0	32	64.0	24	48.0	19	38.0	15	30.0	14	28.0	10	20.0	10	20.0

Table 3 Various longest periodic spaced seeds of the maximum weight for a given number of mismatches n_m and weight w.

n_m	w	n_r	Longest spaced seed
2	16	32	10111001011100101110010111
2	20	39	101110010111001011100101110010111
2	24	44	11111100110101111110011010111111
2	28	51	110111110001101111100011011111000011011111
2	32	56	11111011100101111101110010111110111001011111
2	36	62	101111101110010111110111001011111011100101111110111
2	40	68	1111101110010111110111001011111011100101111101110010111
2	44	73	11111011100101111101110010111110111001011111011100101111110111
2	48	80	111110111001011111011100101111101110010111110111001011111011100101111
2	52	85	1111101110010111110111001011111011100101111101110010111110111001011111011
2	56	91	1110110111111111000111011011111111100011101101111111110001110110111111111
3	16	41	10011010111100010011010111
3	20	48	11110001001101011110001001101011111
3	24	56	100110101111000100110101111000100110101111
3	28	63	111100010011010111100010011010111100010011010111
3	32	71	1001101011110001001101011110001001101011110001001101011111
3	36	78	111100010011010111100010011010111100010011010111100010011010111
3	40	86	10011010111100010011010111100010011010111100010011010111100010011010111
3	44	93	111100010011010111100010011010111100010011010111100010011010111100010011010111
4	16	54	11110010000001000111100100000010001111
4	20	64	11011101000000100001101110100000010000011011101
4	24	70	111011111001001100001011010100011011111
4	28	79	1100111110001101110101000010010110011111000110111
4	32	88	10010110011111000110111010100001001011001111100011011101 01
4	36	96	111110001101110101000010010110011111000110111010100001001011001111
4	40	101	11101111100100110000101101010001110111110010011000010110101000011011111
4	44	110	1100111110001101110101000010010110011111000110111010100001001011001111100011011
5	16	62	11100110101000000100100000110011010 1
5	20	76	11000110101111100000000000000000011000110101111
5	24	86	11100110101000000100100000111001101010000001001000000111001101
6	16	75	10011010111100000000000000000000100110101111
6	20	91	10000111000000100000110000101000011100000010000011000001010000111
6	24	102	11001101010111111000000000000000000000000000011001101010111111
7	16	82	10000010100010001010101000000010000010100010001010101
7	20	96	10101010000000100000101000100010101010000000100000101000100010101010 1
8	16	96	100000100100100000000100000100100100000000100000100100100000000100000100100 1
9	16	105	10011010111100000000000000000000000000000000000000100110101111

Table 4 Blocks for spaced seeds ($n_m = 2$).

w	n_r	Periodic block	n_b	n_d
16	32	1011100	3	5
24	44	1111110011010	2	6
32	56	1111101110010	3	5
40	68	1111101110010	4	4
48	80	1111101110010	5	3
56	91	1110110111111111000	3	16
64	102	11111111110001110110	4	8
72	112	111111111010111100110	4	8
80	123	11011111111010111100	4	19
88	133	1111111110101111001 10	5	8
96	144	11011111111010111100	5	19

Table 5 Blocks for spaced seeds ($n_m = 6$).

w	n_r	Periodic block	n_b	n_d
16	75	10011010111100000000000000000000	1	12
24	102	110011010101111110000000000000000000000000	1	17
32	133	10000011000010100001111000000	3	22
40	161	100000110000101000011100000 0	4	22
48	188	110011010101111110000000000000000000000000000	3	17
56	213	11110000001000001100001010000011000100100 0	4	4
64	239	11000001100001010000111000011100000010000 0	4	30
72	266	10000111000011100000010000011000001100001 0	5	15
80	295	11000001100001010000111000011100000010000 0	6	2
88	316	11000010100001110000111000000100000110000 0	6	23
96	343	10001111000000100000110000101000001100010 0	7	8

Table 6 Examples of most popular spaced seeds.

[28]	Diamond
D_1	111101011101111
D_2	111011001100101111
D_3	11110010010100010011111
D_4	11110010100001001001011 1
[29]	BFAST
B_1	1111111111111111111111
B_2	11111011101110101001010111011111
B_3	10111101011010010110000110100011111111
B_4	10111001101001100100111101010001011111
B_5	11111011011101111011111111
B_6	11111110010100100010111101110111
B_7	11110101110010100010101101010111111
B_8	11110110101101100110000010110100101110 1
B_9	11110110100010001101011001011001101001 11
B_{10}	11110100101101101011100101101110 11
[12]	PatternHunter
P_1	111010010100110111
[21]	PatternHunter II
P_2	111100110010100001011
P_3	110100001100010101111
P_4	1110111010001111
[30]	
S_1	11111111110001110110010010011101001110001010010100001010011000010111100000001 1
S_2	11110111111011001111000110101100111010110000001110100011010010100111110011
S_3	11101101010011010110010010010101010110001
S_4	11111111111111110001111111000011001100011100111100001110000000111101110000011
S_5	11000001010111101001000000001101111111111000110000101111100101110111010110001 01
S_6	11100010000111101101011000001000100001110101101101101111010110100010010010 11
[31]	rasbhari
R_1	11110111100110101111101010110 11
R_2	11101010111011001101001111111111
R_3	11111101011010111001110110011111
[32]	Quadratic Residues
Q_1	101100001
Q_2	1011000010101111001

Table 7 Minimum lengths of reads required to achieve full sensitivity for a given number n_m of mismatches. Minimum lengths found with PerFSeeB approach are in bold. For the found lengths the corresponding weights of seeds generated with the proposed approach are in parentheses.

Seed	w	$n_m = 2$	$n_m = 3$	$n_m = 4$	$n_m = 5$	$n_m = 6$	$n_m = 7$
Q_1	4	14 (5)	16 (4)	21 (5)	24 (4)	28 (4)	30 (4)
best		**11**	**14**	**17**	**20**	**23**	**26**
Q_2	10	28 (13)	37 (14)	44 (12)	52	58	65
best		**22**	**29**	**37**	**44**	**52**	**58**
P_1	11	27 (12)	40 (15)	46 (13)	51	62	68
P_2		32 (16)	37 (14)	50 (14)	58	63	72
P_3		31 (15)	37 (14)	48 (14)	55	62	70
P_4		28 (13)	38 (15)	44 (12)	52	60	68
best		**23**	**31**	**40**	**46**	**56**	**62**
D_1	12	33 (16)	38 (15)	51 (15)	56	69	74
D_2		31 (15)	41 (16)	49 (14)	58	67	76
D_3		30 (15)	39 (15)	55 (16)	60	66	77
D_4		33 (16)	40 (15)	56 (17)	63 (16)	68 (17)	76
best		**25**	**33**	**43**	**50**	**59**	**66**
S_3	20	51 (28)	76 (34)	85 (31)	95 (26)	114 (27)	
best		**39**	**48**	**64**	**76**	**91**	
B_1	22	66 (38)	88 (40)	110 (44)	132 (39)	154 (38)	
B_2		49 (27)	68 (31)	80 (28)	94 (26)	111 (25)	
B_3		55 (31)	82 (38)	90 (32)	102 (28)	127 (31)	
B_4		51 (28)	70 (31)	88 (32)	97 (27)	112 (26)	
B_5		54 (30)	65 (29)	82 (29)	94 (26)	110 (25)	
B_6		49 (27)	76 (34)	84 (30)	97 (27)	119 (28)	
B_7		54 (30)	72 (32)	89 (32)	100 (27)	116 (27)	
B_8		58 (33)	68 (31)	91 (32)	101 (27)	117 (27)	
B_9		54 (30)	69 (31)	90 (32)	97 (27)	111 (25)	
B_{10}		55 (31)	73 (32)	88 (32)	102 (28)	113 (26)	
R_1		54 (30)	68 (31)	85 (31)	99 (27)	112 (26)	
R_2		54 (30)	74 (32)	86 (31)	98 (27)	117 (27)	
R_3		50 (27)	70 (31)	81 (29)	97 (27)	114 (27)	
best		**42**	**52**	**68**	**82**	**97**	
S_1	39	98 (61)	121 (58)	160 (69)	176 (54)		
best		**67**	**83**	**100**	**132**		
S_6	40	107 (68)	131 (65)	163 (72)	179 (54)		
best		**68**	**86**	**101**	**134**		
S_5	42	100 (62)	149 (77)	172 (76)	189 (59)		
best		**70**	**91**	**105**	**138**		
S_2	43	96 (60)	123 (60)	163 (72)	180 (55)		
best		**72**	**92**	**107**	**140**		
S_4	46	115 (74)	151 (78)	187 (82)	223 (71)		
best		**78**	**97**	**114**	**153**		

Figures

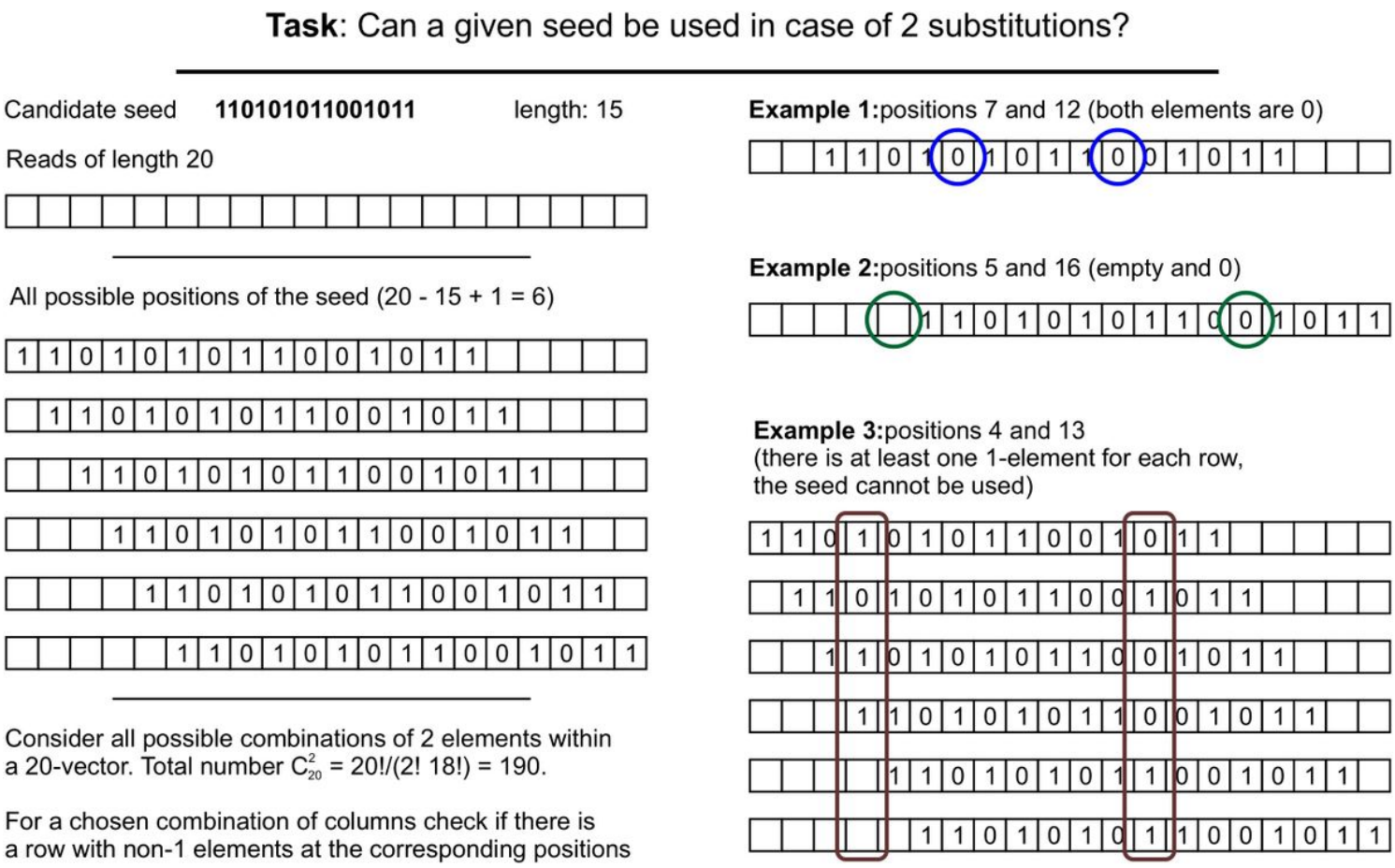

Figure 1

Checking if a seed is a full sensitivity one for a given length of a read and number of mismatches.

$V \mid R_p = V$?

R_1 = **110101011001011000001111**
R_2 = **011010101100101100001111**
R_3 = **001101010110010110001111**
R_4 = **000110101011001011001111**
R_5 = **000011010101100101101111**
R_6 = **000001101010110010111111**

V_1 = **111111011110111111111111**

$V_1 \mid R_1$ = **111111011111111111111111**
$V_1 \mid R_2$ = **111111111110111111111111**
$V_1 \mid R_3$ = **111111011110111111111111**
$V_1 \mid R_4$ = **111111111111111111111111**
$V_1 \mid R_5$ = **111111011111111111111111**
$V_1 \mid R_6$ = **111111111110111111111111**

V_3 = **111011111111011111111111**

$V_3 \mid R_1$ = **111111111111011111111111**
$V_3 \mid R_2$ = **111011111111111111111111**
$V_3 \mid R_3$ = **111111111111011111111111**
$V_3 \mid R_4$ = **111111111111011111111111**
$V_3 \mid R_5$ = **111011111101111111111111**
$V_3 \mid R_6$ = **111011111111111111111111**

Figure 2

Seed validation based on padded rows. When the requirement is met, the row is in green colour, otherwise, in red colour.

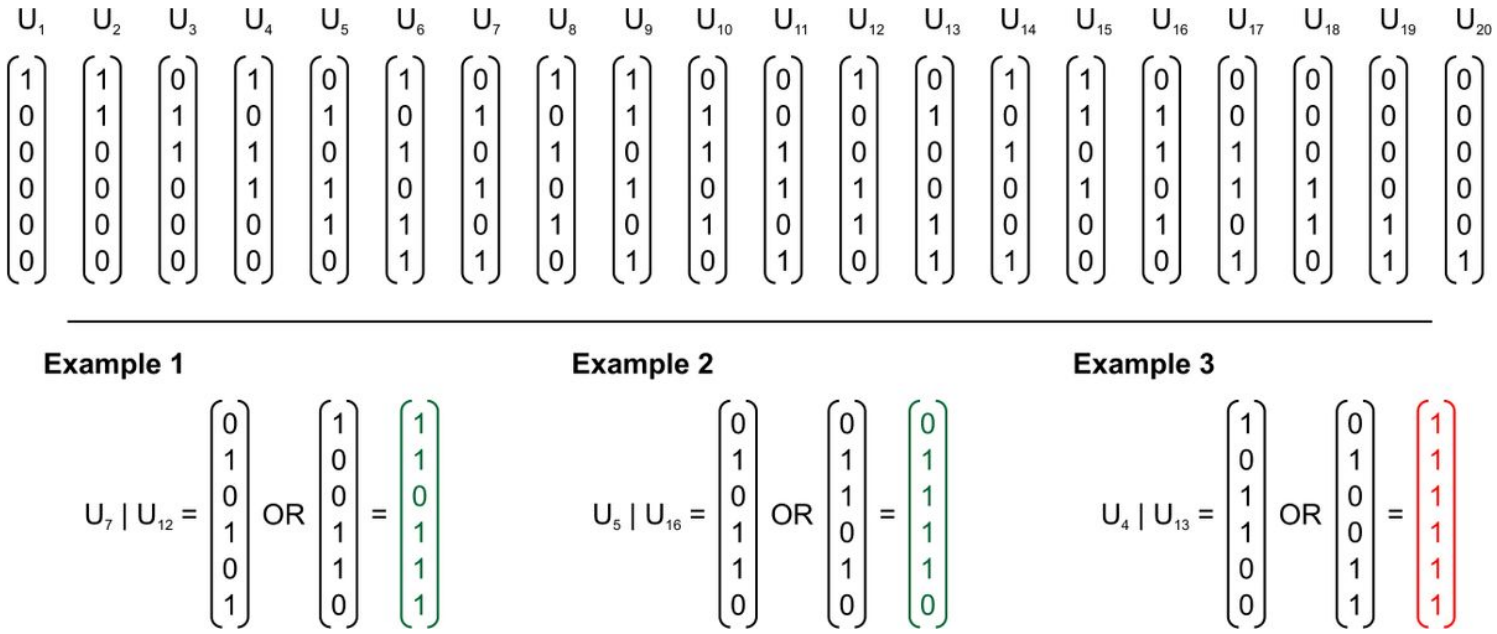

Figure 3

Seed validation based on columns. When the requirement is met, the row is in green colour, otherwise, in red colour.

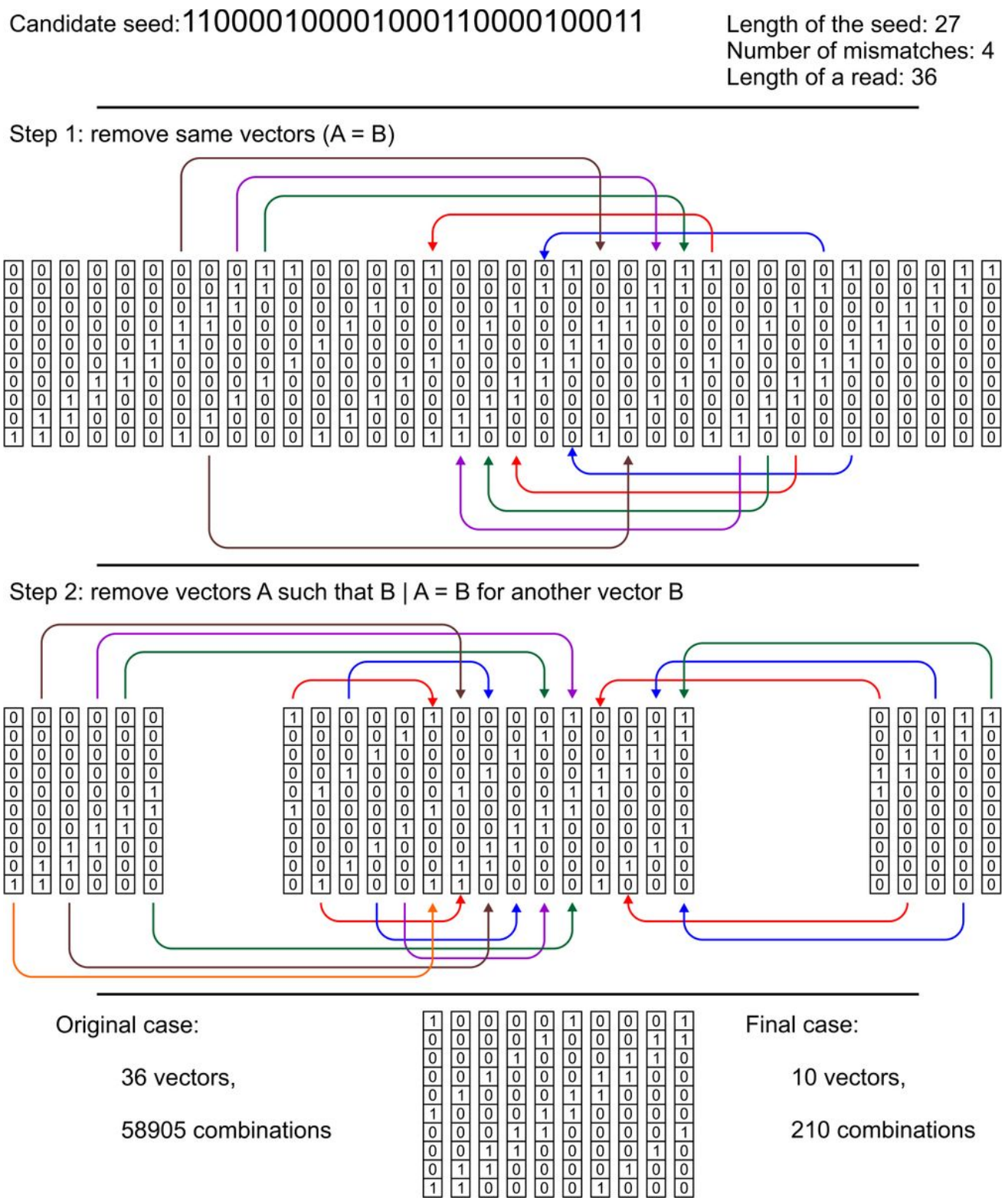

Figure 4

Reducing the number of combinations when checking seed's validity.

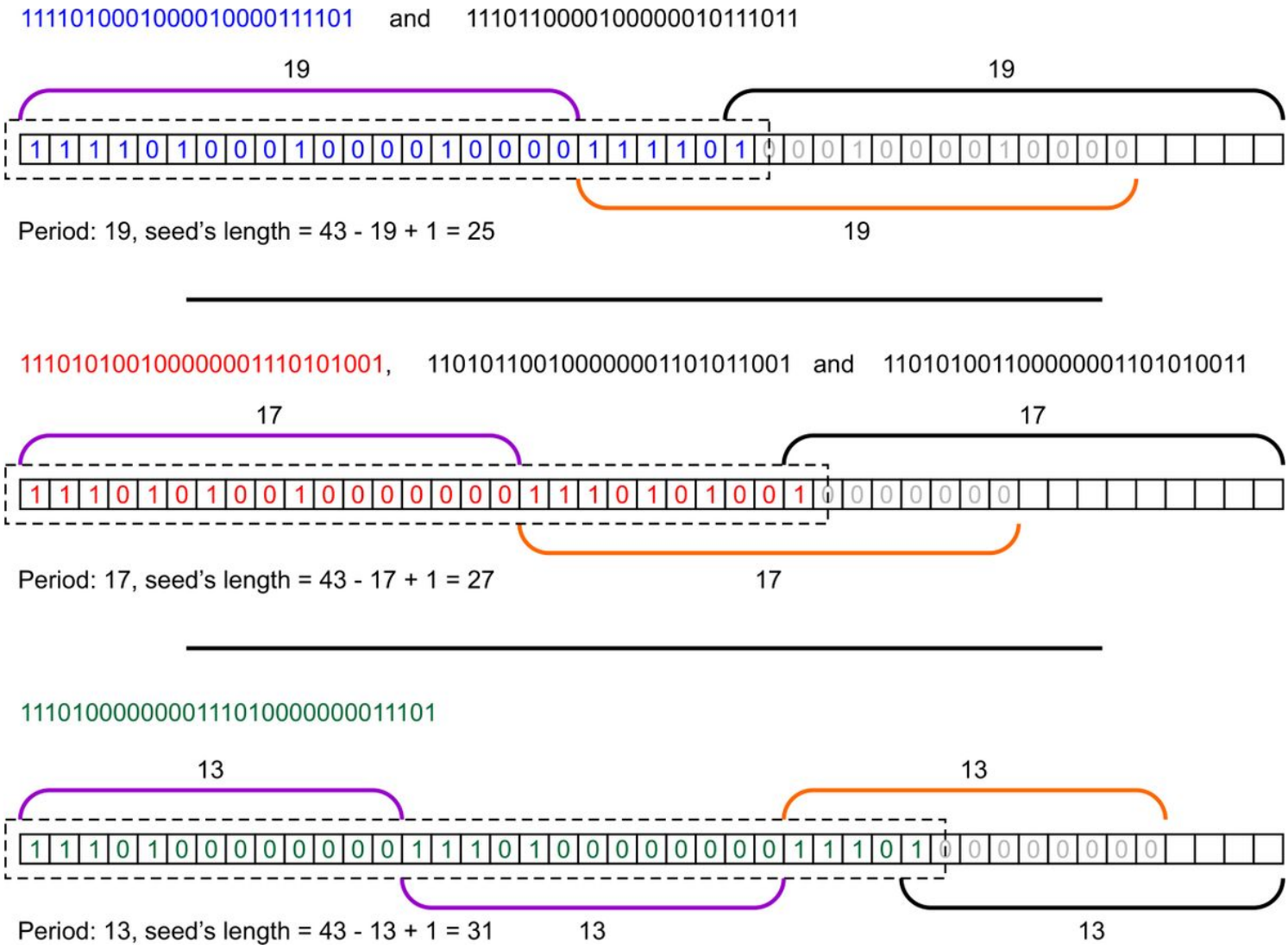

Figure 5

All seeds of maximum weight (12) found for reads of length 43, number of mismatches is 4. Flipped seeds are not shown.

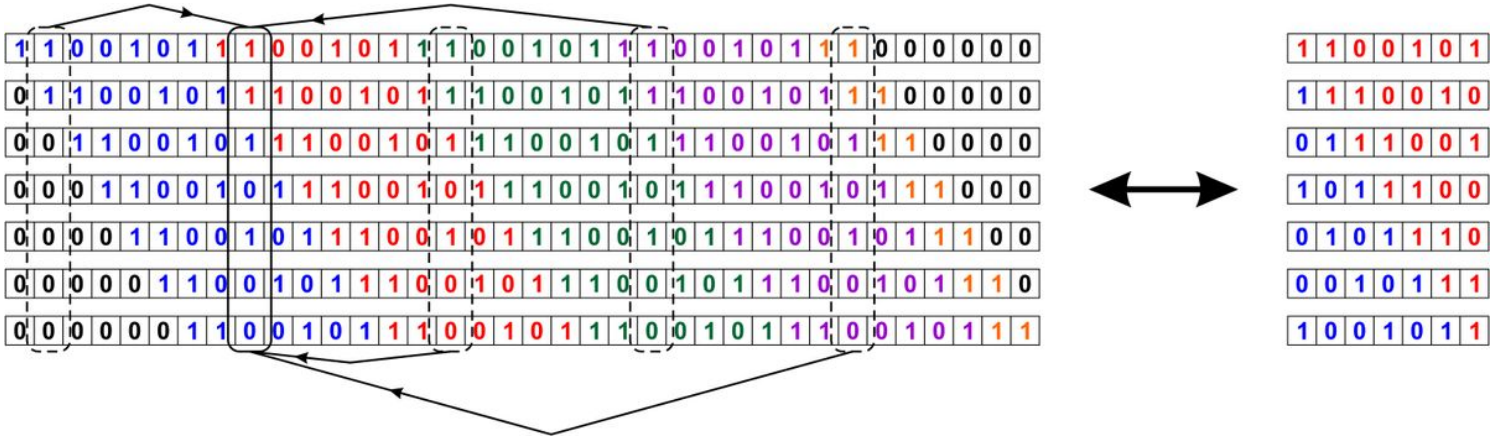

Figure 6

A seed is valid only when its periodic block is valid.

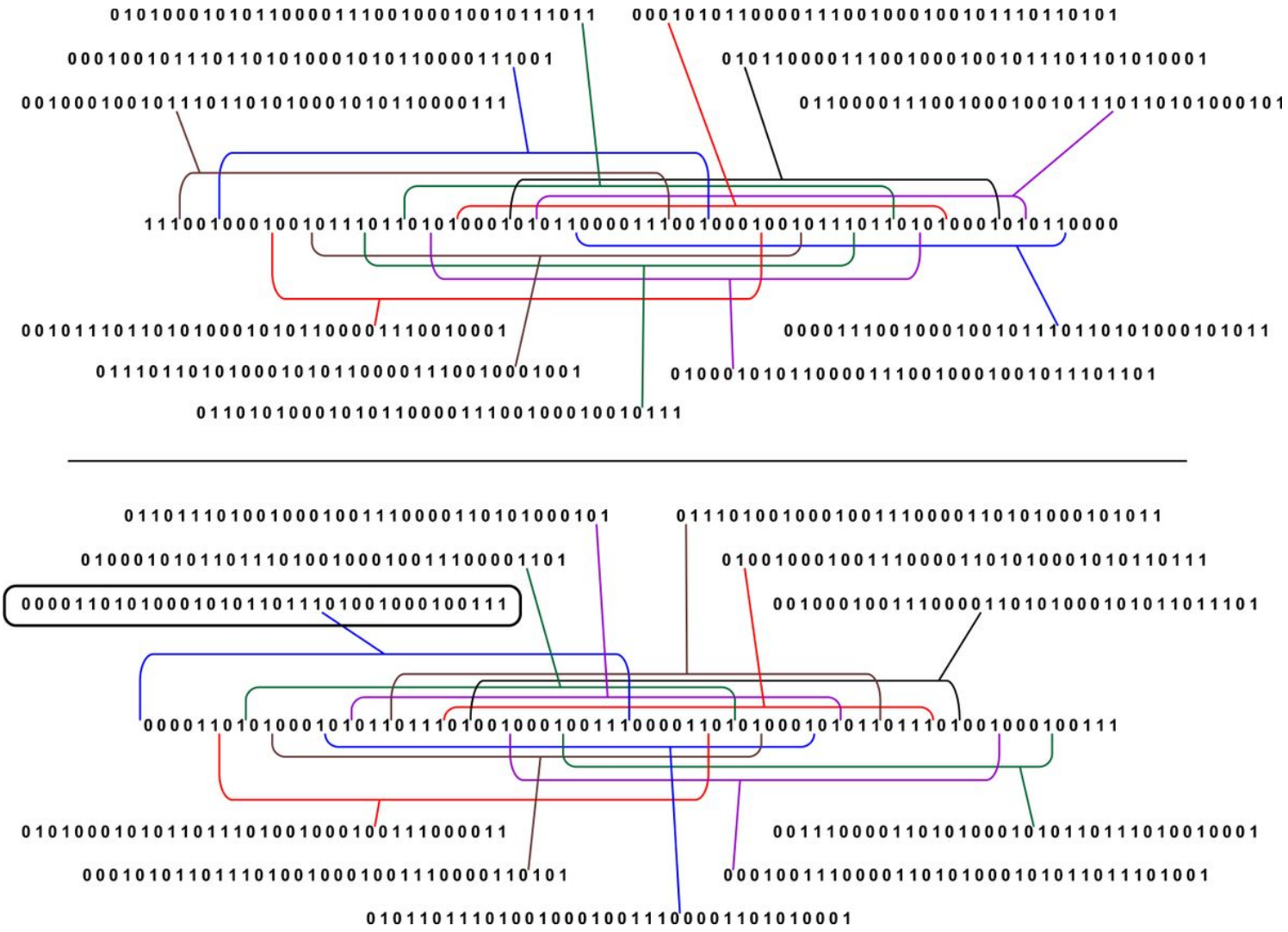

Figure 7

Equivalent periodic blocks generated for block 10111010010001001110000110101000101010. Only blocks with the first 0-element and last 1-element are shown. The bottom part is for the reverse block. The circled block is the one we choose from the group.

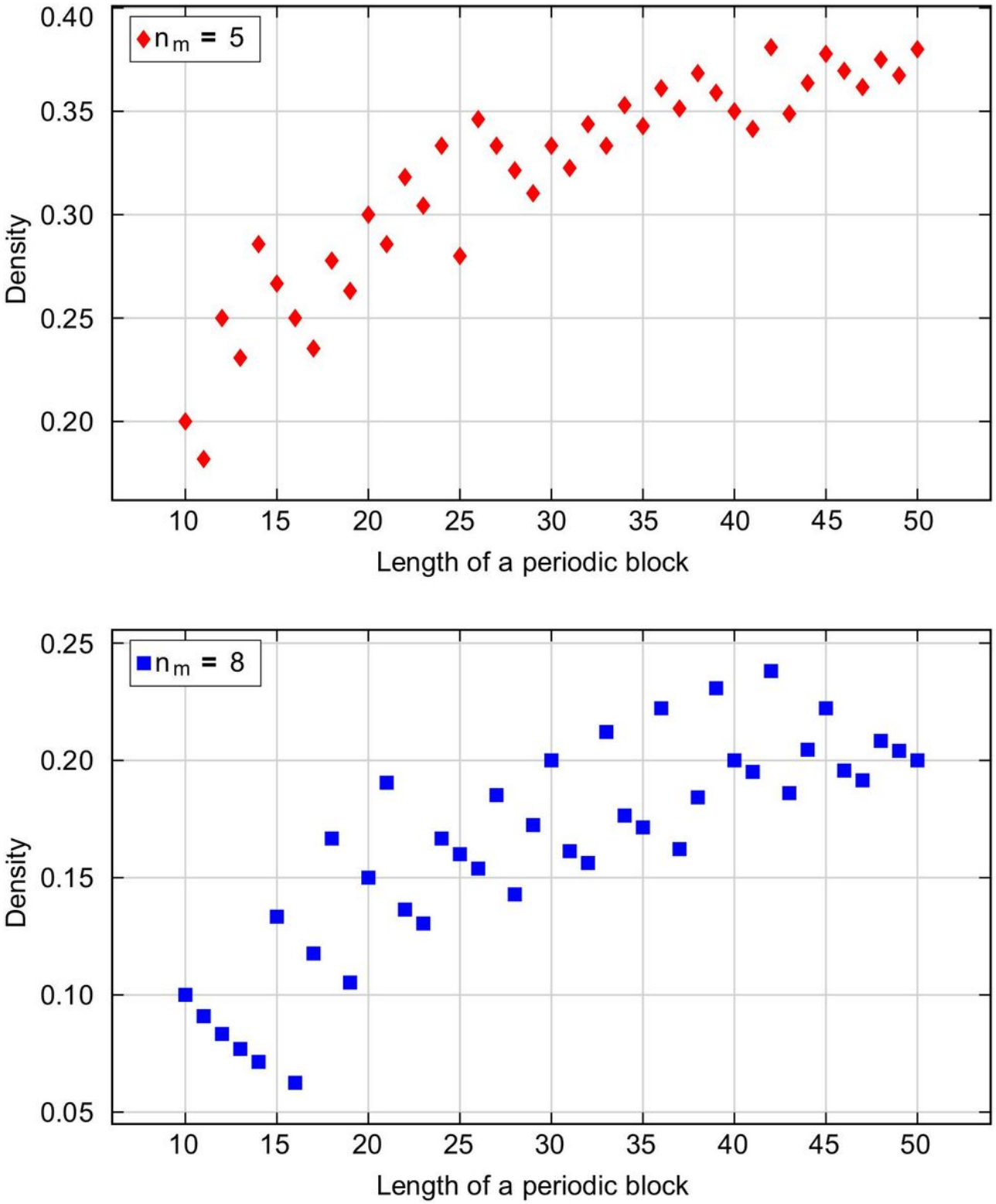

Figure 8

Maximum numbers of 1-elements per length of periodic blocks, nm = 5 and nm = 8.

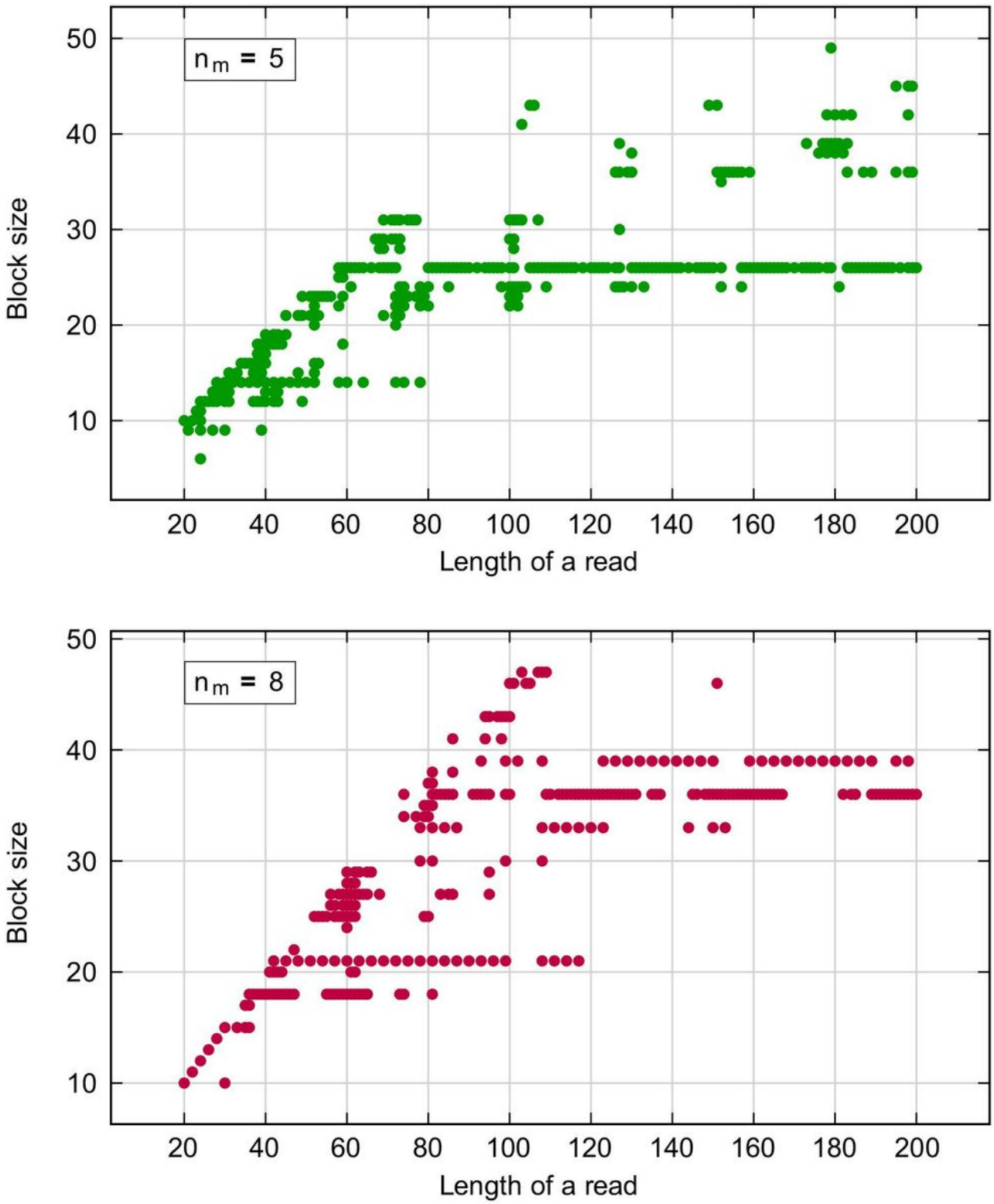

Figure 9

Maximum numbers of 1-elements per length of periodic blocks, nm = 5 and nm = 8.

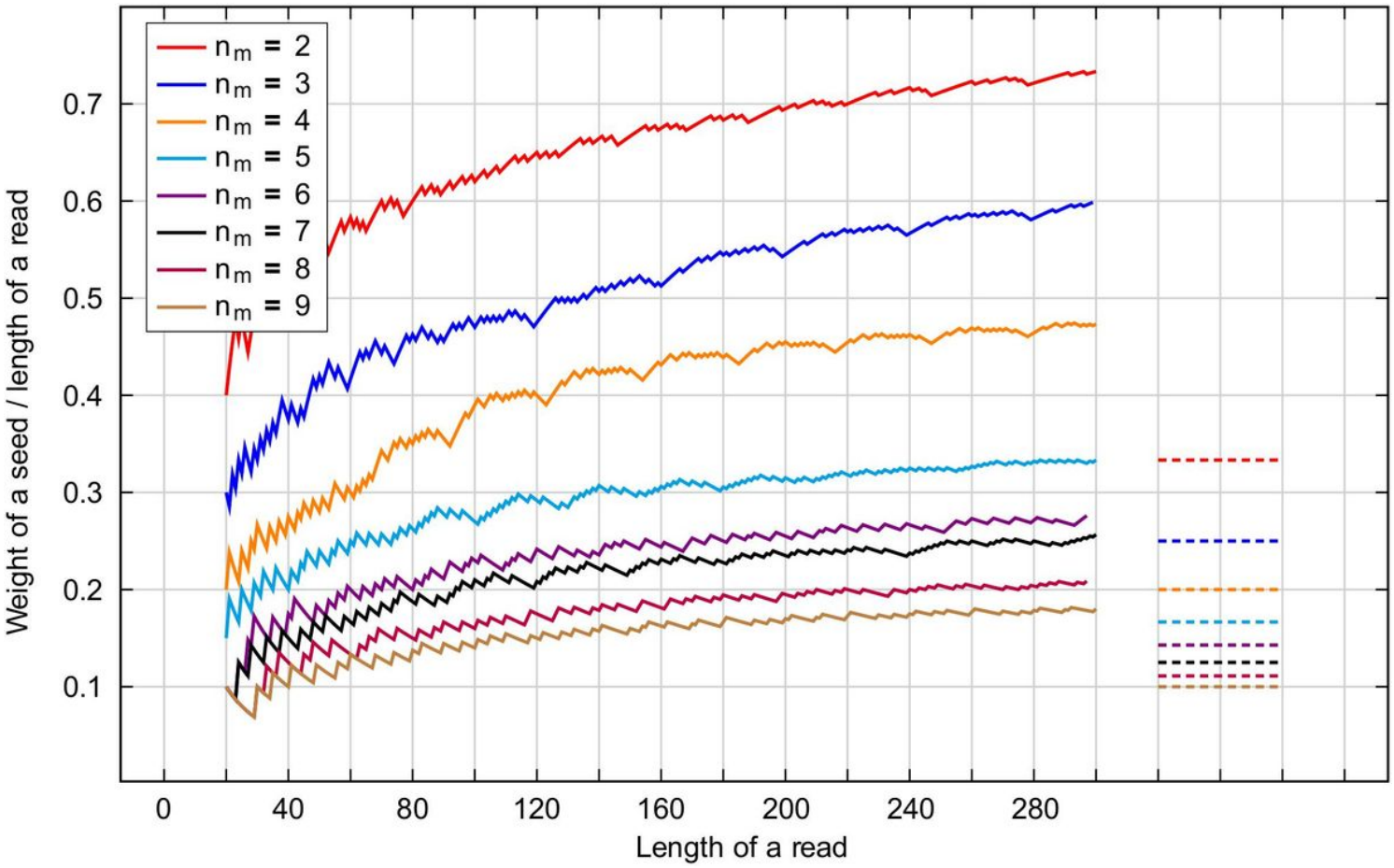

Figure 10

Weights of the best spaced seeds per reads' lengths (solid lines). Maximum ratios for contiguous seeds (dashed lines).

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- bestSeeds.zip
- supSeed.pdf