

All Hits All The Time: Parameter Free Calculation of Spaced Seed Sensitivity

Denise Y.F. Mak¹, Gary Benson^{2*}

¹Graduate Program in Bioinformatics, Boston University, Boston, MA 02215,

²Dept. of Computer Science, Dept. of Biology, Boston University, Boston, MA 02215

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXXX

ABSTRACT

Motivation: Standard search techniques for DNA repeats start by identifying *seeds*, that is, small matching words, that may inhabit larger repeats. Recent innovations in seed structure have led to the development of *spaced seeds* (Ma *et al.* (2002)) and *indel seeds* (Mak *et al.* (2006)) which are more sensitive than contiguous seeds (also known as k-mers, k-tuples, l-words, etc.). Evaluating seed *sensitivity* requires 1) specifying a homology model which describes types of alignments that can occur between two copies of a repeat, and 2) assigning probabilities to those alignments. *Optimal* seed selection is a resource intensive activity because essentially all alternative seeds must be tested (Li *et al.* (2006)). Current methods require that the model and probability parameters be specified in advance. When the parameters change, the entire calculation has to be rerun.

Results: In this paper, we show how to *eliminate* the need for prior parameter specification. The ideas presented follow from a simple observation: given a homology model, the alignments *hit* by a particular seed remain the same regardless of the probability parameters. Only the weights assigned to those alignments change. Therefore, if we know all the hits, we can easily (and quickly) find optimal seeds. We describe a highly efficient preprocessing step, which is computed just *once* for each seed. In this calculation, strings which represent possible alignments are *unweighted* by any probability parameters. Then we show several increasingly efficient methods to find the optimal seed when given specific probability parameters. Indeed, we show how to determine exactly which seeds can *never* be optimal under any set of probability parameters. This leads to the startling observation that out of thousands of seeds, only a handful have any chance of being optimal. We then show how to identify optimal seeds and the boundaries within probability space where they are optimal. We expect this method to greatly facilitate the study of spaced seed sensitivity and the use of alternative definitions of optimality.

Contact: dyfmak@bu.edu f

1 INTRODUCTION

We are interested in solving the following problem Mak and Benson (2007). Given 1) a homology model (iid, Markov chain, hidden Markov model, etc.) which describes the types of alignments that occur between DNA repeats, 2) a maximum length for the

alignments, and 3) a class of seeds (number of matches, number and type of wildcards), efficiently preprocess all the seeds in the class so that when given a set of probability parameters for the model, the optimal seed can be quickly identified.

As an example, assume the homology model is *match/mismatch iid*, where alignments of repeats are presumed to consist solely of matches and mismatches. A *representative string* for an alignment in this model is a binary sequence of 1's (matches) and 0's (mismatches), and the probabilities for 1 and 0 at each position in the string are independent and identically distributed. Also assume that alignments have maximum length 100 and that the seed class has 11 matches and 7 match/mismatch wildcards (this is the PatternHunter seed class (Ma *et al.* (2002))). We seek to preprocess all the seeds in the class (there are 16 choose 7 or 11,440 of them and because mirror images have the same sensitivity, actually 5,720 different seeds) so that when given the parameters which fully describe the homology model we can quickly choose the optimal seed. In this case, if the parameters, specified as (probability of match, alignment length), are (0.7, 64), then we want to quickly identify the optimal seed, which is the PatternHunter seed (111*1**1*1**11*111) (Ma *et al.* (2002)). *But*, if we are then given another set of parameters which can be different in both probabilities and length, say (0.75, 50) we want to again quickly find the optimal seed, which is (111*1*1**11*1**111) (Choi *et al.* (2004)). Even more, we would like to know at what particular values (probability and length) *optimality ends for one seed and begins for another*.

Many existing methods (Brejová *et al.* (2004); Choi and Zhang (2004); Mak *et al.* (2006)) find the optimal seed for a single combination of homology model, parameter set, and seed class by enumerating *all patterns* for a given seed (*e.g.*, 128 patterns for the PatternHunter seed) for *all seeds* in the class (*e.g.*, 5,720 seeds for the PatternHunter class) and testing these against the representative strings from the homology model. The most efficient methods have time complexity in $O(LPwS)$ where L is the string length, P is the number of patterns per seed (an exponential in the number of wildcards), w is the length of the seed, and S is the number of seeds (the number of combinations of positions available in a seed for the wildcards). Importantly, when the parameter set changes, the entire calculation must be rerun. Choi *et al.* (Choi *et al.* (2004)), in an extensive sampling of seed sensitivities, have taken this approach, rerunning a seed sensitivity algorithm for many combinations of parameter values.

*This research was supported in part by NSF grant DBI-0413462.

Our seed preprocessing, which only needs to be calculated once, increases the time complexity by a factor dependent on the homology model (see section 3). But, it gives us three benefits. It allows repeated searches for the optimal seed with different parameters to proceed more efficiently, it allows us to identify seeds that can *never* be optimal, and allows us to partition the parameter space into regions, each covered by a single optimal seed. (Note: a general method for parametric inference, which allows the partitioning of parameter space, has been described in (Pachter and Sturmfels (2004)).)

The ideas presented in this paper follow from a simple observation: given a model, the alignments *hit* by a particular seed remain the same regardless of the probability parameters. Only the weights assigned to those alignments change. Therefore, if we know all the hits, we can easily (and quickly) find optimal seeds. In essence, our preprocessing identifies, for each seed, the specific set of representative strings it hits. Because the number of strings can be enormous (2^{100} for alignments of length 100 in the binary iid model) we cannot actually save the entire set of strings for each seed. Rather, we save *counts* for each probability class that the strings can occupy.

The remainder of the paper is organized as follows. In section 2 we give a formal definition of the problem. In section 3 we present the seed preprocessing method. In section 4 we show several increasingly efficient methods to find the optimal seed when given specific model parameters. In section 6 we show results from applying our parameter free calculation to several seed classes. Finally, in section 7 we discuss implications of this method and several uses of the information provided by preprocessing the seeds.

2 PROBLEM DESCRIPTION

Formally, the problem we address is:

Seed All Hits Preprocessing

- **Given:** 1) a homology model (specified by probability variables), 2) a seed class (specified by seed width, number and type of wildcards), and 3) a maximum length for alignments called the target length t .
- **Compute:** A preprocessing of the seed class so that when values are assigned to the probability variables and a length L between 1 and t is selected, the optimal seed is returned.

For the remainder of this paper we will assume that the homology model is binary (match/mismatch) iid, *i.e.*, the Bernoulli model. The method can be extended to iid models with more parameters such as the ternary alphabet model (match/transition/mismatch) of Noé and Kucherov (Noé and Kucherov (2004)) and to Markov chains as well.

3 METHOD

We start with some definitions.

Definition 1: An alignment is described by a *representative string* over a binary alphabet: 1 indicates a match and 0 indicates a mismatch. A *probability equivalence class* contains representative strings that have the same probability when the probability parameters are specified.

In the Bernoulli model, strings belong to an equivalence class if they are the same length and contain the same number of 1's because all such strings have the same probability when $P(1) =$ the probability of 1, is defined. For strings with target length t , the number of equivalence classes is $t + 1$, corresponding to the possible number of 1's, *i.e.* 0, 1, 2, \dots , t . For example, when $t=100$, each string with 70 1's, *no matter how arranged*, has probability $p^{70}(1 - p)^{30}$.

Definition 2: A *seed* is a string beginning and ending with a 1 and containing 1's and *'s, where 1 represents a match and * is a wildcard that denotes either 1 (match) or 0 (mismatch). The length w of a seed is the total number of 1's and *'s. A *pattern* of a seed is any string of 1's and 0's which is obtained by replacing each

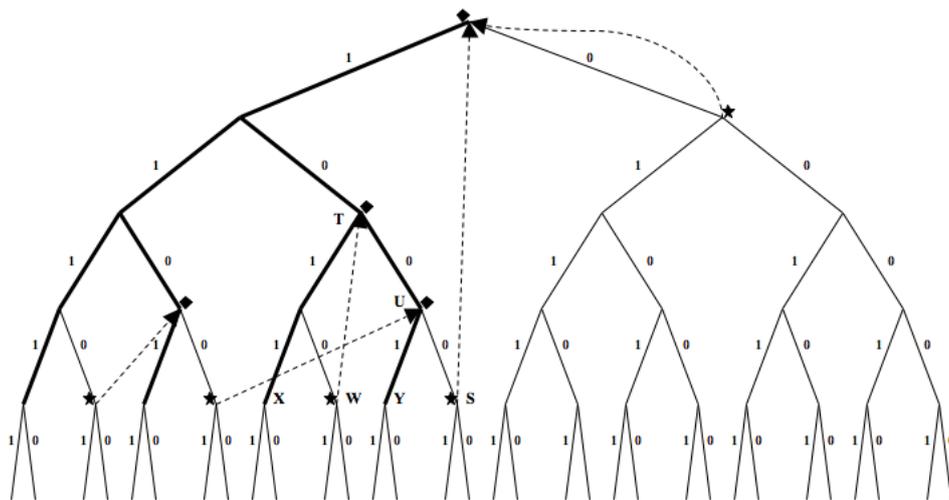


Fig. 1. Aho-Corasick tree for seed 1**1 (bold) overlaid on all-strings tree. Phantom nodes are shown with *, fail-to nodes are shown with ., and phantom nodes are linked to their fail-to nodes with $--\rightarrow$. Letters mark nodes discussed in the text.

wildcard in the seed by either a 1 or 0. A pattern *hits* a representative string if the pattern occurs in the string. Any pattern or set of patterns will hit a certain number of representative strings and these strings will fall into different equivalence classes. We record how many strings in each equivalence class are hit using a vector of counts which we call the *PECC vector* (probability equivalence class counts vector) of length $t + 1$ with the classes ordered by the number of 1's from zero to t .

Our preprocessing algorithm derives from a recognition that seed hits can be summarized as hits within probability equivalence classes, that is, by PECC vectors (see figure 2). The result of the preprocessing for a single seed is a collection of PECC vectors, one for each representative string length L from 1 to t . Conceptually, the preprocessing, generates all possible representative strings for all lengths L from 1 to t , tests each to see if it contains a hit to the seed, and stores counts for the ones that do in the appropriate equivalence class. In actuality, strings are *not* generated. Instead, we collect and store PECC vectors which are used recursively in a dynamic programming algorithm.

Our data structure for collecting information is the Aho-Corasick tree (AC tree) for the seed patterns which is constructed as the initial step of the preprocessing. For the remainder of this section, we illustrate the preprocessing algorithm with a simple example using the seed $1^{**}1$, and a target length $t = 5$. In figure 1, we show the AC tree for $1^{**}1$ overlaid on the tree of all representative strings (*all-strings tree*). The following definitions refer to that figure.

Definition 3: For a node n , the number of edges in the path from the root to n is $depth(n)$. The *node string* for n , in the AC tree or all-strings tree, is the concatenation of edge labels from the root to n . The node string for a leaf (in either tree) is a *leaf string*. The leaf string for a leaf f in the AC tree is a pattern of the seed and is called pat_f . The number of 1's in the node string for node n is $ones(n)$. Each leaf, f , of the AC tree is the root of a subtree T_f in the tree of all-strings. A *fail node* in the AC tree is a non-leaf which has one *phantom* child, that is, a child node in the all-strings tree which does not occur in the AC tree. Each phantom p is the root of a subtree T_p in the all-strings tree. For each phantom p , its *fail-to* node ft_p is the node in the AC tree whose node string is the longest *proper* suffix of the node string for p .

Observation: Each phantom p has a fail-to node in the AC tree. The depth of a fail-to node ft_p is at most one less than the depth of p (alternately, at most the depth of the fail node parent of p).

Computing the PECC Vectors. Consider the AC tree overlaid on the all-strings tree in (figure 1). PECC vectors are recorded for leaf, phantom, and fail-to nodes. The vector for a given node X and a given all-strings tree depth L (denoted X_L) records the number of strings of length L in each equivalence class which start with the node string for X and are hit by the seed. These are the leaf strings for the leaves of the all-strings subtree rooted by X which are hit by the seed. Our dynamic programming uses the fact that each phantom node can find information about longer strings in a PECC vector stored at its fail-to node due to the longest proper suffix relationship between the nodes.

Base case: $L = w$. (For $L < w$, the PECC vector is all zeros.) The leaf strings of the AC tree are the only strings hit by the seed. The PECC vector for these strings can be determined when the tree is constructed. For example, in the tree in figure 1, the leaf X with leaf string 1011 has vector $X_4 = (0, 0, 0, 1, 0)$ where position in the vector indicates number of 1's in the equivalence class from zero to

Table 2. Results for several seed classes for homologous regions of length 64. Note the very low number of dominant and optimal seeds in all classes. All computation was done on seeds after removal of mirrors (one of each pair of distinct mirror image seeds). Time is for computing all PECC vectors in a single run for regions of length 50 to 64. Lengths under 50 were also computed in the same run, but PECC vectors were not saved. Calculations were made on a dual 1GHz PIII processor with 2GB RAM.

class		number of seeds				Time
1's	*s	all	minus mirrors	dominant	optimal	(hr:min:sec)
9	6	1,716	868	7	4	0:03:40
10	6	3,003	1,519	6	4	0:06:27
11	7	11,440	5,720	12	5	0:47:00
12	6	8,008	4,032	10	3	0:19:06
12	7	19,448	9,752	36	10	1:24:35
13	6	12,376	6,216	13	7	0:32:25
13	7	31,824	15,912	20	5	2:28:32
14	6	18,564	9,324	20	7	0:49:51
14	7	50,388	25,236	22	3	4:09:25
15	6	27,132	13,608	24	4	1:17:58
15	7	77,520	38,760	23	5	6:29:26

L . For this node and depth, there is just one string, 1101, with three 1's, which is hit by the seed. Similarly, the leaf Y with leaf string 1001 containing two 1's has vector $Y_4 = (0, 0, 1, 0, 0)$.

Induction Step: $L = k + 1$. PECC vectors are already stored for $L = w, \dots, k$. We calculate a PECC vector at level $k + 1$ for each of the three node types as follows:

- **Leaf.** Each leaf, f , of the AC tree is the root of a subtree T_f in the tree of all-strings. Every leaf string in T_f (from the common root of the trees) is hit by the seed because it is an *extension* of pattern pat_f to length $k + 1$. Using regular expression notation, the leaf strings in T_f can be specified as

$$pat_f\{0|1\}^{k+1-w}.$$

The PECC vector for leaf f is therefore the vector of counts for *all* the leaf strings in T_f . These strings fall into $k + 2$ probability equivalence classes. Only $k + 2 - w$ of these are non-zero and they correspond to strings containing $ones(f) + i$ 1's with i in the range

$$i = 0, \dots, k + 1 - w.$$

The number of strings for each i is determined by a combinations formula

$$(k + 1 - w) \text{ choose } i.$$

Notice that the *subtrees* rooted by all AC tree leaves are identical in their edge labeling and can be treated identically, only the values $ones(f)$ to the root of these subtrees differ. The combination formulas can be computed in advance and consulted when needed through a look-up table. Examples of vectors for leaves X and Y at depths 5, 6, 7, and 8 are shown in table 1.

- **Phantom node.** Each phantom child p of a fail node n , is the root of a subtree T_p in the tree of all-strings. Unlike the subtrees T_f , not all leaf strings in T_p are hit by the seed. But, there is a simple way to determine which strings are hit. The information is already stored (as part of the dynamic programming) as a

Table 1. PECC vectors for nodes S, T, U, W, X and Y in figure 1 from depth 4 to 8.

Node	Depth				
	4	5	6	7	8
S	-	-	-	-	(0 0 0 1 2 1 0 0 0)
T	(0 0 1 1 0)	(0 0 1 2 1 0)	(0 0 1 4 4 1 0)	(0 0 1 5 8 5 1 0)	(0 0 1 7 16 15 6 1 0)
U	(0 0 1 0 0)	(0 0 1 1 0 0)	(0 0 1 2 1 0 0)	(0 0 1 3 3 1 0 0)	(0 0 1 5 8 5 1 0 0)
W	-	-	(0 0 0 1 1 0 0)	(0 0 0 1 2 1 0 0)	(0 0 0 1 4 4 1 0 0)
X	(0 0 0 1 0)	(0 0 0 1 1 0)	(0 0 0 1 2 1 0)	(0 0 0 1 3 3 1 0)	(0 0 0 1 4 6 4 1 0)
Y	(0 0 1 0 0)	(0 0 1 1 0 0)	(0 0 1 2 1 0 0)	(0 0 1 3 3 1 0 0)	(0 0 1 4 6 4 1 0 0)

PECC vector in p 's fail-to node ft_p . There is a difference in recursion level which reflects the difference in p 's and ft_p 's depth. There is also a difference in the number of 1s which reflects the prefix part of p 's node string which does not form the node string for ft_p . For example, from the tree in figure 1, $W_6 = T_4$ (because the difference in depth between these nodes is 2), with the values shifted right by one position (due to the extra 1 in W 's node string relative to T 's node string), and additional zeros padding absent values. See also table 1.

- **Fail-To node.** Each fail-to node ft must store a *series* of PECC vectors for look-up by phantom nodes that point to it. Each vector in the series represents a different depth of the all-strings tree. The vector for node ft at iteration L is the sum of vectors for nearest descendant leaf, phantom, and fail-to nodes from its AC subtree at the same iteration. For example, in figure 1, the vector for node U at level 8 is the sum of vectors Y_8 and S_8 and the vector for node T at level 6 is the sum of vectors X_6 , W_6 , and U_6 (see table 1). The PECC vectors for the root, which is also a fail-to node, are the output vectors for the seed, one for each length L from w to t .

Time complexity

Theorem 1: *The time complexity for computing the PECC vectors for a single seed for all lengths between $L = 1$ and $L = t$ is $O(vtPx)$ where v is the number of probability equivalence classes, t is the target length, P is the number of patterns for the seed, and x is the number of ones in the seed. The time complexity for all seeds in a class is $O(vtPxS)$ where the class contains S seeds.*

Proof: Building the AC tree for all patterns of the seed, determining phantom nodes and fail-to nodes is linear in the size of the AC tree which has size $O(Px)$. There are $t - w$ iterations through the tree. In each iteration, PECC vectors are updated for the leaves, fail nodes and fail-to nodes. The total number of these nodes is linear in the size of the tree. For efficiency, the vector for each leaf, phantom, and fail-to node is pushed up the tree for addition into its nearest ancestor fail-to node. This assures that the number of additions are also linear in the size of the tree. Note that every addition requires adding two vectors of size at most v . Determining where each vector must be added, and in the case of phantom nodes, the difference in recursion level for the copied vector, can be done with a single traversal of the AC tree when it is constructed. ■

Note: In the match/mismatch iid homology model, $v = t + 1$, so the time complexity is $O(t^2Px)$. This is an increase by a factor of v over the current best calculations for seed sensitivity for a single probability parameter value p .

Algorithm performance. Table 2 gives times for computing the PECC vectors for several seed classes.

4 FINDING OPTIMAL SEEDS

Below we present three ways to find optimal seeds when given a probability value p and a homology region length L . We assume that the seeds in the class under consideration have already been preprocessed for their PECC vectors.

4.1 Scan the PECC Vectors

Using the PECC vector for length L from each seed, compute the total sensitivity of the seed and return the seed with maximum sensitivity. A PECC vector for length L contains $L + 1$ equivalence classes. The probability of a representative string which belongs to the i th equivalence class (with $i \in [0, \dots, L]$ specifying the number of 1's in the string) is $p^i q^{L-i}$ where p is the probability of 1 and $q = 1 - p$. Let the count in equivalence class i be C_i . Then the polynomial $C_0 \cdot p^0 q^L + C_1 \cdot p^1 q^{L-1} + \dots + C_L \cdot p^L q^0$ gives the sensitivity of the seed. This method recomputes the sensitivity for each seed when p changes.

4.2 Dominant Seeds

Definition 4: For two seeds A and B with PECC vectors \vec{V}_A and \vec{V}_B , if the difference vector $\vec{D} = \vec{V}_A - \vec{V}_B$ contains only zero values, then A and B are *equivalent*. If the difference vector contains no negative values and some values are positive, then A *dominates* B . If A is never dominated by another seed, then A is *dominant*.

If A dominates B , it means that in every probability equivalence class, A hits at least as many strings as B and in some classes hits more. In this situation, B can *never* be optimal and can be removed from the seed class. For this method, we first determine the dominant seeds, then compute sensitivity only for their PECC vectors as in the previous section, and return the seed with maximum sensitivity. Determining which seeds are dominant occurs just once. When p changes, only the sensitivity of each dominant seed must be recomputed.

Amazingly, only a few seeds are dominant within the seed classes for the match/mismatch iid homology models we have investigated. For example, in the PatternHunter seed class, with 5720 seeds (after removing equivalent mirrors), at length 64, only 12 seeds are dominant. Table 2 shows the number of dominant seeds in several seed classes.

4.3 Partitioning the Parameter Space

Definition 5: For two dominant seeds A and B with PECC vectors \vec{V}_A and \vec{V}_B , if the difference vector $\vec{D} = \vec{V}_A - \vec{V}_B$ contains both positive and negative values, then A and B flip. A has more hits in some probability equivalence classes than B , and B has more hits in other classes. In this case, A and B will usually partition the probability space (figure 3) with A having better sensitivity in some regions of the space (where A wins relative to B) and B having

better sensitivity in others (where B wins relative to A). A dominant seed A is *optimal* in some region of probability space only if A wins in that region relative to every other dominant seed.

Figure 2 shows the PECC vectors and difference vector for a pair of seeds that flip. To determine the winning regions for a seed pair A and B in the match/mismatch iid model, we solve for the roots of a polynomial equation with a single unknown variable p . Let the difference in counts in equivalence class i be D_i .

	number of 1's	hit counts		different in hit counts
		seed A	seed B	A - B
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	38	38	38	0
6	1520	1520	1520	0
7	29640	29640	29640	0
8	375332	375332	375334	-2
9	3468729	3468729	3468766	-37
10	24928629	24928629	24928696	-67
11	144948172	144948172	144942776	5396
12	700456139	700456139	700364916	91223
13	2867679722	2867679722	2866827015	852707
14	10087980802	10087980802	10082362006	5618796
15	30819118260	30819118260	30790682312	28435948
16	82428580640	82428580640	82313087704	115492936
17	194186285079	194186285079	193799799947	386485132
18	404765317464	404765317464	403681622157	1083695307
19	748926920505	748926920505	746353121681	2573798824
20	1232762203160	1232762203160	1227550871937	5211331223
21	1807611479754	1807611479754	1798588935398	9022544356
22	2362723608570	2362723608570	2349367736133	13355872437
23	2753640936852	2753640936852	2736789617398	16851319454
24	2861644355241	2861644355241	2843634715311	18009639930
25	2652331424044	2652331424044	2636186283832	16145140212
26	2194080691374	2194080691374	2182108728673	11971962701
27	1622400592905	1622400592905	1615198100214	7202492691
28	1075039055326	1075039055326	1071615007952	3424047374
29	640344850572	640344850572	639104255291	1240595281
30	343883821799	343883821799	343558130745	325691054
31	166774138475	166774138475	166716560683	57577792
32	73001069494	73001069494	72994947436	6122058
33	28759911342	28759911342	28759587965	323377
34	10150595182	10150595182	10150589139	6043
35	3190187285	3190187285	3190187260	25
36	886163135	886163135	886163135	0
37	215553195	215553195	215553195	0
38	45379620	45379620	45379620	0
39	8145060	8145060	8145060	0
40	1221759	1221759	1221759	0
41	148995	148995	148995	0
42	14190	14190	14190	0
43	990	990	990	0
44	45	45	45	0
45	1	1	1	0

$$-2x^8(1-x)^{37} - 37x^9(1-x)^{36} - 67x^{10}(1-x)^{35} + 5396x^{11}(1-x)^{34} + 91223x^{12}(1-x)^{33} + 852707x^{13}(1-x)^{32} + 5618796x^{14}(1-x)^{31} + 28435948x^{15}(1-x)^{30} + 115492936x^{16}(1-x)^{29} + 386485132x^{17}(1-x)^{28} + 1083695307x^{18}(1-x)^{27} + 2573798824x^{19}(1-x)^{26} + 5211331223x^{20}(1-x)^{25} + 9022544356x^{21}(1-x)^{24} + 13355872437x^{22}(1-x)^{23} + 16851319454x^{23}(1-x)^{22} + 18009639930x^{24}(1-x)^{21} + 16145140212x^{25}(1-x)^{20} + 11971962701x^{26}(1-x)^{19} + 7202492691x^{27}(1-x)^{18} + 3424047374x^{28}(1-x)^{17} + 1240595281x^{29}(1-x)^{16} + 325691054x^{30}(1-x)^{15} + 57577792x^{31}(1-x)^{14} + 6122058x^{32}(1-x)^{13} + 323377x^{33}(1-x)^{12} + 6043x^{34}(1-x)^{11} + 25x^{35}(1-x)^{10}$$

Fig. 2. Top) PECC vectors for seeds A: 111*1**1 and B: 111**1*1 (for the seed class with five 1's and three *'s and homologous region length 45) and the difference vector showing positive and negative values. Bottom) Polynomial constructed from the difference vector.

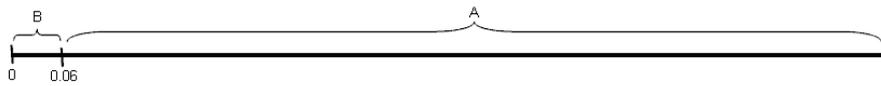


Fig. 3. Partition of parameter space into regions won by seeds A and B from figure 2

Then our polynomial P_{AB} , containing $L + 1$ terms has the form: $D_0 \cdot p^0 q^L + D_1 \cdot p^1 q^{L-1} + \dots + D_L \cdot p^L q^0$ (see bottom of figure 2). Replacing q with $1 - p$, we solve the equation $P_{AB} = 0$ using the Mathematica function SOLVE (Wolfram Research (2005)). There are L roots (complex and real) that exist but only real roots between 0 and 1 are of interest. Testing the PECC vector for A on either side of the roots reveals those regions where A wins. To find where a dominant seed A is optimal, we intersect the regions where A wins relative to every other dominant seed.

For this method, the probability space is first partitioned. The optimal seed is then found for the specific value p , and the sensitivity of that seed is computed using the seed's PECC vector. Partitioning occurs only once. When p changes, the optimal seed must again be determined by lookup and its sensitivity computed.

5 CONFIRMABLE ALIGNMENTS

The seed preprocessing discussed so far assumes that the set of representative strings includes any binary string. This means that representative strings with many more 0s than 1s are possible, although such strings do not realistically represents homologous sequences. In fact, if sequences differ in too many positions, they would not be considered homologous. Nonetheless, most methods for calculating seed sensitivity make this assumption. In this section, we restrict our previous analysis of spaced seeds to a model of alignments which can be *confirmed*.

Definition 6: An alignment is *confirmable* if a seed hit can be extended by an alignment algorithm to detect the homology.

We model confirmable alignments in the following way. Given a representative string of an alignment, if every suffix, of length greater than or equal to some small length n_0 , contains a minimum fraction of 1s, then the alignment is confirmable. In other words, in the recursion, for all values of $n \geq n_0$ we ignore entries where $i/n < f$ where i is the number of 1s and f is our threshold fraction. In what follows, we use $n_0 = 25$ and $f = 0.6$. The recursion does not change, except for the adjustment stated above, and we are still able to determine optimal seeds as before.

The effect is to exclude from the recursion binary strings in which there are too few 1s or in which the 1s are clumped together, leaving large patches of 0s. The minimum size n_0 is not necessary, but is

included so as not to exclude strings that are sparse within the first one or two seed lengths (if n_0 was not used, then every string that begins with a 0 would be excluded).

6 RESULTS

Table 2 shows the results of our preprocessing for dominant and optimal seeds on several seed classes. Note the extremely small number of dominant and optimal seeds in these classes. Table 3 shows the partition of probability space by the optimal seeds in each class. Note that some seeds are optimal in more than one region. Figure 4 illustrates the partition for a single seed class, the PatternHunter class, for homologous regions of length 64. More generally, we are interested in the partition of probability space for all seeds in the same weight class, not just a seed class.

Definition 6: A *weight class* of weight k contains all seed classes with k 1's.

Seeds in the same weight class are of interest because they all have the same probability of a random hit, which depends on the number of 1's in the seed. Figure 5 illustrates the partition of probability space by seeds drawn from all seed classes of weight 11, *i.e.*, those having 11 ones and any number of wildcards. Note that the contiguous seed is optimal only at the low end of the probability range. The number of wildcards in the optimal seed increases towards the high end of the range and ultimately decreases again at the highest levels.

Figure 6 shows the partition by seeds from the weight 11 class under the assumption of confirmable alignments. Note the different set of optimal seeds compared to figure 5. Note also that this graph represents the weight class restricted to 11 or fewer wildcard positions due to the prohibitive time required to process all seeds in classes with 12 or more wildcards.

7 DISCUSSION

Variation in intervals for optimal seeds as homology region length varies. The issue of optimality in the limit with respect to length of the homologous regions has been addressed in (Buhler *et al.* (2005); Choi and Zhang (2004); Choi *et al.* (2004)). Using our partition method, we can examine variations in optimal seeds



Fig. 4. Intervals of optimality in probability space for optimal seeds from the PatternHunter class at region length 64. A: 111*1**11*1**111, B: 111*1*1**11*1**111, C: (PH seed) 111*11**1*1**111, D: 11**111*1**1*111*1, E: 1111*1*11**1**111, F: 1111**1**11*1111

and the regions of probability space they cover as the length of the homologous region varies. Figure 5 illustrates this variation for the weight 11 seed classes. Note that in this case, the set of optimal seeds remains relatively stable over a range of lengths, e.g., seeds A, B, C, D, E, G, I, etc. It is interesting to note that if the original PatternHunter paper (Ma *et al.* (2002)) had analyzed alignments of length 55 rather than length 64, what we call the PatternHunter seed would be seed G rather than seed H.

If the property of interval stability holds generally, it could be used to make the time to search for dominant seeds nearly linear in the number of seeds in the class (after preprocessing) rather than quadratic in that number. To search for dominant seeds at length L we could first choose the optimal seeds at length $L - 1$ and then

Table 3. Optimal seeds and the range of probabilities where they are optimal for several seed classes at region length 64. Note that the same seed can be optimal in more than one region.

number of 1's	number of *'s	optimal seeds	probability range		
9	6	A:111**1*1**1*11	0 - 0.110266686		
		B:111*1*1**1**111	0.1110266686 - 0.4327682188		
		C:111***1*1**11	0.4327682188 - 0.9694790865		
		D:111*1*1**1**111	0.9694790865 - 0.9991450536		
10	6	A:111**1*1**1*11	0 - 0.0231912575		
		B:111*1**1*11**11	0.0231912575 - 0.0457879868		
		C:111*1*1**1**11	0.0457879868 - 0.9436271851		
		D:111**1*1**1**11	0.9436271851 - 1		
11	7	A:111*1**1*1**111	0 - 0.0524790924		
		B:111*1*1**1*1**111	0.0524790924 - 0.0775105071		
		C:111*1**1*1**1*11	0.0775105071 - 0.7304317142		
		B:111*1*1**1*1**111	0.7304317142 - 0.9845899783		
		D:111**1*1**1*1**11	0.9845899783 - 0.9997355115		
12	6	A:111*1**1*1**111	0 - 0.0125740804		
		B:111*1**1*1**111	0.0125740804 - 0.9818956319		
		C:111**1*1**1*1**11	0.9818956319 - 1		
		A:111*1**1*1**111	0 - 0.0269449089		
		B:111*1**1*1**111	0.0269449089 - 0.0501533511		
12	7	C:111**1*1**1*1**11	0.0501533511 - 0.1324581579		
		D:111**1*1**1*1**11	0.1324581579 - 0.1713344621		
		E:111**1*1**1*1**11	0.1713344621 - 0.2193278527		
		F:111*1**1*1**1*11	0.2193278527 - 0.5616667374		
		G:111*1**1*1**1*11	0.5616667374 - 0.8266957477		
		H:111*1**1*1**1*11	0.8266957477 - 0.9600165421		
		I:111*1**1*1**1*11	0.9600165421 - 0.9824252510		
		G:111*1**1*1**1*11	0.9824252510 - 0.9833842249		
		J:111*1**1*1**1*11	0.9833842249 - 1		
		13	6	A:111**1*1**1*1**11	0 - 0.0250132023
				B:111**1*1**1*1**11	0.0250132023 - 0.5568886832
C:111**1*1**1*1**11	0.5568886832 - 0.9726822943				
D:111**1*1**1*1**11	0.9726822943 - 0.9845644894				
E:111**1*1**1*1**11	0.9845644894 - 0.9991885007				
F:111**1*1**1*1**11	0.9991885007 - 0.9999389894				
G:111**1*1**1*1**11	0.9999389894 - 1				
A:111*1**1*1**1*1**11	0 - 0.0128994408				
13	7	B:111*1**1*1**1*1**11	0.0128994408 - 0.1449709023		
		C:111*1**1*1**1*1**11	0.1449709023 - 0.7775640305		
		D:111*1**1*1**1*1**11	0.7775640305 - 0.9771389517		
		E:111*1**1*1**1*1**11	0.9771389517 - 1		
		14	6	A:111*1**1*1**1*1**11	0 - 0.0255990753
B:111*1**1*1**1*1**11	0.0255990753 - 0.0266989880				
C:111*1**1*1**1*1**11	0.0266989880 - 0.1397167664				
D:111*1**1*1**1*1**11	0.1397167664 - 0.8585536713				
E:111*1**1*1**1*1**11	0.8585536713 - 0.9733966204				
F:111*1**1*1**1*1**11	0.9733966204 - 0.9774612165				
14	7	G:111*1**1*1**1*1**11	0.9774612165 - 1		
		A:111*1**1*1**1*1**11	0 - 0.0263720408		
		B:111*1**1*1**1*1**11	0.0263720408 - 0.4156480746		
		C:111*1**1*1**1*1**11	0.4156480746 - 1		
15	6	A:111*1**1*1**1*1**11	0 - 0.1140558268		
		B:111*1**1*1**1*1**11	0.1140558268 - 0.5238349212		
		C:111*1**1*1**1*1**11	0.5238349212 - 0.8916876889		
		D:111*1**1*1**1*1**11	0.8916876889 - 1		
15	7	A:111*1**1*1**1*1**11	0 - 0.6176146226		
		B:111*1**1*1**1*1**11	0.6176146226 - 0.6698669044		
		C:111*1**1*1**1*1**11	0.6698669044 - 0.9307720060		
		D:111*1**1*1**1*1**11	0.9307720060 - 0.9762530821		
		E:111*1**1*1**1*1**11	0.9762530821 - 1		

Table 4. Seeds with the most hits in equivalence classes (number of 1's) for weight 11 seeds at region length 64. Seed G from figure 5 is optimal for the model with probability of match equal to 0.60, but seed E does best at hitting alignments with 60% matches.

seed	most hits in equivalence classes	corresponding to % of matches
E: 111*11*11*1*111	33-39	51%-61%
F: 1111*1*1**11*111	40-42	62%-66%
H: 111*11**1*1**1*111	43-47	67%-74%
I: 111*1*1**11*1**111	48-54	75%-85%

test the remaining seeds against only these to eliminate the non-dominant ones.

Optimality within probability equivalence classes. To say a seed is optimal for some homology model can be somewhat misleading. For example, the seed G (111*111*1**1*111) from figure 5 is optimal for the model with probability of matching equal to 60% at region length 64, but this might be construed to mean that it does best at hitting alignments which contain 60% matches, which is not the case. The seed E is better for these alignments as shown in table 4. In fact, G is not best at hitting any individual equivalence class at length 64. This leads us to propose an alternative concept of optimality. One might be interested in seeds that do best in a particular probability equivalence class, rather than seeds that do best across all the classes. Our method for computing PECC vectors allows us to identify these winners, which by definition are dominant seeds.

Where is an optimal seed most sensitive? For an optimal seed A , we may be interested in knowing in what part of its interval of optimality it is most sensitive. For the match/mismatch iid model examined here, this can be determined by taking the derivative of the polynomial formed from the PECC vector for A and finding the sensitivity at the real roots between 0 and 1 that give a maximum (or the endpoints of the interval of optimality if no such roots exist within the interval). For example, the PatternHunter seed at length 64, from figure 5, has a maximum sensitivity of 0.6071 at $p=0.7304$ which is at the upper endpoint of its interval.

Confirmable alignments. The use of confirmable alignments is an attempt to incorporate a more realistic model of sequence homology than the standard Bernoulli model. Buhler *et al.* (2005) approached this from a model free perspective by testing seeds against a sampling of real alignments. It is interesting to note that under our assumption of confirmable alignments, all the optimal seeds are replaced for parameter p below 0.60 (the threshold fraction of 1s or matches), while the PatternHunter seed and two others remain optimal for p above 0.60, at least for homology regions of length 64. In comparison with the last paragraph, the PatternHunter seed has an increased sensitivity of 0.6417, at $p=0.7304$, when considering only confirmable alignments as defined here.

8 CONCLUSION

We have presented a new, efficient preprocessing method to determine optimal seed sensitivity without first specifying the probability parameters for a homology model. This allows us to find the optimal seed quickly once the parameters are specified. It reveals that, under the Bernoulli homology model, the overwhelming

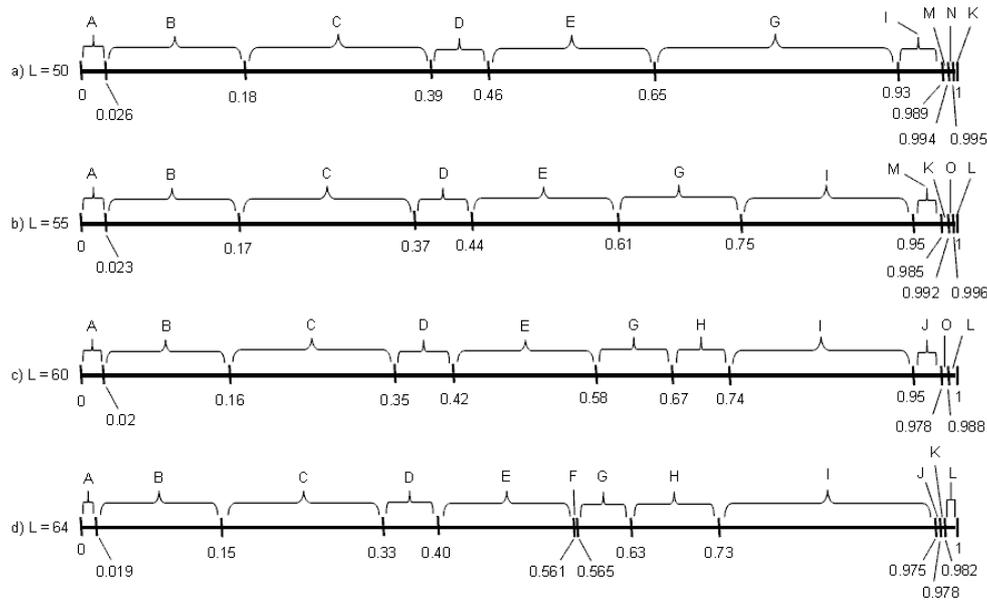


Fig. 5. Probability intervals for optimal seeds of weight 11 at varying homology region lengths. a) region length 50, b) region length 55, c) region length 60, d) region length 64. A: 11111111111, B: 111111*11111, C: 1111*111*1111, D: 11111*1*11*111, E: 111*11*11*1*111, F: 1111*1*1**11*111, G: 111*111*1**1*111, H: (PH seed) 111*11**1*1**1*111, I: 111*1*1**11*1**111, J: 1111**1*1*111*11, K: 111*1111*1**11*1, L: 11111*1*1*11**11, M: 1111**1*111*111, N: 111*1**11*1*1**111, O: 111*1**11*1111*1

majority of seeds have no chance of being optimal and allows us to describe the partition of probability space by optimal seeds. Finally, this method permits us to consider a modified Bernoulli homology model in which only confirmable alignments are considered.

REFERENCES

Brejová, B., Brown, D., and Vinar, T. (2004). Optimal spaced seeds for homologous coding regions. *Journal of Bioinformatics and Computational Biology*, **1**, 595–610.
 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.
 Choi, K. and Zhang, L. (2004). Sensitivity analysis and efficient method for identifying optimal spaced seeds. *Journal of Computer and System Sciences*, **68**, 22–40.

Choi, K., Zeng, F., and Zhang, L. (2004). Good spaced seeds for homology search. *Bioinformatics*, **20**, 1053–1059.
 Li, M., Ma, B., and Zhang, L. (2006). Superiority and complexity of the spaced seeds. In *SODA*, pages 444–453. ACM Press.
 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.
 Mak, D., Gelfand, Y., and Benson, G. (2006). Indel seeds for homology search. *Bioinformatics*, **22**(14), e341–349.
 Noé, L. and Kucherov, G. (2004). Improved hit criteria for DNA local alignment. *BMC Bioinformatics*, **5**, 149–158.
 Pachter, L. and Sturmfels, B. (2004). Parametric inference for biological sequence analysis. *PNAS*, **101**(46), 16138–16143.
 Wolfram Research, I. (2005). *Mathematica: Solve function*. Version 5.2. Wolfram Research, Inc.

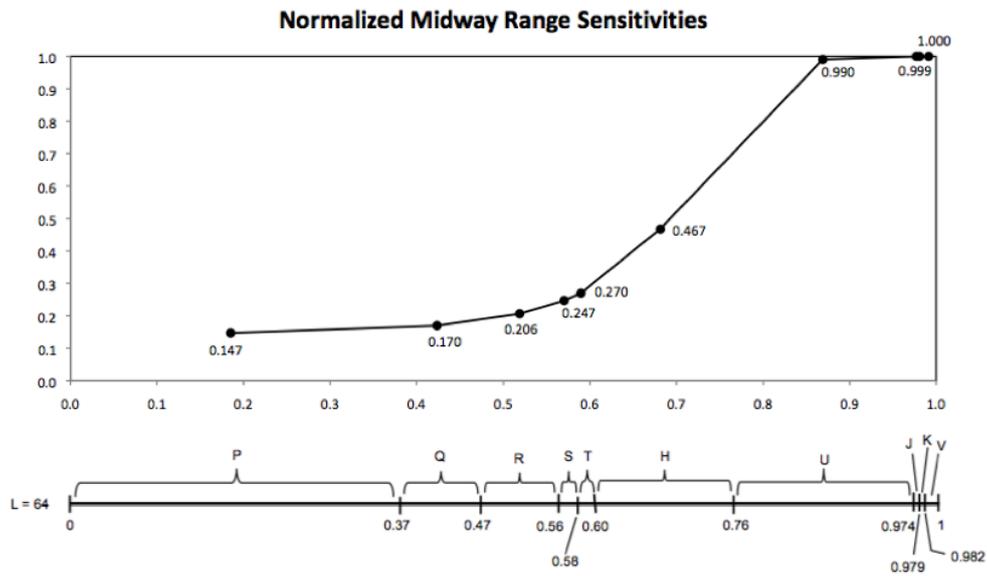


Fig. 6. Confirmable Alignments. Bottom: Probability intervals for optimal seeds of weight 11 (and 11 or fewer wildcards) at region length 64 under the assumption of confirmable alignments. P: 1**1****11***1*11111 Q: 1**1**1*1***11**11111 R: 11****1*1**1**11111 S: 11**1*1***11**1*1111 T: 11*1*1**1**11**1111 H: 111*1**1*1**11*111 U: 111*1*1**11*1**111 J: 11*1111*1**1111 K: 111*1111*1**11*1 V: 11111*1*1*11**11. Top: The graph shows the normalized sensitivity of each optimal seed at the midway point within its optimal interval. Normalization adjusts the sensitivities so that they apply only to confirmable alignments rather than to all representative strings.