Exact Distribution of a Spaced Seed Statistic for Applications in DNA Repeat Detection

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

[1] Computer Science, Biology, 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 $\{1, *\}$ which starts and ends with a 1. For example $S = 11 * 1$. S *occurs* in a binary string B at position k if S can be positioned so that the last letter in S aligns with the kth letter in B, and each 1 in S aligns with a 1 in B. A 1 in B is *covered* by S if there exists some occurrence of S in B such that the 1 in B aligns with a 1 in the occurrence of S.

We show how to compute the exact probability distribution for the *number* of 1s covered by a seed S in an i.i.d Bernoulli string of length n with probability of 1 equal to p. We refer to the new probability distribution as C_{nSp}, for *covered*, with S being the seed. When S consists entirely of 1s, for example $S = 111$, this reduces to the familiar R_{nkp} which is the probability distribution for the number of 1s which occur in *runs* of length k or longer (k is the number of 1s in S).

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.

The C_{nSp} distribution has applications in the detection of approximate DNA repeats using spaced seeds.

1 Introduction

Let a *seed*, S, be a string from the alphabet $\{1, *\}$ which starts and ends with a 1. For example $S = 11 * 1$. S *occurs* in a binary string B at a position k if S can be positioned so that the last letter in S aligns with the kth letter in B, and each 1 in S aligns with a 1 in B. For example, if $B = 101101111$, then S occurs in B at positions 6 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
```

A 1 in B is *covered* by S if there exists some occurrence of S in B such that the 1 in B aligns with a 1 in the occurrence of S. In the example, five 1s are covered, those at positions 3, 4, 6, 7, and 9.

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

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 i.i.d Bernoulli process with probability of 1, Prob(1), equal to p. We refer to the new probability distribution as C_{nSp}, for *covered*, with S being the seed. When the seed consists entirely of 1s, for example $S = 111$, this reduces to the familiar distribution R_{nkp} which is the probability distribution for the number of ones which occur in *runs* of length k or longer (k is the number of 1s in S) (Benson and Su (1998), Lou (2003))..

The R_{nkp} distribution has proven useful as a criterion for detecting tandem and inverted repeats in DNA sequences (Benson (1999), Warburton et al. (2004)). In those applications, the binary string (here called the *representative string*) represents a possible alignment between two sequence fragments that have a common evolutionary ancestor but 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.* a column) will remain the same. A typical assumption might be that 80% of the positions are expected to remain the same (*i.e.* $p = 0.8$).

Sequences with a common evolutionary ancestor are called *homologous* and 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.

Searching with seeds is an effective method, but it suffers from a serious drawback. A short seed will occur frequently at random, especially in long genomic sequences, and as a result, too much time is spent extending hits. One solution 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 repeats do not contain long unmutated stretches of nucleotides.

Another solution requires multiple seed hits clustered closely together and this is where the R_{nkp} distribution comes in. With it, we can estimate the expected number of aligned positions that participate in those hits. This allows us to set a reasonable threshhold criterion to distinguish between hits produced randomly by unrelated sequences and hits produced by homologous sequences. Fortuitously, the R_{nkp} distribution also allows us to set a distinct criterion for repeats of different sizes by varying the parameter n.

Spaced seeds. A few years ago, interest turned to spaced seeds (Ma et al. (2002), Buhler et al. (2003), Keich et al. (2004), Mak and Benson (2007)) which increase single hit sensitivity without simultaneously increasing the number of random hits. While a standard or *contiguous seed*, such as 111,

is a short word, a spaced seed contains a number of 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.

Spaced seeds have been studied primarily in terms of the single hit paradigm, in part, because there was no straightforward way to compute the distribution C_{nSp}. In this paper, we derive an efficient method to calculate the distribution exactly and this also yields an efficient method for the computation of the R_{nkp} distribution.

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. The probablity independent method has previously been used in a different form by us to identify *all* optimal spaced seeds for single hit homology detection (Mak and Benson (2007)).

2 Methods

Probability equivalence classes. Binary strings (representative strings) which have the same probability, when the parameter $p = \text{Prob}(1)$ 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 strings 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 $A^n[i, j]$ be the *number* of binary strings with i ones, j of which are covered by the seed S when considering all strings of length n. In this formulation, there are $(n + 1)^2$ equivalence classes and the combined probability of all strings which have j covered 1s is

$$\sum_{i=0}^{n} A^n[i, j]p^i(1-p)^{n-i}. \tag{1}$$

Fixing n and p and summing the terms as above gives us C_{nSp} directly. The calculation of the $A^n[i, j]$ is accomplished using a dynamic programming recurrence. In particular, $A^n[i, j]$ is the sum of terms from 2^k arrays where k is the length of the seed. In what follows, we describe the ideas behind the recurrence using an example seed $S = 1 * 11$, with $k = 4$.

Let $V_0, \ldots, V_{2^k-1}$ be the set of binary strings of length k with $V_x = x$ base 2. For example, $V_{11} = 1011$. Every binary string of length $n \geq k$ must start with one of these strings. We think of the V_x as nodes in a directed graph and we add two out edges to each node, one labeled with 1 and the other labeled with 0. The edge with label 1 goes from V_x to V_y if concatenating a 1 to the right end of V_x and discarding the left most bit yields V_y. For example, we

add an edge labeled 1 from V_{11} to V_7 because

1011 → concatenate 1 → 10111 → discard leftmost bit → 0111 = 7 base 2.

Similarly, the edge labeled 0 goes from V_x to V_z if concatenating a 0 to the right end of V_x and discarding the left most bit yields V_z. In the example, we add an edge labeled 0 from V_{11} to V_6 because

1011 → concatenate 0 → 10110 → discard leftmost bit → 0110 = 6 base 2.

For each V_x we keep a 4 dimensional array, $V_x[n, i, j, b_x]$, where b_x is one of a set of bit strings B_x specific to V_x. $V_x[n, i, j, b_x]$ stores a *count* with the following meaning. It is the number of strings of length n, each of which starts with V_x, containing i 1s with j 1s covered by the seed S such that the first $k-1$ bits in the string are covered as specified by b_x. For example, if $b_x = 011$, then the second and third 1s are covered.

To obtain the equivalence class counts $A^n[i, j]$, we sum the counts for the V_x:

$$A^n[i,j] = \sum_x \sum_{b_x \in B_x} V_x[n, i, j, b_x]. \tag{2}$$

The recursion for each V_x has the following specific form:

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

$$\begin{aligned} V_x[n,i,j,b_x] = & \sum_{b_y \in B_y(b_x)} V_y[n-1, i-od(xy), j-cvd(b_y), b_y] \\ & + \\ & \sum_{b_z \in B_z(b_x)} V_z[n-1, i-od(xz), j-cvd(b_z), b_z]. \end{aligned} \tag{3}$$

where V_x, V_y, and V_z are connected by edges as specified above. Note that V_x at n gathers information from V_y and V_z at $n-1$. This is the result of extending strings to the left by one bit in the recursion. *od* is the increase in 1s and *cvd* is the increase in covered 1s when doing this extension. $B_y(b_x)$ is the specific subset of B_y that will lead to the covered positions specified by b_x. Definitions of *ov*, *cvd*, and $B_y(b_x)$ are all straightforward and are omitted because of space limitations.

Base cases. The bottom of the recursion occurs at $n = k$ where k is the length of S. For $n < k$ no 1s are covered.

$$\forall i \in [0,k], j \in [0,k], b_x \in B_x \quad V[k,i,j,b_x] =$$

$$\begin{cases} 1 & \text{if } i = \text{number of 1s in } V_x,\ j = 0,\ b_x = 000,\ V_x \text{ not a seed pattern.} \\ 1 & \text{if } i = \text{number of 1s in } V_x,\ j = i,\ b_x = b_p,\ V_x \text{ a seed pattern.} \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

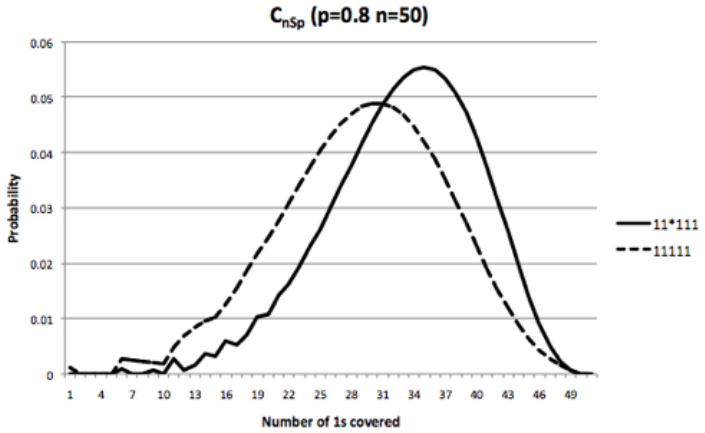

Fig. 1. C_{nSp} for seeds 11111 and $11 * 111$ at $p = 0.8$ and $n = 50$

where b_p is obtained from S by replacing every asterisk with a 0 and then dropping the right bit.

Time complexity. Calculating C_{nSp} by the method outlined above requires three steps:

1. Each V_x (of which there are 2^k for a seed of length k) computes a four dimensional array $V_x[n', i, j, b_x]$ for $n' \in [k \dots n], i \in [0, n], j \in [0, n], b_x \in B_x$ as in equations 3 and 4. For each element of an array, there are a set of additions, some from the array V_y and some from the array V_z. The time complexity is therefore $O(2^k n^3 \overline{B})$, where $\overline{B}$ is the average number of additions per V_x. We do not have a formula for $\overline{B}$, but for seeds we have examined up to length 12 with more 1s than *s, $\overline{B}$ ranges from 2 to 6.
2. After the $V_x[n, i, j, b_x]$ are calculated, $A^n[i, j]$ is computed as in equation 2. This has time complexity $O(2^k n^2 \overline{B_x})$ where $\overline{B_x}$ is the average size of B_x. From seeds we have examined as above, $\overline{B}_x$ ranges from 2 to 3.
3. After the $A^n[i, j]$ are calculated, the distribution C_{nSp} is computed as in equation 1. This has time complexity $O(n^2)$.

Step 1 dominates, so overall time complexity is $O(2^k n^3 \overline{B})$

3 Results

Figure 1 shows the exact probability distribution C_{nSp} for two seeds, 11111, and 11*111 for strings of length 50 and $p = 0.8$. 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 longer seed took approximately 30 seconds on a 64 bit dual processor 2.2 GHz computer.

4 Acknowledgments

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

References

Benson, G. (1999). Tandem repeats finder: a program to analyze DNA sequences. *Nucleic Acids Research*, **27**, 573–580.

Benson, G. and Su, X. (1998). On the distribution of k-tuple matches for sequence homology: a constant time exact calculation of the variance. *J. Computational Biology*, **5**, 87–100.

Buhler, J., Keich, U., and Sun, Y. (2005). Designing seeds for similarity search in genomic DNA. *Journal of Computing and System Sciences*, **70**, 342–363.

Keich, U., Li, M., Ma, B., and Tromp, J. (2004). On spaced seeds for similarity search. *Discrete Applied Mathematics*, **138**(3), 253–263.

Lou, W. (2003). The exact distribution of the k-tuple statistic for sequence homology. *Statistics and Probability Letters*, **61**(1), 51–59.

Ma, B., Tromp, J., and Li, M. (2002). Patternhunter: faster and more sensitive homology search. *Bioinformatics*, **18**, 440–445.

Mak, D. and Benson, G. (2007). 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.

Warburton, P., Giordano, J., Cheung, F., Gelfand, Y., and Benson, G. (2004). 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.