BIOINFORMATICS

Vol. 18 no. 3 2002
Pages 440–445

PatternHunter: faster and more sensitive homology search

Bin Ma[1], John Tromp[2] and Ming Li[3]

[1]Computer Science Department, University of Western Ontario, London N6A 5B8, Canada, [2]Bioinformatics Solutions Inc., 145 Columbia Street West, Waterloo, Ont. N2L 3L2, Canada and [3]Computer Science Department, University of Waterloo, Waterloo, Ont. N2L 3G1, Canada and Bioinformatics Lab, Computer Science Department, University of California, Santa Barbara, CA 93106, USA

Received on August 24, 2001; revised on October 10, 2001; accepted on October 15, 2001

ABSTRACT

Motivation: Genomics and proteomics studies routinely depend on homology searches based on the strategy of finding short seed matches which are then extended. The exploding genomic data growth presents a dilemma for DNA homology search techniques: increasing seed size decreases sensitivity whereas decreasing seed size slows down computation.

Results: We present a new homology search algorithm 'PatternHunter' that uses a novel seed model for increased sensitivity and new hit-processing techniques for significantly increased speed. At Blast levels of sensitivity, PatternHunter is able to find homologies between sequences as large as human chromosomes, in mere hours on a desktop.

Availability: PatternHunter is available at http://www.bioinformaticssolutions.com, as a commercial package. It runs on all platforms that support Java. PatternHunter technology is being patented; commercial use requires a license from BSI, while non-commercial use will be free.

Contact: mli@cs.ucsb.edu

INTRODUCTION

We are interested in faster and more sensitive methods for finding all approximate repeats or homologies in one DNA sequence or between two DNA sequences, as performed by the popular Blastn (Altschul *et al.*, 1990) program. One particular application of this task is in comparative genomics where large genomes or chromosomes such as the human one (International Human Genome Sequencing Consortium, 2001; Venter *et al.*, 2001) need to be compared.

Many programs have been developed for the task. These include FASTA (Lipman and Pearson, 1985), SIM (Huang and Miller, 1991), the Blast family (Altschul *et al.*, 1990; Gish, 2001; Altschul *et al.*, 1997; Zhang *et al.*, 2000; Tatusova and Madden, 1999), SENSEI (States, 2000), MUMmer (Delcher *et al.*, 1999), QUASAR (Burkhardt *et al.*, 1999), and REPuter (Kurtz and Schleiermacher, 1999).

Smith–Waterman alignment which compares all bases against all bases is clearly too slow. Two lines of approach lead to improvements. The first is exemplified by Blast, which is used routinely by thousands of scientists. This approach finds short exact 'seed' matches (hits), which are then extended into longer alignments. However, when comparing two very long sequences, FASTA, SIM, Blastn (BL2SEQ), WU-Blast, and Psi-Blast run very slow and need large amounts of memory. SENSEI is somewhat faster and uses much less memory than the above programs, but is currently limited to ungapped alignments. MegaBlast runs quite efficiently with its default gap scores and large seed length of 28 but turns out to have worse output quality and doesn't scale as well to huge sequences.

Another line of approach, exemplified by MUMmer, QUASAR and REPuter, uses suffix trees. Suffix trees suffer from two problems: they are meant to deal with precise matches and are limited to comparison of highly similar sequences (Delcher *et al.*, 1999; Burkhardt *et al.*, 1999; Kurtz and Schleiermacher, 1999). They are very awkward in handling mismatches. The second problem with suffix trees is that they have an intrinsic large space requirement.

We introduce novel seeding schemes and hit-processing methods, which are implemented in our program PatternHunter. On a modern desktop, its running time ranges from seconds for prokaryotic genomes to minutes for *Arabidopsis* chromosomes to hours for human chromosomes, with very modest memory use, and at provably higher sensitivity than the default Blastn.

SELECTING GOOD SEEDS: EXPECT LESS TO GET MORE

A dilemma for a Blast type of search is that large seeds lose distant homologies while small ones creates too many random hits which slow down the computation. We use a new idea that allows us to have a higher probability of a hit in a homologous region, even while having somewhat lower expected number of random hits.

Blast looks for matches of k (default $k = 11$ in Blastn and $k = 28$ in MegaBlast) consecutive letters as seeds. Instead we propose to use *non*consecutive k letters as seeds. We call the relative positions of the k letters a *model*, and k its *weight*.

This seemingly simple change has a surprisingly large effect on sensitivity. An appropriately chosen model can have a significantly higher probability of having at least one hit in a homologous region, compared to Blast's consecutive seed model, even while having a lower *expected* number of hits[†]. For example, in a region of length 64 with 70% identity, Blast's consecutive weight 11 model has a 0.30 probability of having at least one hit in the range, while a nonconsecutive model of the same weight has a 0.466 probability of getting a hit, see Figure 1. On the other hand, the expected number of hits in that region by the Blast consecutive model is 1.07, while the nonconsecutive model expects 0.93 hits. This is because the length 11 model can shift over 54 places within the length 64 window, while the length 18 model has only 47 places to fit. The reason for the increased sensitivity is that the events, of having a match at different positions, become more independent for spaced models. If a model and a shifted copy share many 1s in the same position, then a base mismatch in any of these shared positions will make both matches fail, hence the corresponding matching events are far from independent. Independent events are better at pooling their success probabilities together. Generally, the fewer bases shared by a model and any of its shifted copies, the higher its sensitivity is. Clearly, by this measure, consecutive models are the worst, since shift of 1 shares all but one bases.

For convenience, we denote a model by a 0–1 string, where the 1-positions represent required matches, while the 0s are 'don't cares'. For example, if we use a weight six model 1110111, then `actgact` versus `acttact` is a seed match, as well as `actgact` versus `actgact`. So Blast uses models of the form 1^k. Blast actually matches two or three bytes, each containing four bases, simultaneously, and extends these hits to the left and right. This is fine for the default of $k = 11$ because any length 11 match necessarily contains a match of two bytes, but for k smaller than 11, it will miss some seeds.

Suppose a substring s from the query sequence is similar to a same-length substring t from the target sequence. Let $v = v(s, t)$ be a 0–1 string indicating the matches between s and t, i.e. $v[i] = 1$ if and only if $s[i] = t[i]$. Say that a model m *covers* a binary string u of the same length if $m[i] \leqslant u[i]$, i.e. u has a 1 wherever m has a 1. Then a hit between s and t occurs wherever the model covers a substring of $v(s, t)$.

[†] For statistical purposes, we count overlapping hits separately, while the Blast program ignores hits overlapping the last recorded one.

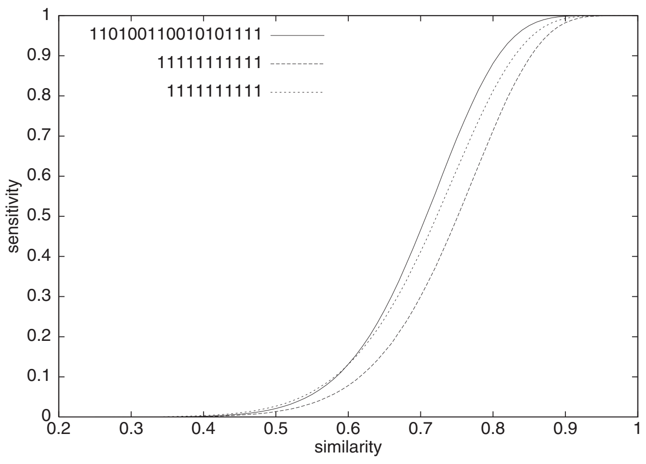

Fig. 1. 1-hit performance of weight 11 spaced model versus weight 11 and 10 consecutive models, coordinates in logarithmic scale.

To evaluate a model, we compute its probability of generating a hit in a fixed length region of given similarity, by dynamic programming (which works fine up to model size 20 and quickly becomes prohibitive in both time and space beyond that). We somewhat arbitrarily chose a region length of 64 because in practice ungapped homologies are typically of size 20–200 bases. Note that the shorter the region length, the more a (longer) spaced model suffers by having fewer places to fit. For weight 11, the most sensitive model is 111010010100110111 with sensitivity 0.467 122 for 70% similarity. Figures 1 and 2 compare the nonconsecutive model 110100110010101111 with Blast's consecutive models. This model, found heuristically before we discovered the dynamic programming algorithm, is slightly suboptimal, with a sensitivity of 0.465 485, and was used in all experiments in this paper for weight 11. The new PatternHunter uses the optimal model 111010010100110111 and will support user-defined, randomly generated, and neighbourhood models in future. Using different models allows PatternHunter to have different outputs in different runs, thus increasing the sensitivity. Neighbourhood models use multiple similar models simultaneously to further increase sensitivity. For each similarity percentage shown on the x-axis, the percentage of regions acquiring at least one hit is plotted on the y-axis as the sensitivity at that similarity level in Figure 1. Recalling our earlier discussion of independence, the (near) optimality of these two models is witnessed by the fact that any shift shares no more than five bases.

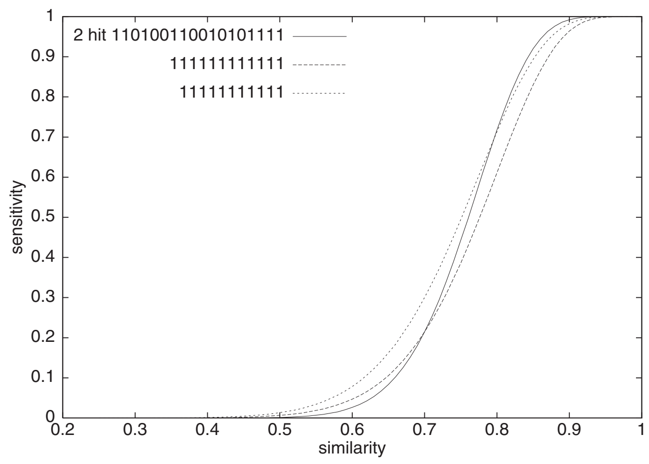

Fig. 2. 2-hit performance of weight 11 spaced model versus single hit weight 11 and 12 consecutive models.

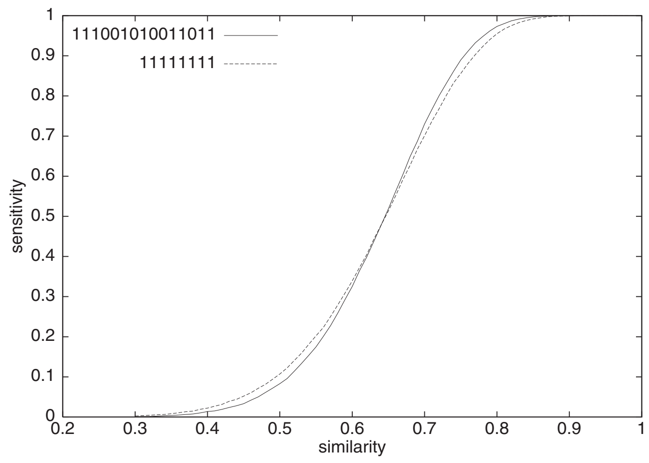

Fig. 3. 1-hit performance of weight 8 consecutive model versus weight 9 nonconsecutive model.

The current 1.4 version of Blast triggers an extension if two disjoint hits are found on the same diagonal within a certain distance (Altschul *et al.*, 1997). The increased selectivity more than offsets the loss in sensitivity, so that it can use a smaller weight model and still generate fewer extensions than an equally sensitive 1 hit model of larger weight. The same can be done with spaced models, with the advantage that hits are no longer required to be disjoint in order to gain a lot of sensitivity. Figure 2 compares the sensitivity of a double hit spaced weight 11 model against single hit weight 11 and 12 consecutive models.

SENSEI uses a default seed size of eight; Figure 3 compares its sensitivity with that of a spaced weight nine model.

The expected number of hits in a region can be easily calculated as in the following Lemma.

LEMMA 1. *The expected number of hits of a weight W, length M model within a length L region of similarity $0 \leqslant p \leqslant 1$, 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. □

Observations

- Figures 1–3 show that the steeper curve of the spaced seed model has *smaller* hit probability in low similarity regions, with respect to the closest consecutive model in terms of sensitivity. In fact, Figures 1 and 3 show that we can use a spaced model of weight 9 to replace a consecutive weight 8 model, gaining sensitivity above 64% similarity, or use a weight 11 spaced model to replace a weight 10 consecutive model gaining sensitivity above 60%. Note that increasing the weight by 1 reduces the number of random hits by approximately a factor of 4.

- It has been brought to our attention, that a related but conceptually different approach, has been applied to ungapped homology search in Buhler (2001), and Califano and Rigoutsos (1995).

  Buhler (2001) applies a random hashing/projection technique known as Locally-Sensitive Hashing (LSH; Indyk *et al.*, 1998), as follows. In each of hundreds of iterations, a newly chosen random hash function is applied to every region of a fixed size (of about 100), and regions mapping to the same value are fully compared. Similar overlapping regions on the same diagonal are then merged into ungapped alignments. Unlike Blast, a long ungapped alignment can only be found if the regions found to be similar cover its whole length. Earlier, a similar idea has been applied in Flash (Califano and Rigoutsos, 1995), which used shorter regions. Both papers focused on covering a homology entirely with hits, instead of doing hit-extension in Blast style. See also Buhler and Tompa (2002)

  Retrospectively, our carefully chosen deterministic spaced seed model maximizes the chance of any HSP to contain *at least one* seed, while minimizing random hits. Experiments show that SENSEI (States, 2000) (which is also limited to ungapped alignment), at its default size 8 seed, is faster than LSH.

PATTERNHUNTER: COMPARE WHOLE GENOMES ON YOUR DESKTOP

We have implemented PatternHunter in Java using the spaced seed model and various algorithmic improvements using advanced data structures.

Hit generation

PatternHunter uses a method of generating hits similar to that of MegaBlast[†]. For each position in one sequence, compute an index from fitting the model at that particular position. This index is 2 × weight bits long (two bits per base). Then do a lookup in a big table which gives the first position in the other sequence where the model matches. This gives the first hit. Subsequent hits are found using another table, which for each position gives the next position where the model matches. This table requires one int (four bytes) per base. If the number of hits found for one index is large then because of the relative cost of computing the index becomes negligible.

For each hit, we look up its diagonal in another hashtable, the hit table, to find the rightmost matched position on that diagonal. If this position is to the right of the hit then we ignore the hit as being part of an already found match.

If the double hit option is chosen then in the absence of a recent hit on the same diagonal, we merely record the new one.

Hit extension

Next we extend this hit in a greedy fashion to the left and right, stopping when the score drops by a certain amount. If the resulting segment pair has a score below a certain minimum, then we ignore it, else we have a Highscoring Segment Pair (HSP). The position of the last comparison, which reached the dropoff score, is stored in the hit table, so that future equivalent hits within this HSP can be recognized as redundant.

Gapping extension

To find the best way to extend an HSP to the left across gaps, we try all candidates from a diagonal-sorted set of recently found HSPs, after adding to this set some new HSPs found by local hit generation. We use a variation of a red–black tree to implement the set of HSPs sorted by diagonal. HSPs are inserted in the tree once an optimal gapped alignment to its left is found, and retired from the tree once newly generated HSPs are too far beyond its right endpoint to make use of it. Retired alignments are put into a priority queue according to their scores.

The local hit generation finds triple hits of the small model 1101 in a limited length region to the left of the HSP and stores them in the tree if they have a certain minimum length.

For each candidate HSP to gap a newly found HSP to, we compute the gapping cost as the sum of the gap open plus gap extension penalties plus the cost of adjusting either HSP in size to make a perfect fit. From this data

[†] Note that (Mega)Blast computes its indices more efficiently by packing four bases into one byte.

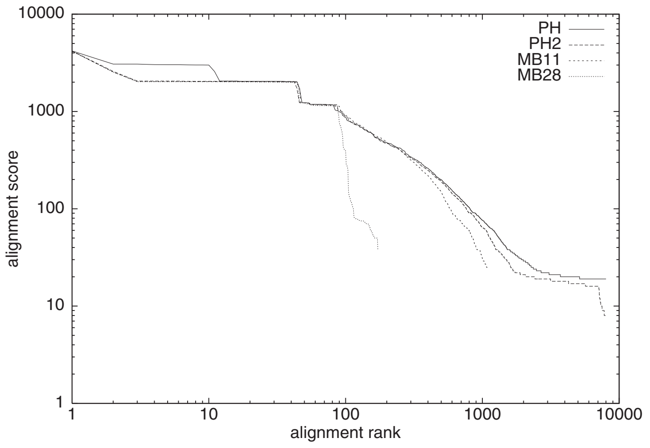

Fig. 4. Input: *H. influenza* and *E. coli*. Run times are shown in Figure 7. Score is plotted as a function of the rank of the alignment, with both axes logarithmic. MegaBlast (MB28) misses over 700 alignments of score at least 100. MB11 is MegaBlast with seed size 11 (50 times slower and 10 times more memory use than PH2), indicating the missed alignments by MB28 are mainly due to seed size.

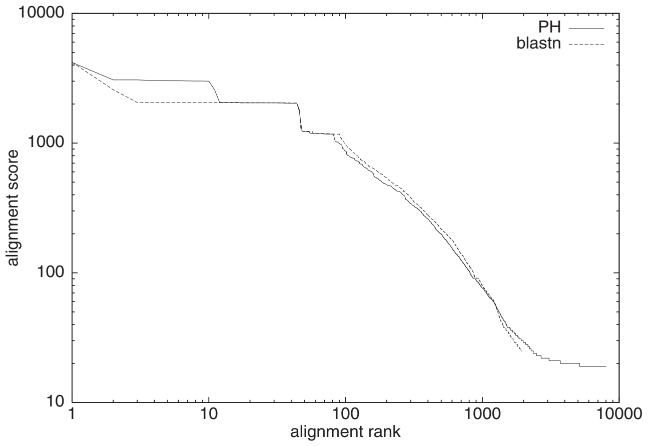

Fig. 5. Input: *H. influenza* and *E. coli*. Run times are shown in Figure 7. PatternHunter produces better quality output than Blastn while running 20 times faster.

the best HSP, if any, to link to, is chosen and used to compute the optimal partial alignment score. Overlapping alignments are not reported.

IMPLEMENTATION

PatternHunter is implemented in Java, hence platform-independent. We have made great efforts to write simple, clean, and short code. The executable size of PatternHunter is 40 kB, only 1% of the size of Blast, while offering a large fraction of its functionality.

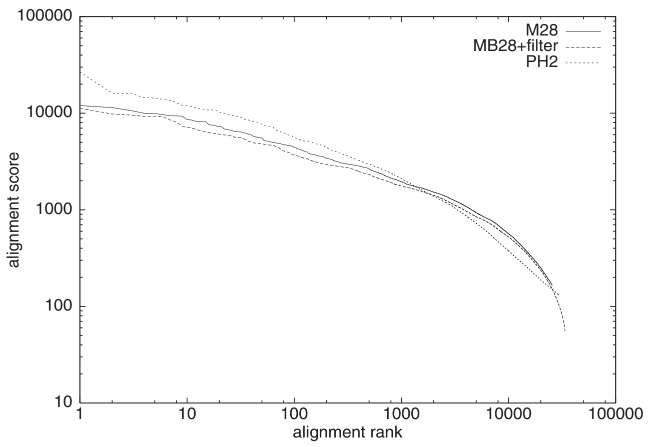

Fig. 6. Input: *A. thaliana* chr 2 and 4. Run times are shown in Figure 7. PatternHunter (PH2) outscores MegaBlast in one sixth of the time and one quarter the memory. Both programs used MegaBlast's nonaffine gap costs (with gapopen 0, gapextend -7, match 2, and mismatch -6) to avoid MegaBlast from running out of memory. For comparison we also show the curve for MegaBlast with its default low complexity filtering on, which decreases its runtime more than 6-fold to 3305s.

With two input sequences of length n and m, $m > n$, and model weight 11, PatternHunter uses 16 Mbytes for the hashtable entries and $4n$ bytes for the array of next-occurrence indices. Together with $n + m$ bytes to store the inputs, and a variable amount for other data structures, PatternHunter uses about $(4^{W+1} + m + (5 + \epsilon)n)$ bytes of memory, in addition to the Java Virtual Machine's memory footprint (which can range from half a Mbyte to a dozen Mbytes). Here, W is the model weight, and $\epsilon > 0$ is a small constant depending on the options PatternHunter uses. For the default $W = 11$, 4^{W+1} amounts to 16 Mbytes. Typically ϵ is much smaller than 1. We conclude that the reason this theoretical estimation is not exactly reflected in Figure 7 is that the Java virtual machine does not do garbage collections often when there is still enough free memory in the heap to use. Our recent experiments on larger human chromosomes demonstrated that our estimation in memory is accurate. This allows PatternHunter to work on complete chromosomes of the human genome on a modern desktop computer with 2 Gbytes of memory.

RESULTS

Here, we report several test runs of PatternHunter with comparison to other programs. Since the Blast family, especially the newly improved Blastn, is the industry standard, and widely recognized for its sensitivity (Blastn, SENSEI) and speed (Blastn, MegaBlast), we limit ourselves to comparison with these programs. All experiments are performed on a 700 MHz Pentium III PC with 1 Gbyte of memory. The table in Figure 7 compares PatternHunter with the latest versions of Blastn and MegaBlast, downloaded from the NCBI website on July 9, 2001. All programs were run without filtering (bl2seq option -F F) to ensure identical input to the actual matching engines. The table in Figure 8 compares PatternHunter with SENSEI; note that SENSEI, as currently available, does not do any gapped alignments. One may suspect that PatternHunter sacrifices quality for speed. Figures 4–6 show the opposite. In Figure 4, MegaBlast using seed weight 28 (MB28) misses over 700 high scoring alignments. Using the same parameters, PatternHunter outputs better results than Blastn, is 20 times faster and uses one tenth the memory, (Figure 5). Notice the quick growth of Blastn/MegaBlast time/space requirements, indicating poor scalability. Only MegaBlast (MB28) at its default affine gap costs allowed us to continue the comparison, without running out of memory, but with vastly inferior output quality compared to PatternHunter (PH2), which uses only one fifth the time and one quarter the space, (Figure 6). Finally, PatternHunter has recently been used to compare the human genome with 16 million reads of the unassembled mouse genome, a total of nine billion base pairs. PH2, weight 12, finishes in 20 (Pentium III) CPU-days. PH2, weight 11, finishes in 80 CPU-days. The results are available at http://genome.cse.ucsc.edu/. While MegaBlast is designed for high speed on highly similar sequences and Blastn for sensitivity, PatternHunter simultaneously exceeds Blastn in sensitivity, MegaBlast in speed (on long sequences), and both in memory use. Written in Java, it runs any genome anywhere.

ACKNOWLEDGEMENTS

We are very grateful to Zheng Zhang, whose comments saved us from various pitfalls. He also brought Buhler (2001) and Indyk *et al.* (1998) to our attention. We thank Larry Miller for working on related projects, David Lipman for comments and bringing Califano and Rigoutsos (1995) to our attention, Xiaoqui Huang for his comments, Dan Brown and Jim Kent for letting us use PatternHunter to mouse–human genome comparison, David States for providing information on SENSEI, and the referees for their useful suggestions. B.M. was supported by NSERC Research Grant OGP0046506. J.T. was supported by BSI. M.L. was supported in part by NSERC Research Grant OGP0046506, a CITO grant, and NSERC's Steacie Fellowship. Both B.M. and M.L.'s work was done when they were at University of Waterloo.

Seq1	Size	Seq2	Size	PH	PH2	MB28	Blastn
M. pneumoniae	828 K	*M. genitalium*	589 K	10 s/65 M	4 s/48 M	1 s/88 M	47 s/45 M
E. coli	4.7 M	*H. influenza*	1.8 M	34 s/78 M	14 s/68 M	5 s/561 M	716 s/158 M
A. thaliana chr 2	19.6 M	*A. thaliana* chr 4	17.5 M	5020 s/279 M	498 s/231 M	21 720 s/1087 M	∞
H. sapiens chr 22	35 M	*H. sapiens* chr 21	26.2 M	14 512 s/419 M	5250 s/417 M	∞	∞

Fig. 7. Performance comparison: if not specified, all with match 1, mismatch −1, gap open −5, gap extension −1. PH denotes PatternHunter with seed weight 11, PH2 denotes same with double hit model (sensitivity similar to Blast's single hit size 11 seed, Figure 2) MB28 denotes MegaBlast with default seed size 28, and default affine gap penalties. Blastn (via BL2SEQ) uses default seed size 11. Table entries under PH, PH2, MB28 and Blastn indicate time (s) and space (Mbytes) used; ∞ means out of memory or segmentation fault.

Seq1	Size	Seq2	Size	PH(9)	PH(11)	SENSEI
E. coli	4.7 M	*H. influenza*	1.8 M	279 s/67 M	34 s/78 M	780 s/64 M
A. thaliana chr 2	19.6 M	*A. thaliana* chr 4	17.5 M	677 m/282 M	84 m/279 M	781 m/415 M

Fig. 8. PatternHunter with seed weights 9, 11, 1-hit model versus SENSEI's weight 8 seed. SENSEI only does ungapped alignments. PatternHunter's weight 9 spaced seed has higher single-hit sensitivity than SENSEI's 8 as shown in Figure 3.

REFERENCES

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

Altschul,S.F., Madden,T.L., Schäffer,A.A., Zhang,J., Zhang,Z., Miller,W. and Lipman,D.J. (1997) Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res.*, **25**, 3389–3402.

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

Buhler,J. and Tompa,M. (2002) Finding motifs using random projections. *J. Comput. Biol.*, to appear

Burkhardt,S., Crauser,A., Lenhof,H.-P., Rivals,E., Ferragina,P. and Vingron,M. (1999) q-gram based database searching using a suffix array. In *Third Annual International Conference on Computational Molecular Biology*. Lyon, pp. 11–14.

Califano,A. and Rigoutsos,I. (1995) FLASH: fast look-up algorithm for string homology. *Technical Report*, IBM, T.J.Watson Research Center.

Delcher,A.L., Kasif,S., Fleischmann,R.D., Peterson,J., White,O. and Salzberg,S.L. (1999) Alignment of whole genomes. *Nucleic Acids Res.*, **27**, 2369–2376.

Gish,W. WU-Blast 2.0. Website: http://blast.wustl.edu.

Huang,X. and Miller,W. (1991) A time-efficient, linear-space local similarityalgorithm. *Adv. Appl. Math.*, **12**, 337–357.

Indyk,P. and Motwani,R. (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of 30th Annual ACM Symposium on Theory Computing*. Dallas, TX.

International Human Genome Sequencing Consortium (2001) Initial sequencing and analysis of the human genome. *Nature*, **409**, 860–921.

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

Kurtz,S. and Schleiermacher,C. (1999) REPuter—fast computation of maximal repeats in complete genomes. *Bioinformatics*, **15**, 426–427.

States,D. SENSEI website: http://stateslab.wustl.edu/software/sensei/.

Tatusova,T.A. and Madden,T.L. (1999) Blast 2 sequences—a new tool for comparing protein and nucleotide sequences. *FEMS Microbiol. Lett.*, **174**, 247–250.

Venter,J.C. *et al.* (2001) The sequence of the human genome. *Science*, **291**, 1304.

Zhang,Z., Schwartz,S., Wagner,L. and Miller,W. (2000) A greedy algorithm for aligning DNA sequences. *J. Comput. Biol.*, **7**, 203–214.