Journal of Bioinformatics and Computational Biology
Vol. 2, No. 4 (2004) 819–842

ICP Imperial College Press
www.icpress.co.uk

A TUTORIAL OF RECENT DEVELOPMENTS IN THE SEEDING OF LOCAL ALIGNMENT

DANIEL G. BROWN*, MING LI† and BIN MA‡

*,† *Department of Computer Science, University of Waterloo, Waterloo, ON, Canada, N2L 3G1*

‡ *Department of Computer Science, University of Western Ontario, London, ON, Canada, N6A 5B7*

* *browndg@math.waterloo.ca*

† *mli@uwaterloo.ca*

‡ *bma@csd.uwo.ca*

Received 9 September 2004
Revised 28 September 2004
Accepted 28 September 2004

We review recent results on local alignment. We begin with a review of classical methods and early heuristic methods, and then focus on more recent work on the seeding of local alignment. We show that these techniques give a vast improvement in both sensitivity and specificity over previous methods, and can achieve sensitivity at the level of classical algorithms while requiring orders of magnitude less runtime.

Keywords: Sequence alignment algorithms; homology search; spaced seeds.

1. Introduction

Two sequences are homologous if they share a common evolutionary ancestry. Unfortunately, this is a hypothesis that usually cannot be verified simply from sequence data alone. Local alignment is a way of identifying regions of sequences that may be homologous. If the similarity of two sequences is very high, it is unlikely to have originated by chance. As such, the hypothesis that they are homologous is supported.

Local alignment methods are important as their product, high scoring alignments, are used in a range of areas, from estimating evolutionary histories, to predicting functions of genes and proteins, to identifying possible drug targets. Contemporary molecular biologists use themroutinely, and this results in substantial load on even supercomputers. The NCBI BLAST server for homology search is queried over 100,000 times a day and this rate is growing by 10–15% per month.

The basic homology search problem is so easy that is usually the first topic in a bioinformatics course. However, the problem is becomes very hard as queries and databases grow in size, and the emphasis is on very efficient algorithms with high quality. More programs have been developed for homology search than for any other

problem in bioinformatics, yet after 30 years of intensive research, key problems in this area are still wide open.

As is true of many topics in early bioinformatics, the first two important sequence alignment algorithms, the Needleman–Wunsch algorithm for global alignment and the Smith–Waterman algorithm for local alignment, were both identified by a variety of different groups of authors, working in different disciplines, during the 1970s and early 1980s. However, these algorithms ran in time that was too slow as databases of DNA and protein sequences grew during the 1980s. Since the mid 1980's and 1990's, heuristic algorithms like FASTA[1] and BLAST,[2] those that sacrificed sensitivity for speed became popular; these algorithms offer far faster performance, while missing some fraction of good sequence homologies. The development of good homology search software took another advance recently, as researchers focused on the cores of alignments that are identified by heuristic search programs. This has been seen in local alignment programs such as PatternHunter[3] which allows substantial improvement in sensitivity at minimal cost. The spaced seeds of PatternHunter, in particular, can be optimized to be highly sensitive for alignments matching a particular model of alignments,[4–8] which allows substantial improvement in the sensitivity. In fact, one can use these spaced models to approach Smith–Waterman sensitivity at BLAST speed.[5]

Our focus in this survey is on these recent advances in seeding of alignments. We briefly review basic dynamic programming algorithms for sequence alignment, and then discuss how an awareness of the structure of hits has given rise to new models and mathematics for representing alignments.

2. Sequence Alignment: Scoring and Simple Dynamic Programming Algorithms

Here, we begin our review of sequene alignment by discussing the classical measures and algorithms used for this problem. These basic algorithms are still used in practice; the goal of heuristic sequence aligners is primarily to call them on as few subregions of the two sequences being aligned as possible, while still keeping high sensitivity to high-scoring alignments.

2.1. *Notation: alignments*

Let $s = s_1 s_2 \cdots s_m$ and $t = t_1 t_2 \cdots t_n$ be two sequences over a finite alphabet Σ. We augment the alphabet Σ with a "space" symbol, denoted by "–", that is not in Σ, yielding the alignment alphabet, Σ'. Any equal-length sequences S and T over alphabet $\Sigma \cup \{-\}$ that result from inserting space characters between letters of s and t are called an alignment of s and t.

Usually, the objective of the sequence alignment of s and t is to maximize the similarity of S and T, by having characters in the same position in both sequences be closely related or identical. A simple measure of the similarity between two sequences s and t is their *edit distance.* This is the minimum number of steps to

transform s to t by single letter substitutions, insertions or deletions. This measure can also be used in computing the score of a sequence alignment. If the cost of an alignment is the number of columns of the alignment that differ in the two sequences, then computing the optimal alignment is easily shown to be the same as computing the edit distance between s and t.

We will mostly focus on the *local alignment* problem, which consists of finding alignments of substrings of s and t whose similarity is high. However, we will begin by focusing on the *global alignment* problem, which aligns full sequences.

2.2. *Scoring of alignments*

It is possible to perform biologically meaningful sequence alignments using only the edit distance. However, typical homology search programs optimize somewhat more complicated functions of the alignments. There is a rich probabilistic basis to these scoring methods, but here we do not review this; see the literature[9–11] for details.

2.2.1. *Score matrices*

The edit distance measure causes the same cost to be incurred for every substitution. Yet, especially for protein sequences, some mutations would be less expected to be found in homologous sequences. A *score matrix* is used to discriminate different types of matches and mismatches. The score $M(a, b)$ is the contribution to the alignment score of aligning a and b, for any $a, b \in \Sigma'$. Given two sequences s and t, and their alignment (S, T), the score of the alignment, is defined by $\sum_{i=1}^{k} M(S[i], T[i])$, where $k = |S| = |T|$. When a score matrix is used, we usually let $M(a, b) > 0$ if a and b are closely related. Therefore, the goal is to find the alignment that maximizes the alignment score $\sum_{i=1}^{k} M(S[i], T[i])$.

2.2.2. *Gap penalties*

Another addition concerns the lengths of regions of entirely space characters in an alignment. In homologous sequences, such "gaps" correspond to a single block deletion or insertion. If we use a typical scoring matrix where we score a negative constant $M(a, -)$ for aligning any letter a to a gap, the cost of the gap will be proportional to its length. In both theory and practice, this is undesirable. One typically penalizes gaps through a length-dependent gap penalty, rather than only through the scoring matrix. The most common gap penalty is affine: the score of a gap of length i is $o + ie$. In this scheme, o, the gap "opening" penalty, is typically much more negative than e, the "extension" penalty paid per gap position.

2.3. *Simple dynamic programming algorithms*

The optimal alignment of two sequences s, of lenth m and t, of lenth n, assuming gap costs proportional to their length, can be computed in time proportional to the

product of their lengths using dynamic programming.[12,13] The key observation is that the subalignment preceding any column of the optimal alignment must itself be optimal; this optimal substructure allows one to use dynamic programming.

The straightforward algorithm also requires $O(mn)$ space to store the entire dynamic programming matrix, but this can be reduced to $O(\min(m, n))$ space by using a trick proposed by Hirschber.[14] Here, we keep track of two rows (or columns) of the dynamic programming matrix, and after we have computed row $m/2$ of the matrix, we also remember which cell in the $m/2$ row the alignment path that is optimal in the current cell passed through last. When we compute this for the $[m, n]$ entry of the *DP* matrix, this allows us to divide the problem in half and recurse on smaller problems, which preserves both the $O(mn)$ runtime and the $O(n)$ space.

To compute the optimal local alignment of two sequences, we can also use a classic dynamic programming algorithm.[15] One way to consider local alignment is to think that eliminating prefixes and suffixes of the two sequences is free. Therefore, if a global alignment has negative scores at either end, we can eliminate that end to get a better local alignment. This simple idea gives rise to the standard dynamic programming technique, which has similar runtime to the global alignment algorithm.

Both of these algorithms can be extended to the case of affine gap costs as well. We keep track of multiple dynamic programming matrices, to ensure that we only pay for the cost of opening a gap once. The overall runtime triples in a simple implementation, but can easily be made only twice the runtime of the algorithm for simpler gap costs.

3. Second Generation Homology Search: Heuristics

Filling in an entire dynamic programming matrix when aligning long sequences is quite time consuming. As a result, in the late 1980s and early 1990s, heuristic methods were proposed. These methods share a common theme: they sacrifice sensitivity for speed. That is, they run much faster than full dynamic programming, but they may miss some good local alignments. The two most popular heuristics are found in FASTA[1] and BLAST.[2]

3.1. *FASTA and BLASTN*

One of the earliest of these heuristics is FASTA. FASTA uses a hashing approach to find all matching k-tuples (between 4 and 6 for DNA), between the query and database. Then nearby k-tuples, separated by a constant distance in both sequences are joined into a short local alignment. With these short local alignments as seeds, Smith–Waterman dynamic programming is applied to larger gaps between two high scoring pairs still separated by short distances, with the restriction that only the part of the dynamic programming matrix nearest the diagonal is filled in. FASTA outputs only one alignment per query sequence, after the dynamic programming phase, and estimates the probability of the alignment occurring by chance.

More popular has been the BLASTN heuristic aligner. BLASTN works similarly at its beginning, identifying seed matches of length k (9–11 bases long). Each seed match is extended to both sides until a drop-off score is reached. Along the way, seed matches that are being extended in ways that are not typical of truly homologous sequences are also thrown out. BLAST can be set so that two nonoverlapping seed matches may be required before alignments are extended. Newer versions of BLASTN[16] allow gapped alignments to be built. BLASTN outputs all alignments found, and estimates for each the expected number of alignments whose score would be as large, if the alignments were of unrelated sequences.

3.2. *BLASTP*

The program BLASTP, which is probably more often used than BLASTN, also performs heuristic alignments, but of protein sequences instead. BLASTP works by beginning with much shorter seeds, of three or four letters in length; for three-letter seeds, the requirement is that the two subsequences being aligned have total pairwise score at least +13; since the scoring matrices used in BLASTP are measured in half-bits, this corresponds to 6.5 bits of information. Subsequent to finding a hit, the hit is extended in its local region, filtered to verify that the region is worth extending, and then extended, in a similar way as for BLASTN.

A big difference between these two programs comes in their density of false-positive hits. If a BLASTN seed is an 11-base exact match, then two random 11-letter sequences will match with probability approximately one in four million. By contrast, two random amino acid sequences, with letter frequencies derived from the BLOSUM62 scoring matrices, will form a BLASTP hit with probability approximately 1/1600. This is done because molecular biologists have been unwilling to suffer low sensitivity in protein alignments: they want to find a much larger fraction of the true alignments. However, as protein databases are typically much smaller than nucleotide sequences, this increase in false positives hits is still relatively tolerable.

4. Next-generation Homology Search Software

In the post-genome era, supercomputers and specialized hardware implementing sequence alignment methods in digital logic are employed to meet the ever expanding needs of researchers. Pharmaceutical corporations and large scientific funding agencies proudly spend much money to support such supercomputing centers. Unfortunately, the reliability of these solutions must be considered in light of the consistent doubling of sequence databases, as GenBank doubles in size every two years.[17]

In the late 1990s, however, several methods were developed that improve the sensitivity of homology search software to a level comparable to that of full-scale dynamic programming, while avoiding the very large runtimes. These have largely

focused on characterizing the central seeds from which heuristic alignment programs build their local alignments.

4.1. *New idea: optimal alignment seeds*

BLAST-like heuristics first find short seed matches which are then extended. This technique faces one key problem: as seeds grow, fewer truly homologous sequences will have the conserved core regions. However, shorter seeds yield more random hits, which significantly slow down the computation.

To resolve this problem, a novel seeding scheme was introduced in PatternHunter.[3] BLAST looks for matches of k consecutive letters as seeds; the default value of k is 11 in BLASTN and 28 in MegaBlast. Instead, PatternHunter uses k *non*-consecutive letters as seeds.The relative positions of the k letters is called a *spaced seed model* (or simply, a *spaced seed*), and k is its *weight*. For convenience, we denote a seed model by a 0-1 string, where ones represent required matches and zeros represent "don't care" positions. For example, if we use the weight 6 model 1110111, then the alignment `actgcct` versus `acttcct` matches the seed, as does `actgcct` versus `actgcct`. In this framework, BLAST can be thought of as using models of the form $111\ldots1$.

Let L be the length of a homologous region with no gaps, and M be the length of a seed model. Then there are $L - M + 1$ positions that the region may contain a hit (Fig. 1). If we use a typical heuristic search scheme, as in BLASTN, as long as there is one hit in such a region, the region can be detected. Therefore, although the hit probability at a specific position is usually low, the probability that a long region contains a hit can be reasonably high.

Ma, Tromp and Li[3] noticed that different seed models with identical weight can have very different probabilities to hit a random homology. For a seed with weight W, the fewer zeros it has, the shorter the seed is, and the more positions it can hit the region at. Therefore, intuitively, BLAST's seed model with W consecutive ones seems to have the highest hit probability among all the weight-W seed models. Quite surprisingly, this is not true. The reason is that the hits at different positions of a region are not independent. For example, using BLAST's seed, if a hit at position i is known, the chance to have a second hit at position $i+1$ is then very high because it requires only one extra base match. The high dependency between the hits at different positions make the detection of homologs "less efficient": many

```
   tactgcctg
   |||| ||||
   tactacctg
1: 1110101
2:  1110101
3:   1110101
```

Fig. 1. There are many positions that a homology may contain a hit. In this figure, the seed model 1110101 hits the region at the second position.

regions will have more than one hit, which is unhelpful, while many other regions will be missed entirely.

The same authors noticed that the dependency can be reduced by adding some zeros into the seed model. For example, if the seed model 1110101 is used, and there is a hit at position i, the hit at the position $i+1$ requires three extra base matches (compared to one extra base match of the BLAST's seed). Thus, hits at different positions are less dependent when spaced seed models are used. On the other hand, spaced seed models are longer than the consecutive seed model with the same weight, and therefore have fewer positions to hit a region at. As a result, the optimal seed must balance these two factors. In the same paper, the authors developed a method to find the optimal seed model that maximizes the hit probability in a simple model, and the optimal seeds were used to develop the PatternHunter program, among other things.

Some other seeding or related strategies have also been developed before or after PatternHunter's spaced seed model. In his program WABA,[18] Kent proposed the use of a simple pattern in identifying homologous coding sequences. Since these sequences often vary in the third, "wobble" position of codons, WABA ignores these positions when identifying positions that match a seed. In the spaced seed framework of PatternHunter, this is equivalent to using spaced seeds of the form $110110\ldots$.

Kent's approach takes advantage of the special properties of the coding region homologies. Kent also introduced a different approach for detecting non-coding region homologies in his program BLAT.[19] BLAT uses consecutive seeds, but allows one or two mismatches to occur in any positions of the seed. For example, a BLAT hit might require at least ten matches in twelve consecutive positions. This scheme, naturally, allows more false negatives, but the resultant collection of hits is more enriched for true positives at a given level of false positives than for consecutive seeds where all positions are required to match.

In his random hashing strategy, Buhler[20] used his experience with identifying sequence motifs using random projection[21] to speed up detection of homology search. This idea had been previously identified by Indyk and Motwani.[22] Basically, this approach was to find all hits by random hashing over long sequence intervals. A simple probability calculation allows the computation of how many projections are required to guarantee a given probability that a homologous alignment will have a hit to at least one of these random projections. For good choices of projection weights, this will approach 100% sensitivity. Other than the fact that high weight random projections are not suitable and not designed for BLAST-type searches, this approach also ignores the possibility of optimizing the choice of those projections.

Of these first three approaches (specific spaced seeds, consecutive seeds allowing a fixed number of mismatches, and random spaced seeds), the first, typified by PatternHunter, allows for optimization. That is, one can use a seed specifically tuned for the types of alignments one expects to see, with the highest sensitivity at a particular false positive rate.

The program BLASTZ[23] has immediately adopted the PatternHunter's optimal spaced seed approach and allowed a mismatch to further improve sensitivity. BLAT, BLASTZ, and PatternHunter were extensively used in the large scale comparative genomics projects, such as the human-mouse genome comparison by the Mouse Genome Consortium.

The optimal seed models in the PatternHunter paper[3] were optimized for non-coding regions. Later, Brejová, Brown and Vinař[4] developed an algorithm for optimizing the seeds in more complicated models, specifically for coding regions, as did Buhler, Keich and Sun.[8]

In a later paper, Brejová, Brown and Vinař,[6] proposed a unified framework to represent all of the above mentioned seeding methods, which is described in Sec. 4.4.3.

Several other papers consider ideas in this new area of spaced seeds. Here, we cite some which we do not discuss further in this tutorial.[24–28]

4.2. *Optimized spaced seeds and why they are better*

Optimized spaced seeds can have substantially greater sensitivity than the consecutive seed models of BLAST. Here we give one example. The simplest model of an alignment is of a region of a fixed length where each position matches with some probability p, independent of all other positions. Figure 2 compares the optimal spaced seed model of weight 11 and length at most 18, 111010010100110111, with

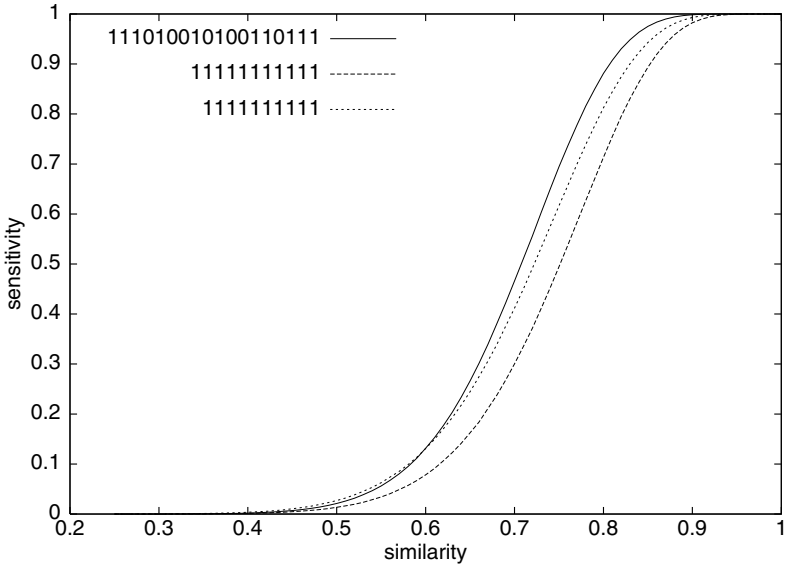

Fig. 2. 1-hit performance of weight 11 spaced model versus weight 11 and 10 consecutive models, coordinates in logarithmic scale.

BLAST's consecutive models of weight 11 and 10, for alignments of this type, of fixed length 64. For each similarity rate p shown on the x-axis, the fraction of regions with at least 1 hit is plotted on the y-axis as the sensitivity for that similarity rate. From the figure, one observes that the seemingly trivial change in the seed model significantly increases sensitivity. At 70% homology level, the spaced seed has over 50% higher probability (at 0.47) to have a hit in the region than BLAST weight 11 seed (at probability 0.3).

However, the added sensitivity does not come at the cost of more false positive hits or more hits inside true alignments[3]:

Lemma 1. *The expected number of hits of a weight W, length M model within a length L region of similarity $0 \leq p \leq 1$, is $(L - M + 1)p^W$.*

Proof. The expected number of hits is the sum, over the $(L - M + 1)$ possible positions of fitting the model within the region, of the probability of W specific matches, the latter being p^W. □

Lemma 1 reveals that spaced seeds have fewer expected hits, but have higher probability to hit a homologous region, as shown in Fig. 2. This is a bit counter-intuitive. The reason is that a consecutive seed often generates multiple hits in a region, because a hit at position i will increase the hit probability at position $i + 1$ to p (only one extra base match is required). However, the optimized spaced seeds are less likely to have multiple hits in a region (because the second hit requires more base matches). Therefore, given many homologous regions, although the total number of *hits* generated by a spaced seed is comparable to the number for a consecutive seed with the same weight, the the spaced seed hits can cover more *regions*.

To quantify this, assume that $p = 0.7$ and $L = 64$, as for the original PatternHunter model. Given that the BLASTN seed 11111111111 matches a region, the expected number of hits in that region is 3.56, while the expected number of hits to the spaced seed 101101100111001011, given that there is at least one, is just 2.05.

Thus, using an optimized spaced seed, a homology search program increases sensitivity and not running time. Inverting the above reasoning, we can use an optimal weight 12 spaced seed to achieve the BLAST weight 11 seed sensitivity, but generating four times fewer noise hits. This will speed up the search process by roughly a factor of four.

We have argued that for particular p, L, and seed models, spaced seeds are better than the consecutive seed. Can we prove a general mathematical statement about this? Mathematical analysis of the spaced seeds turns out to be challenging, very interesting, and related to the classical renewal theory of Markov processes. While the current research is too new to be presented systematically in this tutorial, we sample one theorem, from Keich *et al.*,[7] and refer the reader to other papers.[5,7,8,29] The next theorem shows that in an infinite region, as the seeds scan the region, a spaced seed has a higher probability to hit first than the consecutive seed.

Theorem 1. *Let I be a uniform homologous region of homology level p. For any sequence*

$$\leq i_0 < i_1 \cdots < i_{n-1} = |I| - |s|,$$

let A_j be the event a spaced seed s hits I at i_j, B_j be event the consecutive seed hits I at j. Then

$$\mathrm{P}\left(\bigcup_{j<n} A_j\right) \geq \mathrm{P}\left(\bigcup_{j<n} B_j\right); \tag{1}$$

when $i_j = j$, the strict inequality holds in (1).

Proof. We prove the theorem by induction on n. For $n = 1$, $\mathrm{P}(A_0) = p^W = \mathrm{P}(B_0)$. Assume that the theorem holds for $n = N$, we prove that it holds for $N + 1$. Let E_k denote the event that, when putting the spaced seed s at position $I[0]$, the first k (0-th to $k-1$-st) 1's in the seed are all matched and the next 1 mismatches. Let E'_k be the similar event for the consecutive seed. Clearly, E_ks are a partition of the sample space for $k = 0, .., W$, and $\mathrm{P}(E_k) = p^k(1-p)$. The same is true for E'_k. Thus it is sufficient to show, for $k = W$:

$$\mathrm{P}\left(\bigcup_{j=0,\ldots,N} A_j | E_k\right) \geq \mathrm{P}\left(\bigcup_{j=0,\ldots,N} B_j | E'_k\right). \tag{2}$$

When $k = W$, both sides of (2) equal to 1. For $k < W$ since $\left(\bigcup_{j\leq k} B_j\right) \cap E'_k = \emptyset$ and $\{B_{k+1}, B_{k+2}, \ldots, B_N\}$ are mutually independent of E'_k, we have

$$\mathrm{P}\left(\bigcup_{j=0,\ldots,N} B_j | E'_k\right) = \mathrm{P}\left(\bigcup_{j=k+1,\ldots,N} B_j\right). \tag{3}$$

Now consider the first term in Inequality 2. For each $k \in \{0, \ldots, W-1\}$, at most $k+1$ of the events A_j satisfy $A_j \cap E_k = \emptyset$. This is because $A_j \cap E_k = \emptyset$ iff when aligned at position i_j the seed s has a 1 bit at the overlapping kth bit when the seed was at 0. There are at most $k+1$ choices. Thus, there exist indices $0 < m_{k+1} < m_{k+2} \ldots < m_N \leq N$ such that $A_{m_j} \cap E_k \neq \emptyset$. Since E_k means all previous bits matched, it is clear that E_k is non-negatively correlated with $\bigcup_{j=k+1,\ldots,N} A_{m_j}$, thus

$$\mathrm{P}\left(\bigcup_{j=0,\ldots,N} A_j | E_k\right) \geq \mathrm{P}\left(\bigcup_{j=k+1,\ldots,N} A_{m_j} | E_k\right) \geq \mathrm{P}\left(\bigcup_{j=k+1,\ldots,N} A_{m_j}\right). \tag{4}$$

The inductive hypothesis yields

$$\mathrm{P}\left(\bigcup_{j=k+1,\ldots,N} A_{m_j}\right) \geq \mathrm{P}\left(\bigcup_{j=k+1,\ldots,N} B_j\right).$$

Combined with (3) and (4), this proves (2). Thus this proves the $\geq$ part of the theorem.

We now prove, by induction on n, when $i_j = j$,

$$\mathrm{P}\left(\bigcup_{j=0,\ldots,n-1} A_j\right) > \mathrm{P}\left(\bigcup_{j=0,\ldots,n-1} B_j\right). \tag{5}$$

For $n = 2$, we have

$$\mathrm{P}\left(\bigcup_{j=0,1} A_j\right) = 2p^w - p^{2w-\mathrm{shift1}} > 2p^w - p^{w+1} = \mathrm{P}\left(\bigcup_{j=0,1} B_j\right)$$

where shift1 is the number of overlapped bits of the spaced seed s with itself when shifted to the right by 1 bit. For inductive step, note that the proof of (1) shows that for all $k = 0, 1, \ldots, W$,

$$\mathrm{P}\left(\bigcup_{j=0,\ldots,n-1} A_j | E_k\right) \geq \mathrm{P}\left(\bigcup_{j=0,\ldots,n-1} B_j | E'_k\right).$$

Thus to prove (5) we only need to prove that

$$\mathrm{P}\left(\bigcup_{j=0,\ldots,n-1} A_j | E_0\right) > \mathrm{P}\left(\bigcup_{j=0,\ldots,n-1} B_j | E'_0\right).$$

The above follows from the inductive hypothesis as follows:

$$\begin{aligned}\mathrm{P}\left(\bigcup_{j=0,\ldots,n-1} A_j | E_0\right) &= \mathrm{P}\left(\bigcup_{j=1,\ldots,n-1} A_j\right) > \mathrm{P}\left(\bigcup_{j=1,\ldots,n-1} B_j\right) \\ &= \mathrm{P}\left(\bigcup_{j=0,\ldots,n-1} B_j | E'_0\right).\end{aligned}$$

□

Corollary 1. *Assume I is infinite. Let p_s and p_c be the first positions a spaced seed and the consecutive seed hit I, respectively. Then* $\mathrm{E}[p_s] < \mathrm{E}[p_c]$.

Proof.

$$\begin{aligned}\mathrm{E}[p_s] &= \sum_{k=0,\ldots,\infty} k\mathrm{P}(p_s = k) \\ &= \sum_{k=0,\ldots,\infty} k[\mathrm{P}(p_s > k-1) - \mathrm{P}(p_s > k)] \\ &= \sum_{k=0,\ldots,\infty} \mathrm{P}(p_s > k) \\ &= \sum_{k=0,\ldots,\infty} \left(1 - \mathrm{P}\left(\bigcup_{j=0,\ldots,k} A_j\right)\right) \\ &< \sum_{k=0,\ldots,\infty} \left(1 - \mathrm{P}\left(\bigcup_{j=0,\ldots,k} B_j\right)\right) \\ &= \mathrm{E}[p_c].\end{aligned}$$

□

4.3. *Computing the optimal spaced seeds*

The probability of a seed generating a hit in a fixed length region of a given level similarity can be computed by dynamic programming under various assumptions.[3–8,29] To choose an optimal seed, we compute the hit probability for all seeds, and pick the one with the highest probability.

Suppose we are given a seed s, of length M and weight W, and a homology region R, of length L and homology level p, with all positions independent of each other. In this model, we can compute the probability of s having a hit in R. We represent R by a random string of zeros and ones, where each position has probability p of being a one.

We will say that seed s has a seed match to R at location i if the L-length substring of R starting at position i has a one in each position with a one in the seed s. Let A_i be the event that seed s has a seed match at location i in R, for all $0 \leq i \leq L - M$. Our goal is to find the probability that s hits R: $\Pr\left[\bigcup_{i=0}^{L-M} A_i\right]$.

For any $M \leq i \leq L$ and any binary string b such that $|b| = M$, we use $f(i, b)$ to denote the probability that s hits the length i prefix of R that ends with b:

$$f(i,b) = \Pr\left[\bigcup_{j=0}^{i-M} A_j \mid R[i-l,\ldots,i-1] = b\right].$$

In this framework, if s matches b,

$$f(i,b) = 1.$$

Otherwise, we have the recursive relationship:

$$f(i,b) = (1-p)f(i-1,0b') + pf(i-1,1b'), \tag{6}$$

where b' is b, after deleting its last bit.

Once we have used this dynamic programming algorithm to compute $f(L-M, b)$ for all strings b, we can compute the probability of s hitting the region; it is:

$$\sum_{|b|=M} \Pr[R[i-1,\ldots,i-1] = b] \cdot f(L-M,b).$$

This is the simplest algorithm to compute the hit probability of a seed. This algorithm was used in the original PatternHunter paper[3] and published in an improved form in Keich *et al.*[7] Other algorithms generalize this algorithm to more sophisticated probabilistic assumptions of the region R and reducing time complexity.[4,5,8,29]

4.4. *Computing more realistic spaced seeds*

The simple model of homologous regions described above is not very precise representative of real alignments. Homologous regions are not all of length 64, and vary internally in how conserved they are. (Of course, they also include gaps, but we are still not going to consider this in producing seeds.)

For example, more than 50% of significant alignments in the human and mouse genomes are between exonic regions, and these regions have substantially more conservation in the first two positions of a codon than in the third, which has traditionally been called the "wobble" position.[18] As such, a seed that takes advantage of this three-periodicity by ignoring the third position in codons will be much more likely to hit than a seed that does not. There is also substantial dependence within a codon: if the second position is not matched, it is quite likely that neither are the first or third.

Similarly, real alignments vary substantially internally in their alignment as well, and this is particularly true for coding alignments. Such alignments tend to have core regions with high fidelity, surrounded by less well-conserved regions.

Models that do not account for these variabilities can substantially underestimate the hit probability of a seed.

4.4.1. *Optimal seeds for coding regions*

Two recent papers try to address this need to optimize better models in different ways. Brejová, Brown and Vinař[4] use a Hidden Markov Model (HMM) to represent the conservation pattern in a sequence. Their model accounts for both internal dependencies within codons, and also for multiple levels of conservation throughout a protein. They update the simple dynamic programming model above to this new framework, and show that one can still relatively efficiently compute the probability that a given seed matches a homologous region.

Meanwhile, Buhler, Keich and Sun[8] represent the sequences by Markov chains, and present a different algorithm, based on finite automata to compute the probability of a seed hit in that model.

The difference in using seeds tuned to find hits in homologous coding regions versus seeds for general models is quite large. In particular, the optimal seed (111001001001010111) of weight 10 and length at most 18 for the noncoding regions is ranked 10,350 among the 24,310 possible seeds in its theoretical sensitivity in the HMM trained by Brejová and co-authors, and ranked 11,258 among these seeds in actual sensitivity on a test set of coding region alignments, matching only 58.5% of them. By contrast, the three optimal coding region seeds (which are also optimal for the theoretical hidden Markov model) match between 84.3% and 85.5% of alignments. These seeds, 11011011000011011, 11011000011011011, and 11000011011011011, all ignore the third positions of codons, and also skip entire codons. As such, they model the conservation pattern of real coding sequences substantially better than the non-periodic seeds optimized for noncoding regions.

Much of the advantage does, of course, come from the modeling. A previous program, WABA,[18] uses three-periodic seeds of the form 110110110 ... with no formal justification; in practice, this seed has sensitivity close to the optimum,

81.4%. Still, the optimal seeds give a good improvement over this seed, and also allow the optimization of multiple seeds for still greater sensitivity.

For the homology search in coding regions, an alternative approach would be the use of a *translated homology search* program, e.g., TBLASTX. Such a program first translates the DNA sequences to protein sequences, from which the homologies are then found. The translated homology search is supposed to be more sensitive than a DNA-based program for coding regions, however, is substantially slower.

Kisman and coauthors[30] recently extended the spaced seed idea to translated homology search and developed tPatternHunter, which is both faster and more sensitive than TBLASTX.

4.4.2. *Optimal seeds for variable-length regions*

As to the length of regions, it is quite simple for all of these algorithms to incorporate distributions on the length of homologous regions into the model. For a given generative model, we simply compute the probability $\sigma(\ell)$ of a hit in a region of length ℓ, and the probability distribution $\pi(\ell)$ of lengths of the region; the probability of a hit in a random region is then just $\sum_\ell \sigma(\ell)\pi(\ell)$.

4.4.3. *Vector seeds: unifying the different models of seeds*

In another paper,[6] Brejová, Brown and Vinař combined and generalized the ideas of both BLAT and PatternHunter to a broader representation language for the seeding of alignments.

Their approach is to view the position-by-position scores of an ungapped alignment as a sequence of numbers, which they call an alignment sequence. For the simplest model of nucleotide alignment, the alignment sequence is just a sequence of $+1$ and -1 values, corresponding to positions that match or do not match. However, for protein sequences, this sequence consists of the values from the scoring matrix, M, that correspond to the residues aligned; in the case of the BLOSUM62 matrix, these range from -5 to $+12$. For a given alignment, let this alignment sequence be $A = a_1, \ldots, a_n$.

Then, given an alignment sequence, these authors model a seed as a vector v and a threshold T, and identify a hit at position i if $v \cdot (a_i, \ldots, a_{i+|v|-1}) \geq T$. This framework encodes the seeding strategies described before. For consecutive seeds in nucleotide alignments, the vector is of all ones, and the threshold T is the same as the length of the vector. For spaced seeds, the seed vector is the same as before, and the threshold is the weight of the seed. For BLAT-style consecutive seeds with a fixed number of mismatches, the vector is again of all ones, but the threshold is equal to the length minus two times the number of allowed mismatches (since the score of a mismatch is two less than for a match). This framework also encodes the seeding strategy for BLASTP, which requires three consecutive positions with total score greater than 13 (corresponding to the vector seed $((1, 1, 1), 13)$.

Other strategies can also be expressed in this formalism. When using the simple nucleotide scoring strategy, these can all also be expressed as a collection of many spaced seeds; for example, the vector seed $((1, 1, 0, 1), 2)$ corresponds to the three spaced seeds 1100, 1001 and 0101. However, in the case where positional scores may be from a richer system, vector seeds can encapsulate more complicated relationships.

In their original paper, Brejová and her co-authors do not show much value to using vector seeds where the vector v includes values that are not zero or one; this may be because this would over-emphasize specific positions of the alignment. However, some other seeding approaches that filter initial hits for being in a region of high overall sensitivity can be expressed in this more complicated vector seed framework. For example, a requirement that the first, third, and sixth positions of the seed must match, as well as at least two of the second, fourth and fifth positions, can be encoded with the vector seed $((10, 1, 10, 1, 1, 10), 32)$.

4.4.4. *Variable length seeds: handling bias in character distribution*

In the previous sections, a homolog is modeled by a 0-1 string, where 0 means mismatch and 1 means match. The optimization of the seeds are all based on the distributions of 1 (match) and 0 (mismatch) in the string. As a result, the actual DNA or protein sequence information in each homolog is disgarded and only the matches and mismatches are counted. For example, if one of the following two alignments is hit by a seed, the other one will be hit by the same seed also.

```
AAAAAAAAAA          GGGGGGGGGG
|||| |||||          |||| |||||
AAAACAAAAA          GGGGCGGGGG
```

However, the distribution of the characters (nucleotides for DNA and amino acids for protein sequences) in genomic seqences is often biased. For example, the mouse genome has a 42% (G+C) content and 58% of (A+T). Therefore, even the database is fixed, the same seed will generate different number of false positive hits for different query words.

Naturally, Csürös[31] suggested using shorter seeds for the query words that are rarer. As a result, higher sensitivity can be achieved with a little extra false positive hits. Csürös[31] implemented this variable length seed idea using a pruned seed tree. In the following the pruned seed tree is described using consecutive seeds first. The case of spaced seeds is discussed at the end of this section.

Using a database sequence, a seed tree is first obtained by placing the k-mers of the database sequence to a trie data structure. So, the trie has k levels. Each leaf at level k corresponds to a k-mer. Each internal node at level t corresponds to a t-mer, which is a common prefix of the $(t + 1)$-mers of the child nodes.

In a real genetic sequence, different t-mers occur with different frequencies. For any t-mer v, let $Occ(v)$ be the set of occurrences of t-mer v in the sequence. Then

$Occ(v) = \bigcup_{u \text{ being a child of } v} Occ(u)$ if we ignore the small error caused at the end of the database sequence.

Then the seed tree is pruned. Each step of the pruning removes a group of leaves that are siblings. As a result, the parent node of the siblings becomes a new leaf which occurs more frequently than the siblings. After the pruning is done, for each k-mer u of the query sequence, hits will be generated by finding u's longest prefix v as a leaf in the pruned tree.

It is easy to see that each pruning may decrease the length of the prefix for the hit generation. As a result, the sensitivity of the homology search is increased. On the other hand, the searching speed will be decreased because more hits are generated. Csürös[31] used a greedy heuristic to prune the tree for gaining sensitivity with losing the least speed. That is, the hit number increment caused by the next pruning step should be the least possible.

For a query sequence with length M, if we ignore the errors at the end of the sequence, the expected number of hits a node $v \in \Sigma^t$ generates is

$$hits(v) = M \times |Occ(v)| \times \prod_{j=1}^{t} q(v[j]),$$

where $q(a)$ is the expected frequency of character a in the query sequence. The hit number increment of each possible pruning step can then be easily calculated as $hits^+(v) = hits(v) - \sum_{u \text{ being a child of } v} hits(u)$. Csürös[31] also proved that $hits^+(\cdot)$ is in general larger at a parent node than at the children (unless the query sequence consists mostly the same character). Therefore, by putting $hits^+(v)$ in a sorted data structure, the pruning can be carried out efficiently in a sequential way. Whenever a node is considered for pruning, all of its descendants are pruned already. The pruning stops when the total hit increment reaches a threshold.

The pruned tree approach works for spaced seeds as well because the k-mers mentioned above can be k characters at the "1"-positions defined by a weight-k spaced seed. Also, it is advantageous to use a permutation π to permute the k-mer to maximize the sensitivity. That is, each k-mer $(s_1, s_2, \ldots, s_k)$ is changed to $(s_{\pi(1)}, s_{\pi(2)}, \ldots, s_{\pi(k)})$ during the building of the pruned tree and hit generation. Csürös[31] also suggested a simple algorithm to select the optimal permutation for any given spaced seed.

4.5. *Approaching the Smith–Waterman sensitivity using multiple seed models*

Another idea that comes directly from the idea of optimal spaced seeds is the one of using multiple seed models, which together optimize the sensitivity. In such an approach, a set of several seed models are selected first. Then all the hits generated by all the seed models are examined to produce local alignments. This obviously increases the sensitivity because more hits than using one seed model are examined.

But now, the several seed models need to be optimized together to maximize the sensitivity.

This idea appeared in the PatternHunter paper[3] and is further explored in several recent papers.[5,6,8] Brejová and her coauthors[6] used a heuristic method to design a good pair of seeds; Buhler and co-authors[8] used hill-climbing to locally improve good sets of seeds and a pair of seeds were designed by their method; Li and co-authors[5] extended the dynamic programming algorithm in Sec. 4.3 to compute a suboptimal set of seeds greedily.

Li and his co-authors[5] showed that, practically, doubling the number of seeds would achieve better sensitivity than reducing the weight of the single seed by one. However, for DNA homology search, the former only approximately doubles the number of hits, whereas the latter will increase the number of hits by a factor of four (the size of DNA alphabet). Thus, multiple seeds are a better choice.

It is noteworthy that the multiple seed approach is only possible when spaced seeds are used — there is only one BLAST-type of consecutive seed with a given weight. The newest version of PatternHunter implements the multiple seed scheme,[5] having greedily chosen a set of 16 seeds of weight 11 and length at most 21 in 12 CPU days on a Pentium IV 3 GHz PC. When the random region has length 64 and similarity 70%, the first four seeds are: 111010010100110111, 111100110010100001011, 110100001100010101111, 1110111010001111. Multiple seeds for coding regions are also computed and implemented. The experimental results that will be shown in Sec. 5 demonstrate that using carefully selected multiple seeds can approach Smith–Waterman sensitivity at BLAST's speed.

4.5.1. *Multiple vector seeds for protein alignment*

As noted above, vector seeds can encapsulate similar ideas to those of spaced seeds for more complex scoring schemes. Indeed, their original development was aimed at the alignment of proteins. However, the original paper by Brejová and her co-authors[6] did not show much success in this direction; while vector seeds had slightly better sensitivity than consecutive seeds at the same false positive hit rate, the advantage was slight.

An alternative approach, however, was inspired by the multiple seed approach described in this subsection for nucleotide seeds. Based on a reference set of training alignments, Brown[32] gave an optimization framework for computing a good set of vector seeds. His framework is also through linear programming, as for the paper of Xu and co-authors[33] described below. However, the difference is that the objective is to minimize the rate of false positive hits, while still achieving the very high sensitivity that BLASTP has, while the linear program of Xu and co-authors seeks highest possible sensitivity for a given false positive rate.

Through using multiple vector seeds, with support 3, 4 and 5, Brown finds a seeding strategy that will have approximately the same sensitivity as BLASTP,

while keeping five times fewer false positives. He also points out that since filtering hits in their local region takes a constant amount of time, this step may be used in a two-pass strategy to find good hits that should actually be extended to alignments.

4.6. *The complexity of computing optimal spaced seeds*

Many authors [3,4,6–8,29] have proposed heuristic or exponential time algorithms for the general seed selection problem: find one or several 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. Unfortunately, these are all NP-hard problems.[5] Thus the greedy algorithm and the exponential time dynamic programming are the best we can do. While this tutorial will not provide the proofs, we wish to list some of the recent results for these problems. Letting $f(n)$ be the maximum number of 0's in each seed, where n is the seed length, the following are true[5]:

(1) If $f(n) = O(\log n)$, then there is a dynamic programming algorithm computes the hit probability of k seeds in polynomial time; otherwise the problem is NP-hard.
(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 computing their probabilities; otherwise, even selecting one optimal seed is NP-hard.
(3) If $f(n) = O(1)$, then the greedy algorithm of picking k seeds by enumeration, then adding the seed that most improves the first seed, and so on, approximates the optimal solution within ratio $1 - \frac{1}{\mathbf{e}}$ in polynomial time, due to the bound for the greedy algorithm for the maximum coverage problem[34]; otherwise the problem cannot be approximated within ratio $1 - \frac{1}{\mathbf{e}} + \epsilon$ for any $\epsilon > 0$, unless $NP = P$.

4.7. *Computing a good seed set*

Despite of the complexity of computing optimal spaced seeds, approximation algorithms have been developed. Li and co-authors[5] suggested to use a greedy heuristic to select the multiple seeds one by one, each most improves the hit probability of the existing seeds. The hit probability of the hit set found by this simple greedy algorithm is at least $1 - \frac{1}{\mathbf{e}}$ times the hit probability of the optimal seed set. When the number of '0' positions in the seeds is bounded by a constant, there will be only polynomial number of seeds and the above algorithm runs in polynomial time.

Xu and his co-authors[33] studied another approximation algorithm which formulates the multiple seed selection problem as an integer linear program, solves the corresponding linear programming relaxation, and rounds the results to be integeral.

In Xu and co-authors' paper,[33] both the possible seeds and the homology regions are regarded as input of the problem. Let $S_1, S_2, \ldots, S_m$ be the seeds and $R_1, R_2, \ldots, R_N$ be the regions. The objective of the problem is to select a subset of seeds that hit the most regions. The integer program is straightforward: binary decision variable x_i indicates whether S_i is selected, and y_j indicates whether R_j is hit. Then the problem is formulated as

$$\max \frac{1}{N} \sum_{j=1}^{N} y_j$$

subject to

$$y_j \leq \sum_{S_i \text{ hits } R_j} x_i \quad \text{for } j = 1, \ldots, N. \tag{7}$$

$$\sum_{i=1}^{m} x_i = k \tag{8}$$

$$x_i \in \{0, 1\}$$

$$y_j \in \{0, 1\}.$$

Constraint (7) says that y_j is 1 if at least one of the seed that hits R_j is selected. Constraint (9) limits the total number of seeds being selected.

By removing the integrality requirement, one can solve the resultant linear program efficiently to get the optimal relaxed solution (x^*, y^*). The maximum value of the objective function for this LP relaxation provides an upper bound on the performance of k seeds.

Next, random variables X, which have value i with probability $\frac{x_i^*}{k}$ are employed to do the randomized rounding. We make k independent observations of X and the observed values are the indices of the selected seeds. If a seed is selected more than once in this procedure, only one instance of the seed is kept and the resulted seed set will have size smaller than k. Xu and his co-authors prove[33] that, with high probability, this rounding procedure gives a seed set that hits at least $\left(1 - \frac{1}{\mathbf{e}}\right) \times OPT$ regions, where OPT is the maximum number of regions an optimal set of k seeds can hit.

Therefore, this linear programming relaxation and random rounding algorithm give an polynomial time approximation algorithm with the same approximation ratio as the greedy algorithm. However, if the lower and upper bounds on the number of regions hit by every possible seed are known, in some cases the linear programming-based approximation has a better provable approximation ratio.[33] Nevertheless, the two approximation algorithms have similar performance in practice.

When the homology regions are not given individualy as inputs, they are usually given by a probabilistic model. Then we can randomly sample enough number of homology regions and apply the above mentioned linear programming and rounding

algorithm. Xu and co-authors use a Chernoff-style bound to prove that the seed set computed based on the sampled homology regions performs almost equally well on all homology regions under the probabilistic model.

5. Experiments

Here, we present some experimental comparisons between heuristic search techniques and full Smith–Waterman dynamic programming. As expected, Smith–Waterman dynamic programming is too slow for practical use when the database is large. What is striking is that a good choice of multiple optimal spaced seeds can allow near-total success in detecting alignments, with vastly better runtime.

The results that we show in this section were originally reported by Li and co-authors.[5] In the paper, several software packages, SSearch,[35] BLAST, and PatternHunter, were used to find homologies between 29,715 mouse EST sequences and 4407 human EST sequences. Those sequences are the new or newly revised mouse and human EST sequences in NCBI's GenBank database within a month before April 14, 2003. After downloading the EST sequences, a simple "repeat masking" was conducted to replace all the sequences of ten or more identical letters to letter `N`s, because they are low complexity regions and their existence will only generate so many trivial sequence matches that overwhelm the real homologies.

SSearch is a subprogram in the FASTA package and implments the Smith–Waterman algorithm. Therefore, SSearch's sensitivity is regarded to be 100% in the comparison. The performance of BLAST version 2.2.6, and PatternHunter version 2.0 were compared against SSearch. Each program uses a score scheme equivalent to: match: 1, mismatch: -1, gap open penalty: -5, gap extension penalty: -1. All pairs of ESTs with a local alignment of score at least 16 found by SSearch were recorded, and if a pair of ESTs has more than two local alignments, only the one with the highest score was considered. All of these alignments were kept as being the correct set of homologies, noting that, of course, some of these alignments may between sequences that are not evolutionally related.

As expected, SSearch took approximately 20 CPU days, while BLASTN took 575 CPU seconds, both on a 3 GHz Pentium IV. SSearch found 3,346,700 pairs of EST sequences that have local alignment score at least 16, with maximum local alignment score 694.

It is difficult to compare SSearch's sensitivity with BLASTN and PatternHunter, as BLASTN and PatternHunter are heuristic algorithms, and need not compute optimal alignments. Thus Li and his co-authors decided, a bit arbitrarily, that if SSearch finds a local alignment with score x for a pair of ESTs, and BLAST (or PatternHunter) finds an alignment with score $\geq \frac{x}{2}$ for the same pair of ESTs, then BLAST (or PatternHunter) "detects" the homology. The successful detection rate is then regarded the sensitivity of BLAST (or PatternHunter) at score x. PatternHunter was run several times with different number of spaced seeds. Two

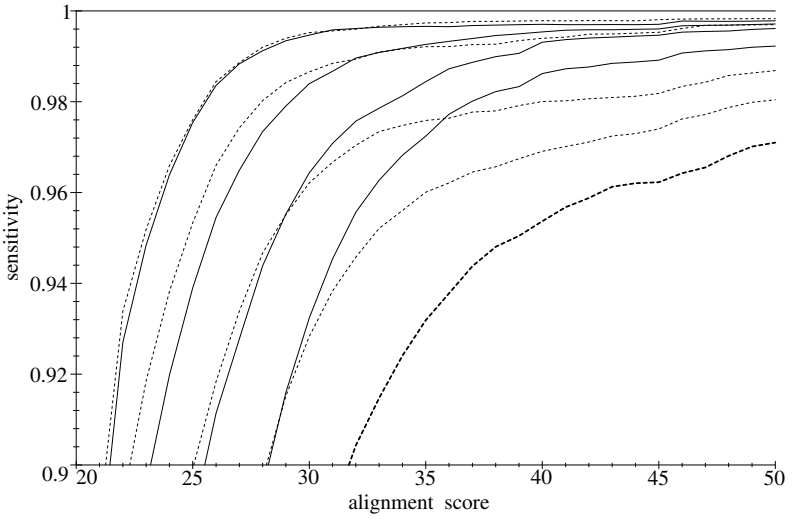

Fig. 3. The thick dashed curve is the sensitivity of BLASTN, seed weight 11. From low to high, the solid curves are the sensitivity of PatternHunter using $1, 2, 4$, and 8 weight 11 coding region seeds, respectively. From low to high, the dashed curves are the sensitivity of PatternHunter using $1, 2,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

sets of seeds, coding-region seeds and general-purpose seeds, are used, respectively. The results are in Fig. 3.

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

SSearch	Blastn	PatternHunter				
20 days	575 s	Seeds	1	2	4	8
		General	242 s	381 s	647 s	1027 s
		Coding	214 s	357 s	575 s	996 s

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

The above table also demonstrates 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.

6. Conclusion

Our brief tour through the world of homology search methods is coming to an end. Early landmarks included the classic global and local alignment algorithms that

use full dynamic programming, while later highlights have been heuristic search algorithms, including increasingly sophisticated ones based on optimizing seeds for a particular type of alignment. At the end of it all, we have an algorithm that is almost as sensitive as Smith–Waterman, but requiring 3 orders of magnitude less time. We cannot believe, however, that the tour is entirely over, and encourage our readers to enjoy some time sightseeing on their own.

Acknowledgments

This work was supported in part by the Natural Science and Engineering Research Council of Canada (NSERC), the Ontario PREA award, Canada Council Research Chair program, the Killam Fellowship, and the Human Frontier Science Program. We would like to thank those who provided preprints or unpublished material that was used in this article.

References

1. Lipman D, Pearson W, Rapid and sensitive protein similarity searches, *Science* **227**: 1985.
2. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ, Basic local alignment search tool, *J Mol Biol* **215**(3):403–410, 1990.
3. Ma B, Tromp J, Li M, PatternHunter: Faster and more sensitive homology search, *Bioinformatics* **18**(3):440–445, March 2002.
4. Brejová B, Brown D, Vinař T, Optimal spaced seeds for homologous coding regions, *J Bioinf Comput Biol* **1**(4):595–610, 2004. Early version appeared in CPM 2003.
5. Li M, Ma B, Kisman D, Tromp J, PatternHunter II: Highly sensitive and fast homology search, to appear in *J Bioinf and Comp Biol* 2004.
6. Brejová B, Brown D, Vinař T, Vector seeds: An extension to spaced seeds allows substantial improvements in sensitivity and specificity, in *Proceedings of the 3rd Annual Workshop on Algorithms in Bioinformatics (WABI)*, pp. 39–54, 2003.
7. Keich U, Li M, Ma B, Tromp J, On spaced seeds of similarity search, *Discrete Appl Math* **138**:253–263, 2004.
8. Buhler J, Keich U, Sun Y, Designing seeds for similarity search in genomic DNA, in *Proceedings of the 7th Annual International Conference on Computational Biology (RECOMB)*, pp. 67–75, 2003.
9. Karlin S, Altschul SF, Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes, *Proc Natl Acad Sci USA* **87**:2264–2268, 1990.
10. Karlin S, Altschul SF, Applications and statistics for multiple high-scoring segments in molecular sequences, *Proc Natl Acad Sci USA* **90**:5873–5877, 1993.
11. Mott R, Maximum likelihood estimation of the statistical distribution of smith-waterman local sequence similarity scores, *Bull Math Biol* **54**:59–75, 1992.
12. Needleman SB, Wunsch CD, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J Mol Biol* **48**:443–453, 1970.
13. Gotoh O, An improved algorithm for matching biological sequences, *J Mol Biol* **162**:705–708, 1982.
14. Hirschberg DS, Algorithms for the longest common subsequence problem, *J ACM* **24**:664–675, 1977.

15. Smith T, Waterman M, Identification of common molecular subsequences, *J Mol Biol* **147**:195–197, 1981.
16. Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W, Lipman DJ, Gapped BLAST and PSI-BLAST: A new generation of protein database search programs, *Nucl Acids Res* **25**(17):3389–3392, 1997.
17. National Center for Biotechnology Information Growth of GenBank, 2002, http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html.
18. Kent WJ, Zahler AM, Conservation, regulation, synteny, and introns in a large-scale *C. briggsae-C. elegans* genomic alignment, *Genome Res* **10**(8):1115–1125, 2000.
19. Kent WJ BLAT–the BLAST-like alignment tool, *Genome Res.* **12**(4):656–664, 2002.
20. Buhler J, Efficient large-scale sequence comparison by locality-sensitive hashing, *Bioinformatics* **17**:419–428, 2001.
21. Buhler J, Tompa M, Finding motifs using random projections, in *Proceedings of the 5th Annual International Conference on Computational Biology (RECOMB)*, pp. 69–76, 2001.
22. Indyk P, Motwani R, Approximate nearest neighbors: towards removing the curse of dimensionality, in *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 604–613, 1998.
23. Schwartz S, Kent W, Smit A, Zhang Z, Baertsch R, Hardison R, Haussler D, Miller W, Human-mouse alignments with BLASTZ, *Genome Res* **13**:103–107, 2003.
24. Sun Y, Buhler J, Designing multiple simultaneous seeds for dna similarity search, in *Proceedings of the 8th Annual International Conference on Computational Biology (RECOMB)*, pp. 76–84, 2004.
25. Kucherov G, Noe L, Ponty Y, Estimating seed sensitivity on homogeneous alignments, in *Proceedings of the 4th IEEE Symposium on Bioinformatics and Bioengineering (BIBE)*, pp. 387–394, 2004.
26. Kucherov G, Noe L, Roytberg M, Multi-seed lossless filtration, in *Proceedings of the 15th Symposium on Combinatorial Pattern Matching (CPM), LNCS 3109*, pp. 297–310, 2004.
27. Noe L, Kucherov G, Improved hit criteria for DNA local alignment, in *Proceedings of Journés Ouvertes de Biologie, Informatique et Mathématiques (JOBIM 2004)*, 2004.
28. Choi KP, Zeng F, Zhang L, Good spaced seeds for homology search, *Bioinformatics* **20**:1053–1059, 2004.
29. Choi K, Zhang L, Sensitive analysis and efficient method for identifying optimal spaced seeds, *J Comp and Sys Sci* **68**:22–40, 2004.
30. Kisman D, Li M, Ma B, Wang L, TPatternHunter: Gapped, fast and sensitive translated homology search, to apppear in *Bioinformatics*, 2004.
31. Csürös M, Performing local similarity searches with variable length seeds, in *Proceedings of 15th Annual Symposium on Combinatorial Pattern Matching (CPM), LNCS 3109*, pp. 373–387, 2004.
32. Brown D, Multiple vector seeds for protein alignment, in *Proceedings of 4th Workshop on Algorithms in Bioinformatics (WABI)*, 2004.
33. Xu J, Brown D, Li, M, Ma B, Optimizing multiple spaced seeds for homology search, in *Proceedings of the 15th Symposium on Combinatorial Pattern Matching (CPM), LNCS 3109*, pp. 47–58, 2004.
34. Hochbaum DS, Approximating covering and packing problems, in Hochbaum DS (ed.) *Approximation Alogithms for NP-hard Problems*, Chapter 3, pp. 94–143, PWS, 1997.
35. Pearson W, Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms, *Genomics* **11**:635–650, 1991.

Daniel Brown received his undergraduate degree in mathematics with Computer Science from the Massachusetts Institute of Technology in 1995, and his Ph.D. in computer science from Cornell University in 2000. He then spent a year as a Research Scientist at the Whitehead Institute/MIT Center for Genome Research, in Cambridge, Massachusetts, working on the Human and Mouse Genome Projects. Since 2001, he has been an Assistant Professor in the School of Computer Science at the University of Waterloo.

Ming Li is a CRC Chair Professor in Bioinformatics, of Computer Science at the University of Waterloo. He is a recipient of Canada's E.W.R. Steacie Followship Award in 1996, and the 2001 Killam Fellowship. Together with Paul Vitanyi they pioneered applications of Kolmogorov complexity and co-authored the book "An Introduction to Kolmogorov Complexity and Its Applications" (Springer-Verlag, 1993, 2nd Edition, 1997). He is a co-managing editor of *Journal of Bioinformatics and Computational Biology*. He currently also serves on the editorial boards of *Journal of Computer and System Sciences*, *Information and Computation*, *SIAM Journal on Computing*, *Journal of Combinatorial Optimization*, *Journal of Software*, and *Journal of Computer Science and Technology*.