Exact Distribution of a Spaced Seed Statistic for DNA Homology Detection

Gary Benson[1⋆] and Denise Y.F. Mak[2]

[1] Departments of Computer Science, Biology, Program in Bioinformatics, Boston University, Boston, MA 02215; gbenson@bu.edu
[2] Graduate Program in Bioinformatics, Boston University, Boston, MA 02215; dyfmak@bu.edu

Abstract. Let a *seed*, S, be a string from the alphabet $\{1, *\}$, of arbitrary length k, which starts and ends with a 1. For example, $S = 11*1$. S *occurs* in a binary string T at position h if the length k substring of T ending at position h contains a 1 in every position where there is a 1 in S. We say that the 1s at the corresponding positions in T are *covered*. We are interested in calculating the probability distribution for the *number* of 1s covered by a seed S in an iid Bernoulli string of length n with probability of 1 equal to p. We refer to this new probability distribution as C_{nSp}, for *covered*, with S being the seed. We present an efficient method to calculate this distribution *exactly*. Covered 1s represent matching positions detected in DNA sequences when using multiple hits of a spaced seed. Knowledge of the distribution provides a statistical threshold for distinguishing true homologies from randomly matching sequences.

1 Introduction

Let a *seed*, S, be a string from the alphabet $\{1, *\}$, of arbitrary length k, which starts and ends with a 1. For example, $S = 11*1$ with length, $k = 4$. S *occurs* in a binary string T at position h if the length l substring of T ending at position h contains a 1 in every position where there is a 1 in S. We say that the 1s at the corresponding positions in T are *covered*.

More formally, let $S = s_1, \ldots, s_k$, and $T = t_1, \ldots, t_n$. Then S occurs at position h in T if for every $i \in [1..k]$ if $s_i = 1$ then $t_{h-k+i} = 1$. A 1 at t_j is *covered* by S if there exists some h and i such that S occurs at position h in T, $s_i = 1$, and $j = h - k + i$. For example, if $T = 101101111$ and $S = 11*1$, then S occurs in T

⋆ This research was supported in part by NSF grant IIS-0612153 and NIH grant 1 R01 GM072084.

at positions 6 and 9. Five 1s in T are covered, those at positions 3, 4, 6, 7, and 9:

```
1 2 3 4 5 6 7 8 9      1 2 3 4 5 6 7 8 9
1 0 1 1 0 1 1 1 1      1 0 1 1 0 1 1 1 1
    1 1 * 1                    1 1 * 1
```

We are interested in calculating the probability distribution for the *number* of 1s covered by a seed S in binary strings of length n randomly generated by an iid (independent and identically distributed) Bernoulli process with p equal to the probability of 1 (success). For a *contiguous* seed (consisting entirely of 1s, for example $S = 111$), this reduces to the R_{nkp} probability distribution for the number of 1s which occur in *runs* of length k or longer (k is the number of 1s in S) in an iid Bernoulli string of length n [4, 8]. For an example of these distributions, see figure 3. We will refer to the new probability distribution as C_{nSp}, for *covered*, with S being the seed.

The R_{nkp} distribution has proven useful as a criterion for detecting tandem and inverted repeats in DNA sequences [2, 11]. In those applications, the binary string (here called the *representative string*) represents a gapless alignment between two homologous sequences (homologous means that the sequences have a common evolutionary ancestor) which have undergone random point mutations. As in the example below, a 1 represents a position where the two sequences remain the same and a 0 represents a position where they differ.

```
Sequence 1:            A C G T G C G T A A T T T C G
Sequence 2:            A C C A G C T T T A T T C C G
Representative string: 1 1 0 0 1 1 0 1 0 1 1 1 0 1 1
```

The parameter p models the expected probability that any two aligned positions (*i.e.* in a column) remain the same. A typical assumption might be that 80% of the positions are expected to remain the same (*i.e.* $p = 0.8$).

Algorithms that seek to detect mutated homologous sequences typically start by finding small matching words. A match is called a *hit* and each hit is tested to see if it is part of an extended region of homology. The *seed*, S, describes the shape of the small matching words. If $S = 111$, then in the example sequences above, the nucleotide triplet ATT is considered a hit because it appears in both sequences and shows up as 111 in the representative string. Hit testing for extended homology uses dynamic programming alignment or some approximation to alignment and is usually the most costly step in homology detection programs.

While searching with seeds is an effective method, it suffers from a serious drawback. A short seed (in the example, three matching letters) will occur frequently at random, especially in long genomic sequences, and as a result, too much time

is spent pointlessly extending hits. One solution to this problem involves using a longer seed (say 11 or 12 letters), which significantly reduces the number of random hits, but also reduces the sensitivity of the detection program because many homologous sequences do not contain long unmutated stretches of nucleotides.

Another solution uses smaller seeds, but requires multiple seed hits clustered closely together. This is where the R_{nkp} distribution comes in. It provides an estimate for the expected number of matching aligned positions that participate in those hits. This permits the setting of a reasonable threshhold criterion to distinguish between clustered hits occurring randomly in unrelated sequences and clustered hits occurring in homologous sequences. Fortuitously, the R_{nkp} distribution for homologous sequences of distinct sizes can be obtained merely by varying the parameter n.

Spaced seeds. A few years ago, interest turned to spaced seeds [9, 5, 7, 10] which increase single hit sensitivity without simultaneously increasing the number of random hits. While a contiguous seed, such as $S = 111$, is a short word, a spaced seed consists of a number of explicit positions which must match separated by "don't care" positions which may or may not match. The seed $11 * 1$ at the beginning of this section is a spaced seed. The 1s indicate matching positions and the * indicates a "don't care" position which may align with either a 1 or 0 (match or mismatch) in the representative string. In the aligned sequences above, the string pairs (GCGT, GCTT) and (TTTC,TTCC) are hits for the seed $11 * 1$ because in each pair, the first, second, and fourth letters match.

Spaced seeds have been studied primarily in terms of a single hit paradigm, in part, because there was no straightforward way to compute the distribution C_{nSp}. In this paper, we present an efficient method to calculate this distribution *exactly*, which also yields an efficient method for the exact computation of the R_{nkp} distribution. (On the way to this result, we developed a less efficient method for calculating the distribution, presented in [3].)

Importantly, our method is *probability independent* in that the calculation yields a formula in terms of the probability parameter p, and does not require fixing the value of p in advance. Previous methods for computing R_{nkp} have all required that p be specified in advance, and in that case, if the model parameter p changes, then the computation must be run again. The probability independent method is based on *probability equivalence classes* which partition the set of representative strings. This method has previously been used in a different form by us to identify *all* optimal spaced seeds for single hit homology detection [10]. These seeds are optimal in the sense that they have the highest sensitivity of all seeds with equivalent random hit probability.

Our coverage concept is similar to coverage defined in [6], although that paper deals with *minimum* coverage using a *fixed number* of spaced seeds, rather than, as here, a probability distribution on coverage for an unconstrained number of seeds. A dynamic programming algorithm is presented in [6] (which is similar

in some ways to [3]) for computing minimum number of spaced *seeds* which hit alignments in which the number of mismatches is fixed. This minimum number of seeds is then used to compute a minimum *coverage* with a branch and bound algorithm.

The remainder of the paper is organized as follows. In section 2 we show how to efficiently calculate C_{nSp}, first in the probability independent manner, and then in the probability dependent manner. We give time and space complexities for the calculation on spaced and contiguous seeds. In section 3 we show how to modify the calculation to restrict the distribution to binary strings that represent confirmable alignments in the hit extension phase of a homology detection program. Finally in section 4 we present several graphs illustrating the distribution.

2 Methods

Probability equivalence classes. Binary strings (representative strings) which have the same probability, when the success probability parameter p is specified, belong to the same probability equivalence class. For strings of length n, there are $n+1$ classes, corresponding to the possible number of 1s, *i.e.* $0, 1, 2, \ldots, n$. For example, when n=100, each string with 70 1s, *no matter how arranged*, has probability $p^{70}(1-p)^{30}$.

For the purposes of this work, we further divide the equivalence classes. Let

$$X[n, i, j, b]$$

be the *number* of binary strings with i ones, j of which are covered by a given seed S when considering all strings of length n such that the strings all begin with the same prefix, represented by X, and all have the same initial covered positions, as indicated by b. For example if X represents 111111, and $b = 10101$, then *all* the strings counted in $X[n, i, j, b]$ start with 11111 and all have positions 1, 3, and 5 covered and positions 2 and 4 uncovered.

Modified Aho-Corasick tree data structure. Our data structure for counting strings in the equivalence classes is a simple modification of the Aho-Corasick tree (AC tree [1]) built on the patterns of the seed S. A *pattern* of a seed is any string of 1's and 0's which is obtained by replacing each wildcard in the seed by either a 1 or 0. For a seed with r stars, there are 2^r patterns.

The AC tree is modified as follows. Refer to figure 1. In a normal AC tree, one unlabeled directed fail edge exits every node. We dispense with these edges and for each node which now has less than two out edges, referred to as a *fail node*, we add one or two new directed *fail edges* to bring the total number of out edges to two. Each new fail edge is labeled with either a 0 or a 1 such that the two out edges at each node have distinct labels. Each fail edge terminates on a *fail-to*

node determined as follows. Let the *node string* for node X be the concatenation of edge labels from the root to X. The node string has length depth(X), where the root has depth zero. If X is a fail node and the new fail edge out of X has label σ, then the fail-to node, Y, is the node whose node string is the longest proper suffix of the node string for X concatenated with σ.

Fail nodes and fail-to nodes are here called *marked nodes*. The X in the probability equivalence class notation above is a marked node in the AC tree. The string represented by X is the node string for X. b is a binary string of length $\min(\text{depth}(X) - 1, 0)$ which indicates a possible pattern of initial covered positions in a string that starts with the node string for X. For example, in figure 1, the node string for node X is 111111, and one possible pattern of covered positions is $b = 10101$.

The root, R, of the AC tree is a special node. It's probability equivalence class holds all the information necessary to determine C_{nSp}. If we fix n, p, S, and j (and realize that b for the root is the empty string b_ϵ), then the combined probability of all strings which have j covered 1s is

$$\sum_{i=j}^{n} (R[n, i, j, b_\epsilon])\, p^i (1-p)^{n-i}. \tag{1}$$

Allowing j to vary and summing the terms as above gives us C_{nSp} directly.

2.1 Recurrence formula

Recurrence idea. The calculation of $X[n, i, j, b]$ for any marked node X is accomplished using a dynamic programming recurrence. We assume that the counts in the equivalence classes for each marked node for string lengths from 1 to $n-1$ have been computed. Each equivalence class saves some additional information about the extreme left ends of its strings (the prefix string represented by X and the coverage pattern by b). Then, inductively, computing an equivalence class for strings of length n consists of following out edges from node X in the AC tree, and accumulating the counts in the equivalence classes down those edges. The effect is to substitute the node string for X in place of an existing prefix, possibly resulting in some additional covered bits (figure 1).

Computing the equivalence class counts. Assume that the recurrence is being computed for marked node X. It has two out edges and depending on X, each out edge is either a fail edge or an edge included in the original AC tree as follows:

- two fail edges (X is a leaf)
- one fail edge and one edge included in the AC tree (X is an internal node preceding a 1 in the seed)

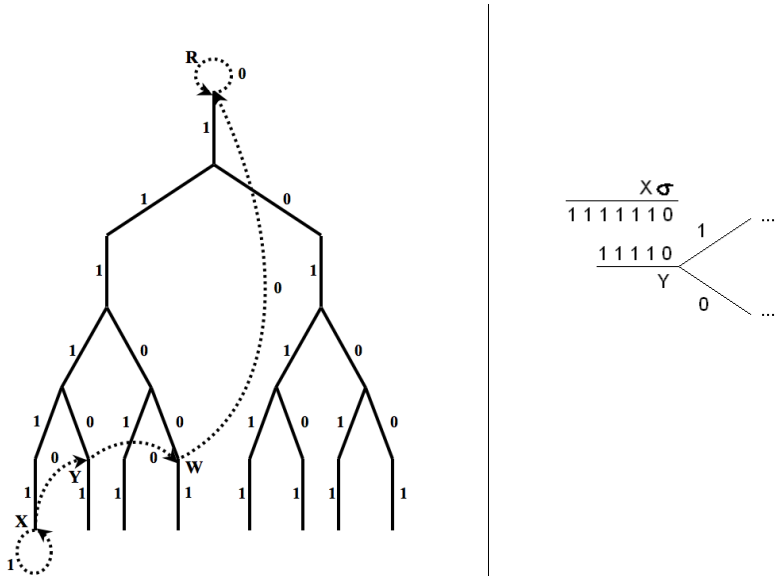


Fig. 1. Left: Modified Aho-Corasick tree for seed 1*1**1. Fail edges are shown for nodes X, Y, W, and R (labeled with $\dashrightarrow$.) Right: Idea for recursion. Node string to X followed by label σ of edge from X to Y replaces node string to Y in one set of strings through Y.

- two edges included in the AC tree (X is an internal node preceding a * in the seed)

The recurrence formula for X depends on its node type. For brevity, we show the formula for the second category above. The fail edge yields a summation using information stored at the fail-to node Y. The AC tree edge yields a summation using information stored at one or more marked descendant nodes $Z_1, Z_2, \ldots$ (the first marked descendant nodes down any path below X in the AC tree edge). An explanation for the notation is given after the formula. Note that the base cases are omitted due to space limitations.

$$\forall i \in [0,n], j \in [0,i], b_x \in B_x$$

$$\begin{aligned} X[n,i,j,b_x] = & \sum_{b_y \in BF_y(b_x)} Y[n - dd_{xy}, i - od_{xy}, j - cv(b_y), b_y] \\ & + \\ & \sum_{Z_l,\ l \in \{1,2,\ldots\}} \left(\sum_{b_{z_l} \in BD_{z_l}(b_x)} Z_l[n,i,j,b_{z_l}] \right). \end{aligned} \tag{2}$$

Depth difference: The difference in recursion level between the nodes as determined by the AC tree.

$$dd_{xy} = \text{depth}(X) + 1 - \text{depth}(\text{Y}).$$

For example, in figure 1 $dd_{xy} = 2$. The 1 in the formula accounts for the fail edge from X to Y.

Ones difference: The increase in 1s in the strings of the equivalence class when substituting the node string for X followed by the fail edge label ($\text{depth}(X) + 1$ bits) for the node string to Y (the leftmost $\text{depth}(Y)$ bits).

$$od_{xy} = \text{ones}(X) + I - \text{ones}(Y) \quad \text{where } I = \begin{cases} 1 & \text{if edge } \boldsymbol{XY} \text{ is labeled 1} \\ 0 & \text{if edge } \boldsymbol{XY} \text{ is labeled 0} \end{cases}$$

where $\text{ones}(X)$ is the number of 1s in X.

Covered difference: The increase in covered positions when replacing the leftmost bits as above. Note that there is only an increase if X is a leaf, otherwise there are no new covered positions. An example is shown on the left side of figure 2.

$$cv(b_y) = \begin{cases} \text{number of 1 bits in } ([b_s \text{ OR } b_y] \text{ XOR } b_y) & \text{if } X \text{ is a leaf} \\ 0 & \text{if } X \text{ is not a leaf} \end{cases}$$

where

- b_s is the bit string obtained from the seed S by replacing every star (*) by 0. These are the positions covered by an occurrence of the seed. For example, if $S = 1 * 1 * *1$, then $b_s = 101001$.
- in the OR and XOR operations, b_y is left filled with zeros so that it contains as many bits as b_s,

Compatible subsets of bit strings: In equation 2, B_x is the set of bit strings which indicate initial covered positions compatible with starting a string with the node string for X. The bit strings in B_x have length $\text{depth}(X) - 1$. If this value is negative, B_x only contains the empty string. For example, in figure 1 the compatible bit strings for node X are 10100, 10101, 10110, 10111, 11110 and 11111. For every marked node X, B_x can be determined by a recursive formula based on the modified AC tree. Details are omitted.

$BF_y(b_x)$ is the set of bit strings in B_y that can lead to bit string b_x when Y follows X along a fail edge (the F stands for fail edge). The method for constructing the set depends on whether or not X is a leaf. Also, in some cases, not all b_x can be achieved down a fail edge to a particular Y. An example for the case where X is a leaf is given on the right side of figure 2.

$$BF_y(b_x) = \phi \text{ if truncate-right}(\text{depth}(Y) - 1, (b_x \text{ XOR } b_s)) \text{ contains 1s and } X \text{ is a leaf.}$$

otherwise

$$BF_y(b_x) = \begin{cases} B_y \cap \{\text{pats-right}(\text{depth}(Y) - 1, (b_x \text{ OR* } b_s))\} & \text{if } X \text{ is a leaf} \\ B_y \cap \{\text{pats-right}(\text{depth}(Y) - 1, (b_x || *))\} & \text{if } X \text{ is not a leaf} \end{cases}$$

where

- ϕ is the empty set (not the same as the set containing the empty string),
- in the XOR and OR* operations, b_x is right filled with zeros so that it contains as many bits as b_s,
- in the OR* operation, if b_s is 1, the result is *, if b_s is 0, the result is the same as the bit in b_x,
- truncate-right(v, s) removes the rightmost v bits from s,
- pats-right(v, s) yields all patterns (bit strings) which can be generated from a string s over the alphabet $\{*, 0, 1\}$ by replacing each * by a 0 or a 1. Only the rightmost v bits are retained in each generated string.
- $\cap$ means intersection,
- $||$ means concatenate,

$BD_z(b_x)$ is the set of bit strings in B_z that can lead to bit string b_x when Z follows X along edges in the AC tree (the D stands for descendant).

$$BD_z(b_x) = B_z \cap \{\text{pats-right}(\text{depth}(Y) - 1, (b_x || *^{\text{depth}(Z) - \text{depth}(X)}))\}$$

where

- $*^{\text{depth}(Z) - \text{depth}(X)}$ is a string of stars of length depth$(Z)-$depth(X).

2.2 Time and space complexity

Theorem 1. *The C_{nSp} distribution for strings of length n and a seed S of length k with r stars can be computed in time $O(2^r(k - r + 1)n^3 B_{max})$, and space $O(2^r(k - r + 1)(k + 2)n^2\overline{B} \cdot B_{\max})$ where $\overline{B}$ is the average size of the set B_x for marked nodes X and B_{max} is the size of the largest such set.*

Although $\overline{B}$ and $B_{\max}$ are potentially as large as 2^{k-1}, (the b_x coverage bit strings are always 1 bit shorter than the depth of node X), in our calculations, we have found that they are typically small integers.

Corollary 1. *The R_{nkp} distribution for strings of length n and a seed S consisting of k 1s can be computed in time $O(kn^3 B_{\max})$ and space $O(2kn^2\overline{B} \cdot B_{\max})$.*

```
   101001  b_s                  101110  b_x right filled with zeros
OR 001010  b_y               OR* 101001  b_s
 ---------                      ---------
   101011                        *0*11*
XOR 001010  b_y                    *11*  rightmost 4 positions
 ---------                                (length of strings in B_y)
   100001
number of 1 bits = 2             {*11*} = {0110, 0111, 1110, 1111}
                                          and intersection with B_y
                                          yields {1111}
```

Fig. 2. Left: Example of $cv(b_y)$ calculation. $S = 1 * 1 * *1$, $b_y = 1010$, and X is a leaf. Then $b_s = 101001$ and b_y left filled with zeros is 001010. The number of 1 bits in 100001 is 2 so $cv(1010) = 2$, which means replacing the node string to Y with the node string to X followed by the label of the fail edge from X to Y yields two additional covered 1's. Right: Example of the calculation of the compatible subset $BF_y(b_x)$ when X is a leaf. $S = 1 * 1 * *1$, $b_x = 10111$, $B_y = \{0000, 0101, 1010, 1011, 1111\}$. *11* produces four strings, of which one string intersects with the set B_y.

2.3 Probability dependent recursion

The recursion can be made probability *dependent* which requires picking p first. The formulation is similar to that shown in equation 2, except the variable i (the number of 1s in the strings), is omitted since equivalence classes now only depend on the number of covered positions j. Also, the array elements hold probabilities rather than sequence counts. Formulas are omitted due to space limitations. The time and space complexities are reduced by a factor of n.

3 Restricting C_{nSp} to confirmable alignments

The calculations described above assume that alignments of homologous sequences can be represented by all possible bit strings. Yet, representative strings that contain too many 0s reflect homologies that *can not be confirmed* in the hit extension step of homology detection programs because there are too many mismatches. For example, "homologous sequences" in which only 30% of the positions match could never be confirmed because the alignment score would be too low. Although the probabilities of representative strings that contain many 0s are relatively small when p is set high, for example, $p = 0.8$, these probabilities nonetheless have a significant impact on the C_{nSp} distribution as we now show.

We recalculated the C_{nSp} distribution with a restriction that eliminates the contribution from strings with many 0s. Different approaches are possible. We chose to use the following. Define a representative string of length n, as *confirmable*

if for every suffix of the bit string, of length $\geq n_{\min}$, the fraction of 1s is $\geq f$. In the recurson arrays, this corresponds to the elements of $X[n, i, j, b_x]$ in which $i/n \geq f$. Therefore, to restrict our results to confirmable alignments, any element of an array with $n \geq n_{\min}$ that has a ratio $i/n < f$ is not used at higher recursion levels (and is therefore not computed). Note that the distribution C_{nSp} assumes that the total probability across all j is 1. Here we must normalize the distribution so that the total probability is the sum of the probabilities of the confirmable alignments.

4 Results

Figure 3(a) shows the exact probability distribution C_{nSp} for two seeds, 11111, and 1*11*11 for strings of length 100 and $p = 0.8$. These seeds have *equal weight*, that is, they have equal expected probability of random hits (determined by the number of 1s in the seed). As expected, the spaced seed which is more sensitive in the single hit paradigm shifts the probability distribution to the right. Note that for seed 11111, C_{nSp} is identical to the distribution R_{nkp}. The calculation for the spaced seed took 5 seconds on a 64-bit system, dual, dual core 2.8GHZ with 4GB RAM.

Figure 3(b) shows the cumulative probabilities for the same seeds shown in figure 3(a). For the purposes of DNA homology detection, the threshold for number of matching positions detected to trigger a hit extension is typically set at the 5% level of the distribution. Note the significant shift in this threshold to a higher value for the spaced seed (47.5) over the contiguous seed (38). Recall that the average number of 1s is 80 in these sequences.

Figure 4(a) shows the probability distribution C_{nSp} for the seed $1*11*11$ under several assumptions for the ratio $i/n \geq f$. Figure 4(b) shows the cumulative probabilites. Note again the shift to a higher threshold as the value of f increases.

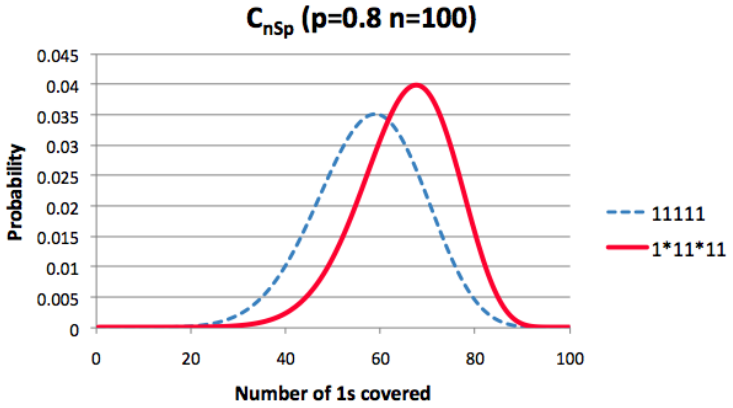
5 Conclusion

This paper presents a dynamic programming method for calculating the probability distribution C_{nSp} exactly. The distribution describes the expected number of 1s in a random bit string covered by occurrences of a spaced seed. The recurrence is built on a modified Aho-Corasick tree data structure. Two methods are presented. The first is probability independent, in that the time consuming steps are performed before setting the parameter p, the probability of a match between aligned positions in homologous sequences. Once completed, different values of p can be selected and the calculation of C_{nSp} proceeds quickly. The second method is probability dependent and faster overall, but only applies to a single value of p. An example shows the improvement in coverage of a spaced

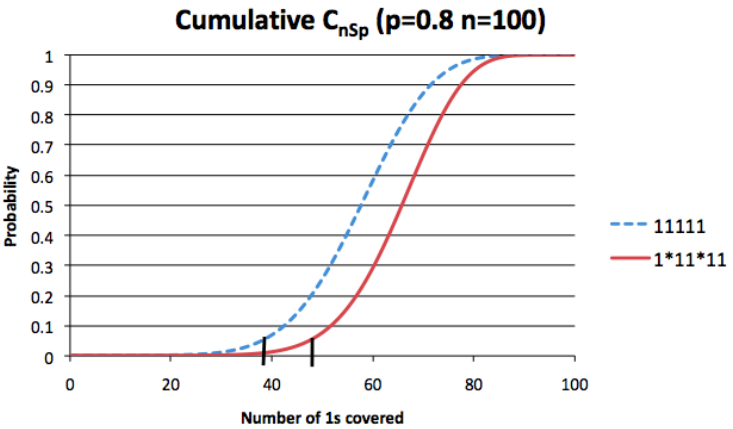
seed over a contiguous seed of equal weight. Calculation of the distribution over bit strings which represent confirmable alignments is discussed. An example illustrates the increase in hit extension threshold value under this assumption.

6 Acknowledgments

The authors would like to thank Brandon Mensing, Harshith Chennamaneni, and Won-Beom Kim for their assistance in validating our calculations.

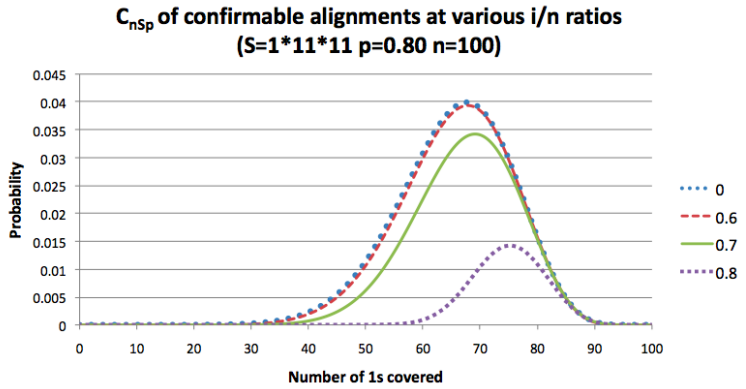


(a) C_{nSp} for seeds 11111 and $1 * 11 * 11$ at $p = 0.8$ and $n = 100$.

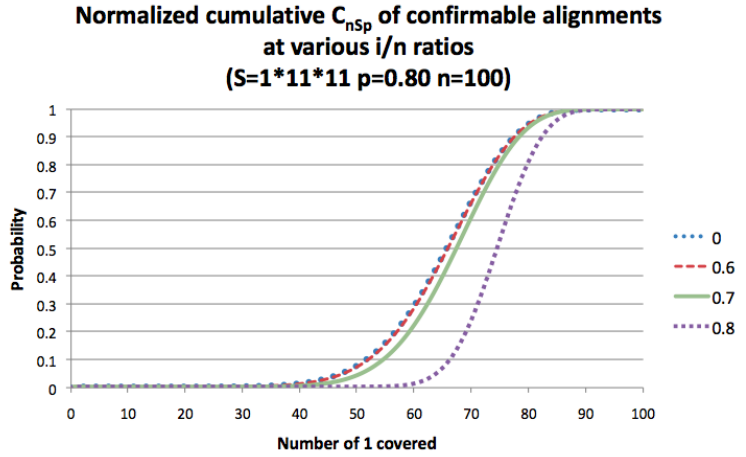


(b) Cumulative C_{nSp} for seeds 11111 and $1 * 11 * 11$ at $p = 0.8$ and $n = 100$. The small vertical lines mark the 5% threshold.

Fig. 3.



(a) C_{nSp} for seed $1 * 11 * 11$ at ratio $i/n >= 0.0$, 0.6, 0.7, and 0.8 with $p = 0.8$ and $n = 100$ and $n_{\min} = 14$.



(b) Normalized cumulative C_{nSp} for seed $1 * 11 * 11$ at ratio $i/n >= 0.0$, 0.6, 0.7, and 0.8 with $p = 0.8$ and $n = 100$ and $n_{\min} = 14$ (twice the seed length).

Fig. 4.

References

1. A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Comm. ACM*, 18:333–340, 1975.
2. G. Benson. Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research*, 27:573–580, 1999.
3. G. Benson and D.Y.F. Mak. Exact distribution of a spaced seed statistic for applications in DNA repeat detection. In *Proceedings of the 2008 International Workshop on Applied Probability (IWAP 2008)*, 2008.

4. G. Benson and X. Su. On the distribution of k-tuple matches for sequence homology: a constant time exact calculation of the variance. *J. Computational Biology*, 5:87–100, 1998.
5. J. Buhler, U. Keich, and Y. Sun. Designing seeds for similarity search in genomic DNA. *Journal of Computing and System Sciences*, 70:342–363, 2005.
6. S. Burkhardt and J. Kärkkäinen. Better filtering with gapped q-grams. *Fundam. Inform.*, 56(1-2):51–70, 2003.
7. U. Keich, M. Li, B. Ma, and J. Tromp. On spaced seeds for similarity search. *Discrete Applied Mathematics*, 138(3):253–263, 2004.
8. W.Y.W Lou. The exact distribution of the k-tuple statistic for sequence homology. *Statistics and Probability Letters*, 61(1):51–59, 2003.
9. B. Ma, J. Tromp, and M. Li. Patternhunter: faster and more sensitive homology search. *Bioinformatics*, 18:440–445, 2002.
10. D.Y.F. Mak and G. Benson. All hits all the time: Parameter free calculation of seed sensitivity. In *Proceedings of the 5th Asia-Pacific Bioinformatics Conference*, pages 327–340. Imperial College Press, 2007.
11. P. Warburton, J. Giordano, F. Cheung, Y. Gelfand, and G. Benson. Inverted repeat structure of the human genome: the X chromosome contains a preponderance of large highly homologous inverted repeats which contain testes genes. *Genome Res.*, pages 1861–1869, 2004.