PatternHunter II: Highly Sensitive and Fast Homology Search (Extended Abstract)

Ming Li[1] **Bin Ma**[2]
mli@uwaterloo.ca bma@csd.uwo.ca
Derek Kisman[3] **John Tromp**[4]
dkisman@BioinformaticsSolutions.com tromp@cwi.nl

[1] Dept. Computer Science, Univ. of Waterloo, Waterloo, ON, Canada N2L 3G1.
[2] Dept. Computer Science, Univ. of Western Ontario, London, ON, Canada N6A 5B7.
[3] Bioinformatics Solutions Inc., 2B-145 Columbia W., Waterloo, ON, Canada N2L 3L2.
[4] CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands.

Abstract

Extending the single optimized spaced seed of PatternHunter [20] to multiple ones, PatternHunter II simultaneously remedies the lack of sensitivity of Blastn and the lack of speed of Smith-Waterman, for homology search. At Blastn speed, PatternHunter II approaches Smith-Waterman sensitivity, bringing homology search technology back to a full circle.

Keywords: homology search, sensitivity, speed.

1 Introduction

The task of homology search is to find similar segments, or local alignments, between two DNA or protein sequences, measured by match, mismatch and gap scores. Homology search is crucial to biological research and routinely needed by biologists. For example, the NCBI Blast [1] server processes over 10^5 queries a day, and this rate is growing by 10-15% per month. Because of the large sizes of DNA and protein databases, homology search is very time consuming and often needs supercomputers to conduct. Yet, as GenBank doubles in size every 18 months [23] and the list of completed genomes (now including human [14, 28], mouse [22], and rice amongst many other species) expands quickly, the current computational cost is only the tip of the iceberg.

While having made tremendous contributions to science in the past 20 years, the current homology search tools are showing their age. Heuristic searches with members of the Blast family [1, 11, 13, 27] or FASTA [19] are slow for modern genomic data and miss many alignments. Meanwhile, the exhaustive Smith-Waterman methods based on dynamic programming, like SSearch [26, 24], are often too slow to be practical, even with supercomputers. Specialized software, such as MegaBlast [29], BLAT [16], and MUMmer [8], were developed to speed up Blast for highly similar sequences. Many commercial and academic parallel "BlastMachines" were also built to cope with the huge computational load.

PatternHunter [20] is a new generation general purpose homology search tool designed to meet this tremendous need. At Blastn default sensitivity, PatternHunter runs at MegaBlast speed [20]. PatternHunter was used [22] to compare the human genome against the mouse genome at a speed over a hundred times faster than Blastn at the same sensitivity. It uses novel approaches, including an "optimized spaced seed", to substantially improve sensitivity and speed simultaneously. Nonetheless, a last piece of the puzzle remains to be settled: to achieve 100% sensitivity, Smith-Waterman dynamic programming is still the only option, but it is way too slow. The design goal of PatternHunter II is to solve this sensitivity problem: PatternHunter II aims to achieve a sensitivity approaching that of Smith-Waterman with a speed similar to the default Blastn.

One new idea in PatternHunter [20] was the introduction of an "optimized spaced seed". In Blast, exact matches of k continuous letters is used as a "seed" to find long matches around it, whereas in PatternHunter, a seed is k discontinuous letter matches, where the relative positions of the k letters are optimized in advance. This has helped PatternHunter to significantly increase its sensitivity over Blast. It was noticed in [20] that more spaced seeds will help increase sensitivity further and at low cost. This fact was also recently independently noticed and investigated by Buhler, Keich and Sun [5] and Brejova, Brown and Vinar [3]. Earlier, not in the context of Blast-type homology search, non-optimal or randomized multiple spaced patterns were studied by Pevzner and Waterman in [25] as multiple filtration techniques, used in FLASH system by Califano and Rigoutsos [6], and to cover a region with near certainty by multiple randomized hash functions by Buhler [4].

Two problems had postponed us from implementing multiple optimal spaced seeds in the original PatternHunter: large memory requirements for multiple hashtables and the difficulty of finding the optimal seed combination. Since [20], many researchers have recently further studied various aspects of spaced seeds [2, 3, 15, 5, 7] and given exponential or heuristic algorithms for computing the optimal seed(s). However, the complexity of finding optimal spaced seeds remains open. Even the complexity of computing the hit probability of given seeds is unknown.

This paper describes PatternHunter II which implements the optimized multiple seed scheme for increased sensitivity. We give a new greedy method for finding near optimal multiple seeds. We generalize the dynamic programming in [15] to compute the hit probability for k seeds. We describe how we handle multiple hashtables and memory issues. We then study the theoretical performance of multiple seeds and compare the practical performance of PatternHunter II, Blastn, and SSearch (Smith-Waterman dynamic programming).

The complexity questions of the optimal spaced seeds have not escaped our attention. We take this chance to settle these open questions in Section 3 of this paper:

- Given k seeds, computing the hit probability under the uniform distribution is NP-hard.
- There is a provably good polynomial time approximation to compute the hit probability for k given seeds, with arbitrary precision.
- The problem of finding k optimal seeds is NP-hard. It cannot be approximated within ratio $\frac{\mathbf{e}-1}{\mathbf{e}}$.
- The problem of finding even one optimal seed is NP-hard.

2 Optimized Multiple Spaced Seeds and PatternHunter II

Following [20], we denote a spaced seed as a binary string. Let a be a seed. The length of a is denoted by $|a|$, while $\|a\|$ denotes the weight of a, i.e. the number of 1's in a. Intuitively, a 1 in the seed corresponds to a required match whereas a 0 means "don't care". Since the 0s serve only to spread the 1s, we restrict attention to seeds whose first and last bits are both 1. For example, the Blastn default seed is 11111111111, or 11 consecutive matches to generate a hit. PatternHunter's default seed 111010010100110111 has weight 11 and length 18. Every seed corresponds to a *regular expression* obtained from it by replacing each 0 by $(0+1)$.

A homologous region is likewise represented by a binary string, with a 1 representing a match and a 0 representing a mismatch. A substring from position i up to (but excluding) j is denoted by $R[i:j]$. Thus $R = R[0:|R|]$. We say a seed *hits* a homologous region if the corresponding regular expression matches anywhere within the region. Or formally, R has a substring $R[j:j+|a|]$, such that $R[j+i] = 1$ whenever $a[i] = 1$ for $0 \le i < |a|$. We also say that a hits R at position j. If a hits R at position $|R| - |a|$ then we say that a hits the tail of R. Let $A = \{a_1, \ldots, a_k\}$ be k seeds. A hits a region R if there is an $a_i \in A$ that hits R. We say that the random region R is uniformly distributed if $Pr(R[i] = 1) = p$ for any $0 \le i < |R|$. In general, if a region has $p = x\%$ identities, then p is called the similarity level of R.

Under any given distribution of R, the hit probability of multiple seeds is obviously no less than the hit probability of any one of them. Therefore, multiple seeds will increase the sensitivity of the homology search. However, the search program must then examine all the hits generated by all of the seeds, which will in turn slow down the search speed. As explained in [20], another way to increase the sensitivity by sacrificing speed is to decrease the weight of a seed.

Proper tradeoffs between these two approaches lead to the promised fast and sensitive PatternHunter II. We first show how to evaluate the hit probabilities of a given k seeds and how to find a near optimal seed set. Then we proceed to PatternHunter II design and performance analysis.

2.1 Computing Hit Probability of Multiple Seeds

In the original PatternHunter and in [15], a dynamic programming algorithm was used to compute the exact hit probability of a single seed. In this section, we extend the algorithm to compute the hit probability of multiple seeds.

Let $A = \{a_1, \ldots, a_k\}$ be a set of k seeds and R a random region of length L with similarity level p. For a binary string b and $|b| \leq i \leq L$, we define

$$f(i, b) = \Pr(A \text{ hits } R[0:i] | b \text{ is a suffix of } R[0:i]).$$

The hit probability of A on R is then equal to $f(L, \epsilon)$, where ϵ is the empty string. Note that for any $i > |b|$,

$$f(i, b) = (1-p)f(i, 0b) + pf(i, 1b).$$

We'll try to compute $f(i, b)$ in terms of other $f(i', b')$ computed earlier, and limit the set of b's we need to consider in the process.

If a suffix b of a region R is itself hit by A, then $f(i, b) = 1$. We call a binary string b *compatible* with a seed a if $b[|b| - j] = 1$ whenever $a[|a| - j] = 1$ for $0 < j \leq \min(|a|, |b|)$. Clearly, if a suffix b of a region R is not compatible with a, then a cannot hit the tail of R. This leads us to

Definition 1 *Let B be the set of binary strings that are not hit by A but compatible with some $a \in A$. Let $B(x)$ denote the longest proper prefix of x that is in B.*

Note that $\epsilon \in B$. Suppose $b \in B$. Then b is compatible with some $a \in A$, and therefore, so is $1b$. If $1b \notin B$, then it must be hit by some $a' \in A$, and $f(i, 1b) = 1$. If $0b \notin B$, then (assuming every seed starts with a '1') it cannot be hit by A, and therefore must be incompatible with every $a' \in A$. In that case $f(i, 0b)$ equals $f(i - |b| + |b'|, 0b')$, where $0b'$ equals $B(0b)$. This shows that $f(i, b)$ can be computed by dynamic programming as follows:

Algorithm DP

Input A seed set A, similarity level p and length L.

Output The probability that A hits a p-random region of length L.

compute the compatible suffix set B
for i from 0 to L do
  for b in B from longest to shortest do
    if $i < |b|$
      $f[i, b] := 0$
    else
      $f_0 := f[i - |b| + |b'|, 0b']$, where $0b' = B(0b)$
      if A hits $1b$
        then $f_1 := 1$
        else $f_1 := f[i, 1b]$
      $f[i, b] := (1-p) \times f_0 + p \times f_1$
output $f[L, \epsilon]$.

Theorem 1 *Let A be a set of seeds and R be a random region, Algorithm DP computes* $\Pr(A \text{ hits } R)$ *correctly.*

Denote the maximum length of a seed in A by M. Both lines 7 and 8 can be precomputed in time $O((|A|+M) \times |B|)$. For any string x, denote the reverse string of x by x^r. We start by building a trie T_A for the reverse seeds. Then we build the trie T_B for reversed strings in B, complete with suffix links $B'(x)$ representing function $(B(x^r))^r$, layer by layer. Suppose that layers $0, \dots, d$ have been computed. For each $a \in A$ in turn, we traverse the constructed part of T_B compatible with a. At each leaf x, we create—if not already present—compatible children in layer $d+1$. That is, if bit d of a^r is 0, we create both a 0- and a 1-child, while if that bit is 1, we only create the 1-child. If this was the final bit of a, we mark the new nodes as terminal, which signals that it is hit by A, and prevented from having children. The suffix link $B'(x1)$ is simply set to $(B'(x))1$, the 1-child of node $B'(x)$, which must exist, since $B'(x)$ has depth less than d. Node $B'(x)$ need not have a 0-child, so to find $B'(x0)$ we must keep following suffix links, upto M times, until we either get to the root or find a node y that does have a 0-child. Then we set $B'(x0)$ to *epsilon* or $y0$, respectively. Setting a suffix link from x to a terminal node makes x itself terminal, and in this way we find all instances of $1b$ being hit by A, for any $b \in B$. The time needed to construct layer d is $O((|A|+M) \times |B^{=d}|)$. Summing over all layers gives a precomputation time complexity of $O((|A|+M) \times |B|)$. Therefore, the total time complexity of the algorithm is $O(((|A|+M+L) \times |B|)$, with $|B| \leq \sum_{a \in A} |a| \times 2^{|a|-\|a\|}$.

We note that a similar method can be used to compute the k-hit probability. A seed set k-hits a region R if R is hit by k different combinations of seed and position. Similarly to the 1-hit probability computation, we use $f[i,b,k]$ to denote the k-hit probability for region $R[0:i]$ with suffix b. Now suppose we already know $f[i,b,k-1]$, in order to compute $f[i,b,k]$, we need only modify the following lines of Algorithm DP:

Algorithm DP-k-hits	
5.	$f[i,b,k] := 0$
7.	$f_0 := f[i-|b|+|b'|, 0b', k]$, where $0b' = B(0b)$
9.	then $f_1 := f[i-|b|+|b'|, 1b', k-1]$, where $1b' = B(1b)$
10.	else $f_1 := f[i, 1b, k]$
11.	$f[i,b,k] := (1-p) \times f_0 + p \times f_1$

Correctness follows from the observation that the probability of a region—the tail of which is hit—to have at least k hits, equals the probability that the region without its tailing bit has at least $k-1$ hits.

Algorithm DP and DP-k-hits can also be extended to compute the hit probability of random regions with more involved distributions. For example, Brejova, Brown, and Vinar [2] studied an $M^{(3)}$ distribution of coding region homologies. Because of the codon period of the coding regions, in an $M^{(3)}$ distribution, the similarities at positions $3i+k$ are p_k, $k = 0,1,2$. In this paper, we will use a three mer (p_0, p_1, p_2) to denote the parameters of the $M^{(3)}$ distribution. More generally, suppose the similarity at position i is p_i, $i = 0, \dots, L-1$. To compute the hit probability for such a random region, the only change is to replace p's with appropriate p_j's in line 11 of each algorithm.

Another distribution studied in [2] is $M^{(8)}$, which is a Hidden Markov Model (HMM) with 8 parameters. Another algorithm, which is an extension of the hit probability computation algorithm in [15], is also introduced in [2] to compute the single hit probability in a random region of an HMM distribution. Without giving the details, we notice that a similar method as used in [2] can extend Algorithm DP and Algorithm DP-k-hits to compute the multiple seed hit probability under an HMM distribution.

We straightforwardly implemented Algorithm DP in a Java program. Using a Pentium IV 3GHz PC, it took 0.70 seconds to compute the hit probability for a set of 16 weight-11 seeds with length $\leq$ 21, on a random region with length 64. This was reduced to 0.37 seconds when the weight was changed to 12, showing that the running time of the algorithm largely depends on the maximum number of zeros in every seed. Reducing the number by one will approximately half the running time.

2.2 Finding a Good Seed Set Greedily

Enumerating all possible sets and evaluate them by Algorithm DP in Section 2.1 is clearly not feasible because of the exponential number of possible sets. Instead, we now show to how construct a good set of seeds in a greedy fashion. That is, we compute the first seed a_1 which maximizes the hit probability of $\{a_1\}$. Then, fixing a_1, we compute the second seed a_2 so as to maximize the hit probability of $\{a_1, a_2\}$. We continue in this manner until the desired number of seeds or the desired hit probability is reached.

Although such a "greedy" seed set may not optimize the combined hit probability, in some sense a greedy seed set is even more desirable than an optimal one: One may first want to do a single seed comparison for a first impression, and only much later, if ever, decide to try another seed in order to find those alignments missed by the first.

It turns out that Algorithm DP is still efficient enough to be used to compute a practical set of weight 11 seeds greedily—it took 12 CPU days for a Pentium IV 3GHz PC to compute a set of 16 weight 11 seeds, each being no longer than 21. When the random region has length 64 and similarity 70%, the first four seeds are: 111010010100110111, 1111001100101000001011, 11010000110001010111, 1110111010001111.

We note that the optimization is done under a general assumption on the length and similarity regions of homology regions, and does not depend on any specific database and query sequence. Consequently, such a computation can be conducted only once in the development phase of a homology search program, and 12 CPU days are acceptable. However, it would take a much longer time if we want to compute a set of even slightly longer seeds. In such a case, we propose a different approach in the following.

Suppose we have already computed a set A with N seeds using greedy, and C is the candidate set for the $(N+1)$-st seed. The hit probability of $A \cup \{a\}$ for each $a \in C$ can be estimated with k random region samples. According to Theorem 6, accuracy of the estimate goes up with k. So, however, does the computing time. Therefore, we used the following strategy to compute the $(N+1)$-st seed: Let C be the set of all candidate seeds, and k be a reasonably large number such as 500. We estimate the hit probability of $A \cup \{a\}$ for each $a \in C$ by k random sample regions, remove the worst performing half of the seeds from C, and increase k to $2k$. This process is repeated until only one seed is left in C.

Using such a heuristic, we computed a set of 16 weight-12 seeds, each is no longer than 22. The first four seeds are 111011001011010111, 1111000100010011010111, 1100110100101000110111, and 1110100011110010001101.

Clearly, our greedy algorithm and the heuristic to compute the seeds can also be used to compute a good seed set for other distributions. For example, for the homology search of coding regions, we used the $M^{(3)}$ distribution $(0.8, 0.8, 0.5)$ to compute a good set of seeds. This preserves the 70% overall similarity that we've focussed on, while recognizing that the first two bases of a codon are more conserved than the third one. In later sections of this paper, we will refer to the seeds optimized for length 64 random regions with 70% similarity as the general purpose seeds, and refer to the seeds optimized for the length 64 regions with $M^{(3)}$ distribution $(0.8, 0.8, 0.5)$ as the coding region seeds.

2.3 The Performance of the Seeds

PatternHunter's spaced seeds facilitate two ways to increase the sensitivity in homology searches: by increasing the number of seeds, and by reducing the weight of a single seed. Both will increase the running time because more random hits will be generated. Here we compare the performances of these two ways.

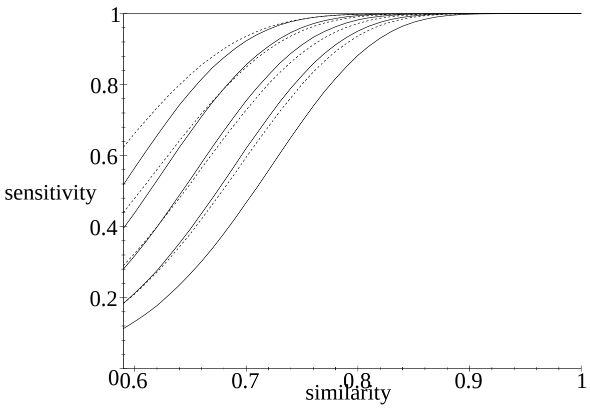

Figure 1: Recall that "similarity" is the percentage of identities in the homology region, and "sensitivity" is the probability of having a hit, in the given region. From low to high, the solid curves are the hit probabilities of using the first k ($k = 1, 2, 4, 8, 16$) weight-11 general purpose seeds, respectively. The dashed curves are the hit probabilities of using the single optimal weight w ($w = 10, 9, 8, 7$) seeds, respectively.

Figure 1 compares the hit probabilities of multiple weight 11 seeds and a single weight < 11 seed. For a random region with length 64 and similarity level x, the hit probabilities of different seed configurations were computed with Algorithm DP. From the figure we can see that the sensitivity is approximately equally improved by doubling the number of seeds. Also, at high similarity levels, doubling the number of the seeds achieves better sensitivity than reducing the weight of the single seed by one. We have observed similar phenomena for weight-12 general purpose seeds. The figure for weight-12 seeds can be found in [18].

According to a lemma in [20], reducing the weight by one increases the expected number of hits by a factor of four (the alphabet size) in DNA homology search. Doubling the number of the seeds, however, only increases the expected number of hits by a factor of two. Therefore, we conclude that using multiple seeds is the preferred way to gain sensitivity in homology search.

Figure 1 also tells us that at an 80% similarity level, 8 or 16 weight-11 seeds can achieve near 100% hit probability on a length-64 random region. For longer regions, the hit probabilities are even higher.

2.4 PatternHunter II Design

PatternHunter II follows the design of the original PatternHunter [20]. That is, for a given weight W length L seed, a hashtable is built for the subject sequences. For each length L substring s of the query sequences, the hashtable provides a way to efficiently retrieve all the hits of s in the subject sequences. Then a gapped extension is performed for each of the hits to find local alignments. In two-hit mode, a gapped extension is performed only if two nearby hits are found on the same diagonal.

A major change in PatternHunter II is the use of multiple seeds. Accordingly, a hashtable is built for each of the given seeds. For each substring of the query sequences, all hits generated from all hashtables are used for the gapped extensions. Also, in two-hit mode, the two nearby hits can be from different hashtables. If not otherwise specified, all performance tests of PatternHunter II in this paper uses the one-hit mode.

The hashtable in PatternHunter consists of an entry table with 4^W entries, and a linked list which is an array with n elements, n being the total size of the subject sequences. All of the entries and elements are represented by 32-bit integers. Therefore, if we use k seeds in PatternHunter II, the hashtables would occupy $4(4^W k + nk)$ bytes. This is still feasible in a medium size homology search, e.g., with $k = 8$, $W = 11$, and $n = 32 \times 10^6$, the hashtables use about 256Mbyte of memory.

However, for very large n, memory requirements exceed the capacity of a desktop computer. In such a case, PatternHunter II divides the very large subject sequences into several smaller segments, and each of the smaller segments is searched in turn. This possibly breaks an alignment into two parts at a division boundary. However, because an alignment is usually much shorter than a segment, the chance of breaking an alignment is low. Moreover, PatternHunter has a mechanism to extend an alignment across the division boundary, and also tries to divide a long DNA sequence at the regions that have long series of letter `N`s (the letters not yet determined by DNA sequencing). All these methods minimize the risk of losing alignments. Given the input and the size of memory, the division of the subject sequence is decided by PatternHunter II and is transparent to the users.

2.5 PatternHunter II Performance

We conducted the sensitivity benchmark of PatternHunter by comparing its performance with that of Blast and the Smith-Waterman algorithm.

The Smith-Waterman implementation that we used is SSearch [24], a subprogram in the FASTA package. The complete FASTA package is downloadable at ftp://ftp.virginia.edu/pub/fasta/. The DNA sequences we used are two sets of human and mouse EST sequences, downloaded from NCBI's ftp site, ftp://ftp.ncbi.nlm.nih.gov/blast/db/FASTA/. The month.est_human.Z and month.est_mouse.Z files at the site contain all new or revised human and mouse EST sequences released in the last 30 days, respectively. We have downloaded the two files released on April 14, 2003. There were 29715 mouse EST sequences and 4407 human EST sequences in these two files.

Due to the fact that there are many long sequences of identical letters, especially long sequences of `A`s and `T`s, the Smith-Waterman algorithm generates too many junk alignments of these sequences. To avoid this, we did a trivial "repeat masking" by turnning all those sequences of ten or more repetitive letters to letter `N`s. To ensure a fair comparison, all of PatternHunter, Blast, and SSearch are fed with the same masked data. Each program uses a score scheme equivalent to: match $= 1$, mismatch $= -1$, gapopen $= -5$, gapextension $= -1$. Only those local alignments with scores no less than 16 are recorded and taken into account.

It took more than 20 CPU days for the SSearch program to align each of the mouse EST sequences with each of the human EST sequences. All pairs of ESTs that contain a local alignment with score equal to or higher than 16 were recorded. If a pair of ESTs has more than two local alignments, only the one with the highest score was considered. In total, 3346700 pairs were found and the maximum local alignment score is 694. All the benchmark data sets and computation results can be found at http://www.bioinformaticssolutions.com/ph/benchmark.html.

Then we ran both PatternHunter II and Blastn to compare the two EST files, and checked how many of the pairs found by SSearch can also be found by them. Because neither PatternHunter nor Blast tries to compute the optimal alignments for the homologies they have found, if SSearch finds a local alignment with score x for a pair of ESTs, we regard the pair as "found" by PatternHunter II (or Blast) if it finds a local alignment of score $\geq \frac{x}{2}$ for the same pair of ESTs.

Suppose Smith-Waterman finds y pairs of ESTs with local alignment score at least x, and y' of the y pairs can also be found by another program with alignment score at least $\frac{x}{2}$. As Smith-Waterman is "lossless", the ratio $\frac{y'}{y}$ can be considered as the sensitivity of the other program at the alignment score x. If the other program is also "lossless", this ratio would be equal to 1.

Figures 2 and 3 compare the sensitivity of Blastn and different configurations of PatternHunter II. The sensitivity versus alignment score curves for Blastn and PatternHunter II with coding region weight 11 seeds is displayed in Figure 2. And Figure 3 compares Blastn, PatternHunter II with both coding and general purpose seeds. In order to display better, Figure 3 only focuses on the region where the alignment score is no more than 50 and the sensitivity exceeds 90%.

The following table lists the running time of different programs, with weight 11 seeds for Blastn and PatternHunter, on a Pentium IV 3GHz Linux PC:

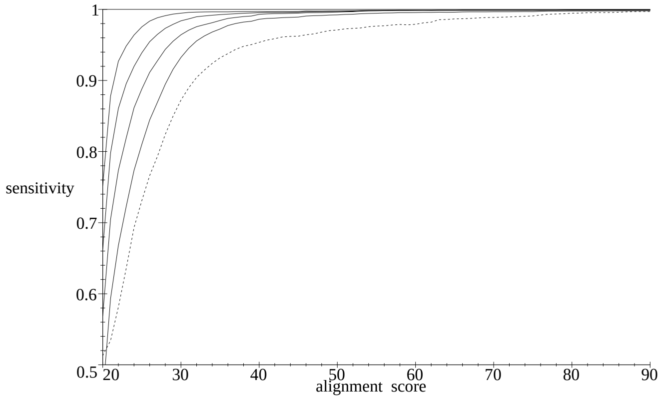

Figure 2: The dashed curve is the sensitivity of Blastn, seed weight 11. From low to high, the solid curves are the sensitivity of PatternHunter II using 1, 2, 4, and 8 weight 11 coding region seeds, respectively.

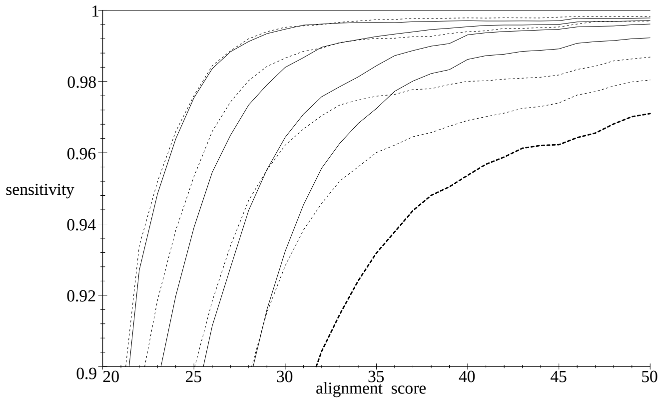

Figure 3: The thick dashed curve is the sensitivity of Blastn, seed weight 11. From low to high, the solid curves are the sensitivity of PatternHunter II using 1, 2, 4, and 8 weight 11 coding region seeds, and the thin dashed curves are the sensitivity of PatternHunter II using 1, 2, 4, and 8 weight 11 general purpose seeds, respectively.

SSearch	Blastn	PatternHunter II				
20 days	575 s	seeds	1	2	4	8
		general	242 s	381 s	647 s	1027 s
		coding	214s	357s	575s	996s

This benchmark demonstrates that PatternHunter II can achieve much higher sensitivity than Blastn at much faster speeds. Furthermore, PatternHunter II with 4 coding region seeds runs at the same speed as Blastn and 2880 times faster than SSearch, but with a sensitivity approaching the latter.

The above table and Figure 3 also confirms a result of [2], that the coding region seeds not only run faster, because there are less irrelevant hits, but are also more sensitive than the general purpose seeds. This is not a surprise because the EST sequences are coding regions.

2.6 Comparison with Other Seeds

The authors of [5] and [2] have also designed some seeds. In this section we briefly compare their seeds with the seeds used in PatternHunter II.

A "Mandala" system is used in [5] to design multiple spaced seeds. Since the paper only provided two weight 12 noncoding seeds $\pi_1 = 111110010000100011111$ and $\pi_2 = 111011101101111$ found by Mandala, we have also used the first two PatternHunter II general purpose weight 12 seeds

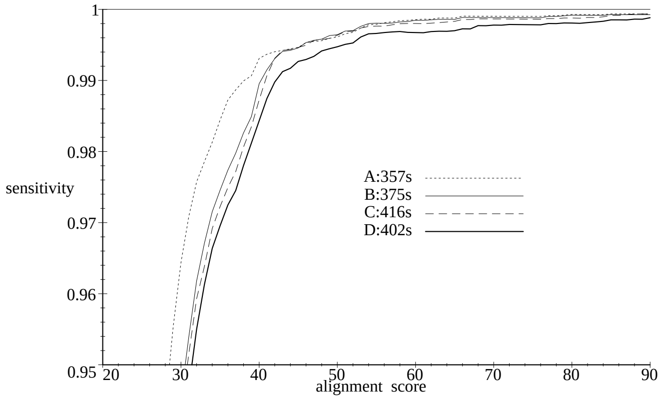

Figure 4: A: PH II's two weight 11 coding region seeds. B: PH II's one weight 10 coding region seed 1101100101000101101. C: $M^{(8)}$ seed 11011011000011011 in [2]. D: $M^{(3)}$ seed 11001011001011011 in [2].

111011001011010111 and 1111000100010011010111, for a fair comparison. Using the same EST data as above, our result showed that Mandala seeds $\pi_1 + \pi_2$ find fewer significant alignments than our two weight 12 seeds, both using PatternHunter II as the main program. The Mandala seeds found more insignificant alignments with scores less than 24, using 284 seconds, and the PatternHunter II seeds found more alignments with scores greater than 24, using 256 seconds.

Using a fifth order Markov model to model the coding regions, the authors of [5] also used their Mandala system to design a weight 11 coding region seed $\pi_{C_5} = 1110000011011011011$. Also, another weight 11 coding region seed $\pi_{sp} = 1101100001101101101$ is designed by manually modifying the seed π_{C_5}. We compared the performance of π_{C_5}, π_{sp} and PH II's weight 11 coding region seed, when used in PH II. The result showed that the seed π_{C_5} designed by their Mandala system is noticeably worse than the other two seeds, while the performances of PH II's seed and their manually designed π_{sp} seed are almost identical. A figure illustrating this comparison can be found in [18].

Brejova, Brown, and Vinar [2] proposed an HMM model $M^{(8)}$ for spaced seeds and demonstrated its clear advantage on coding regions. Using the same EST sequence data, Figure 4 provides a comparison of PH II's two weight 11 coding regions seeds, PH II's single weight 10 coding region seed 1101100101000101101, the $M^{(8)}$ seed 11011011000011011 in [2], and the $M^{(3)}$ seed 11001011001011011 in [2]. Observe that a single weight 10 seed is less sensitive than two weight 11 seeds, Figure 4. The weight 10 $M^{(8)}$ seed also runs slower using 416 seconds on our data, while two weight 11 PH II seeds used only 357 seconds. Intuitively, at a weight one greater, a seed is expected to have only a quarter of the hits, hence fourfold speedup. Two of them should give a twofold speedup, while having the better sensitivity as shown in Figure 4. However, when the query and the subject sequences are not both large enough, other overheads diminish the gain.

All PH II's coding region seeds are computed using the $M^{(3)}$ model with a $(0.8, 0.8, 0.5)$ distribution pattern. We have also tried to compute a set of seeds, S_{HMM}, using the $M^{(8)}$ model provided in [2]. Using the same EST benchmark data, our results showed that the multiple seeds computed with $M^{(3)}$ $(0.8, 0.8, 0.5)$ and the seeds computed with $M^{(8)}$ model perform approximately the same. A figure that illustrates the performances can be found in [18].

The above comparisons reveal that although the more sophisticated model of the coding regions provided remarkable improvement on the seed performance in [5] and [2], this improvement does not necessarily carry over to other data sets (especially, other species). Specifically, for weight 10, the $M^{(8)}$ seed[1] of [2] is slightly worse than the weight 10 coding region seed we have computed using their simpler model $M^{(3)}$ with very intuitive parameters $(0.8, 0.8, 0.5)$.

[1] While this seed was obtained in [2] by limiting number of 0's to 7, we have verified that it is still optimal under their $M^{(8)}$ model when the number of 0's is limited to 11.

3 The Complexity of Finding Optimal Spaced Seeds

Many authors [20, 2, 3, 15, 5, 7] have proposed heuristic or exponential time algorithms for the general seed selection problem: find one or many optimal spaced seeds so that a maximum number of target regions are each hit by at least one seed. A seemingly simpler problem is to compute the hit probability of k given seeds. In this section, we show that these are all NP-hard problems. This gives confidence that the greedy algorithm in Section 2.2 and the exponential time algorithm in Section 2.1 are perhaps the best one can do. We will also give a provably-good approximation algorithm to compute the hit probability of k seeds. In particular, letting $f(n)$ be the maximum number of 0's in each seed, where n is the seed length, the following are true.

1. If $f(n) = O(\log n)$, then the algorithm in Section 2.1 computes the hit probability of k seeds in polynomial time; otherwise the problem is NP-hard, Section 3.1.
2. If $f(n) = O(1)$, one or a constant number of optimal seeds can be computed in polynomial time by enumerating all seed combinations and use Item 1 to compute their probabilities; otherwise, even selecting one optimal seed is NP-hard, Section 3.3.
3. If $f(n) = O(1)$, then the greedy algorithm in Section 2.2 for finding k seeds approximates the optimal solution within ratio $1 - \frac{1}{\mathbf{e}}$ in polynomial time, as this is implied by the greedy bound for the maximum coverage problem [12]; otherwise the problem cannot be approximated within ratio $1 - \frac{1}{\mathbf{e}} + \epsilon$ for any $\epsilon > 0$, Section 3.2.

Because of the space limit, we omit all the proofs in this section. However, the proofs can be found in [18].

3.1 Computing the Hit Probability of Multiple Seeds Is NP-hard

Theorem 2 *Computing the hit probability of many seeds on a uniformly distributed random region is NP-hard.*

3.2 The Hardness of Finding the Optimal Seed Set

The NP-hardness of the hit probability computation (Theorem 2) does not imply the hardness of finding the optimal seed set. In this section, we prove that finding the optimal seed set is hard to approximate when the random homologous regions are not uniformly distributed. More specifically, we prove that approximating the following problem is hard:

Region Specific Optimal Seeds
Let $R_1, \ldots, R_m$ be m homologous regions of length L. Find k seeds with weight W and length $\leq M$ hitting the maximum number of homologous regions.

Theorem 3 *The Region Specific Optimal Seeds problem cannot be approximated within ratio* $1 - \frac{1}{\mathbf{e}} + \epsilon$.

Corollary 4 *Removing length constraints on the seeds preserves NP-hardness of the Region Specific Optimal Seed Set problem.*

3.3 The Hardness of Finding One Optimal Seed

Theorem 5 *The problem of finding a seed of given weight and length that hits each of a given set of homology regions, is NP-hard. Consequently, the problem of finding an optimal seed of given weight and length that hits the maximum number of given set of homology regions, is NP-hard.*

3.4 A PTAS for Computing the Hit Probability

In Section 2.1 we presented a dynamic programming algorithm for accurate hit probability computation. However, the dynamic programming algorithm runs in time exponential in $|a| - \|a\|$, which in many cases makes it infeasible. In this section we present a PTAS (polynomial time approximation scheme) to compute the hit probability approximately. It is noteworthy that in this section we do not make any assumption on the distribution of the random regions.

Our algorithm is simple: randomly sampling m homologous regions, and using the hit frequency of the seed set on the m regions as the approximation to the hit probability. When this m is sufficient large, the computation is sufficient accurate, i.e., we have the following

Theorem 6 *Let A be a set of seeds. Let R be a random region under certain distribution and $R_1, \ldots, R_m$ be m independent samples of the random region. Let p be the probability that A hits R and m' of the m samples are hit by A. Then for any $0 < \delta < 1$, $\Pr(|p - \frac{m'}{m}| > \delta) \leq 2\exp(-\frac{m\delta^2}{3})$. Consequently, if $m > \frac{3\log K}{\delta^2}$, then with probability greater than $1 - \frac{2}{K}$, $|p - \frac{m'}{m}| \leq \delta$.*

4 Conclusion

Homology search is a very time-consuming computational task. Due to the small query and database sizes in protein-protein searches, the bottleneck is with DNA-DNA (Blastn) searches, translated DNA-protein (tBlastx) searches, and the exhaustive Smith-Waterman computation.

Using optimized spaced seeds and new algorithms, PatternHunter speeds up Blastn by 5-100 times, depending on data size [20, 22], at the same sensitivity.

Using optimized multiple spaced seeds, PatternHunter II is over a thousand times faster than Smith-Waterman at approximately the same sensitivity, for DNA sequence search.

We are currently investigating new multiple optimal seed schemes to approximate the Smith-Waterman sensitivity for protein-protein searches, at Blastp speed. We are also developing the translated PatternHunter that aims to speed up tBlastx.

References

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E., and Lipman, D.J., Basic local alignment search tool, *J. Mol. Biol.*, **215**, 403–410, 1990.

[2] Brejova, B., Brown, D., and Vinar, T., Optimal spaced seeds for homologous coding regions, *CPM*, 42–54, 2003.

[3] Brejova, B., Brown, D., and Vinar, T., Vector seeds: an extension to spaced seeds allows substantial improvements in sensitivity and specificity. *WABI*, 39–54, 2003.

[4] Buhler, J., Efficient large-scale sequence comparison by locality-sensitive hashing, *Bioinformatics*, **17**, 419–428, 2001.

[5] Buhler, J., Keich, U., and Sun, Y., Designing seeds for similarity search in genomic DNA, *RECOMB*, 2003.

[6] Califano, A. and Rigoutsos, I., FLASH: fast look-up algorithm for string homology, *Tech. Rep.*, IBM T.J. Watson Research Center, 1995.

[7] Choi, K.P. and Zhang, L., Sensitive analysis and efficient method for identifying optimal spaced seeds, To appear in *Journal of Computer and System Sciences*, 2003.

[8] Delcher, A.L., Kasif, S., Fleischmann, R.D., Peterson, J., White, O., and Salzberg, S.L., Alignment of whole genomes, *Nucleic Acids Research* **27**:11, 2369–2376, 1999.

[9] Feige, U., A threshold of $\ln n$ for approximating set cover, *Journal of the ACM* **45**:4, 634–652, 1998.

[10] Garey, M. and Johnson, D., *A guide to the theory of NP-completeness*, W.H. Freeman, 1979.

[11] Gish, W. WU-Blast. At http://blast.wustl.edu.

[12] Hochbaum, D.S., *Approximation Algorithms for NP-hard Problems*, PWS Publishing Company, 1997.

[13] Huang, X. and Miller, W., A Time-efficient, Linear-Space Local Similarity Algorithm, *Advances in Applied Mathematics* **12**, 337–357, 1991.

[14] Human Genome Sequencing Consortium, Initial sequencing and analysis of the human genome, *Nature* **409**, 860–921, 2001.

[15] Keich, U., Li, M., Ma, B., and Tromp, J., On spaced seeds for similarity search, To appear in *Discrete Applied Math*, 2003.

[16] Kent, W.J., BLAT: the Blast-like alignment tool, *Genome Research*, **12**, 656-664, 2002.

[17] Li, M., Ma, B., and Wang, L., On the closest string and substring problems, *J. ACM*, **49**:2, 157–171, 2002.

[18] Li, M., Ma, B., Kisman, D., and Tromp, J., PatternHunter II: Highly Sensitive and Fast Homology Search, To appear in *J. Bioinformatics and Computational Biology*, 2003.

[19] Lipman, D.J. and Pearson, W.R., Rapid and sensitive protein similarity searches, *Science*, **227**, 1435–1441, 1985.

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

[21] Motwani, R. and Raghavan, P., *Randomized Algorithms*, Cambridge University Press, 1995.

[22] Mouse Genome Sequencing Consortium, Initial sequencing and comparative analysis of the mouse genome, *Nature*, **420**, 520–562, 2002.

[23] National Center for Biological Information, Growth of GenBank, http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html.

[24] Pearson, W.R., Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms, *Genomics* **11**, 635–650, 1991.

[25] Pevzner, P.A. and Waterman, M.S., Multiple filtration and approximate pattern matching, *Algorithmica* 13(1/2), 135–154, 1995.

[26] Smith, T.F. and Waterman, M.S., Identification of common molecular subsequences, *J. Mol. Biol.* **147**, 195-197, 1981.

[27] States, D., SENSEI, At http://stateslab.wustl.edu/software/sensei/.

[28] Venter, J.C., *et al*, The sequence of the human genome, *Science* **291**, 1304, 2001.

[29] Zhang, Z., Schwartz, S., Wagner, L., and Miller, W., A Greedy Algorithm for Aligning DNA Sequences, *J. Comp. Biol.*, **7**:1-2, 203–214, 2000.