Rapid homology search with two-stage extension and daughter seeds

Miklós Csűrös[1] and Bin Ma[2]

[1] Department of Computer Science and Operations Research
Université de Montréal
C.P. 6128, succ. Centre-Ville, Montréal, Qué., Canada, H3C 3J7
csuros@iro.umontreal.ca
[2] Department of Computer Science
University of Western Ontario
London, Ont., Canada, N6A 5B7
bma@csd.uwo.ca

Abstract. Using a seed to rapidly "hit" possible homologies for further examination is a common practice to speed up homology search in molecular sequences. It has been shown that a collection of higher weight seeds have better sensitivity than a single lower weight seed at the same speed. However, huge memory requirements diminish the advantages of high weight seeds. This paper describes a two-stage extension method, which simulates high weight seeds with modest memory requirements. The paper also proposes the use of so-called daughter seeds, which is an extension of the previously studied vector seed idea. Daughter seeds, especially when combined with the two-stage extension, provide the flexibility to maximize the independence between the seeds, which is a well-known criterion for maximizing sensitivity. Some other practical techniques to reduce memory usage are also discussed in the paper.

1 Introduction

An important task in the analysis of molecular sequences is the search for *local alignments*, formed by pairs of substrings from two sequences, which score high according to some string similarity metric. Local alignments are the "unit operations" in comparative genomics [1], where sequence conservation and lack of it are used to reason about evolutionary relationships and biological function. For instance, even alignments between different species' genomes [2] rely on anchors, which are local alignments between the genomes that restrict the search space for whole-genome alignments.

The importance of the local alignment problem led to a large body of research, starting in the early 1980s with the algorithm of Smith and Waterman [3], later improved by Gotoh [4]. The Smith-Waterman-Gotoh algorithm uses dynamic programming to find all local alignments scoring above a fixed threshold in $O(|S| \cdot |T|)$ time for two sequences S and T over a finite alphabet Σ. For genomic sequences, Σ is the DNA alphabet of size four. While the speed of a full sensitivity search can be improved by a logarithmic factor [5], a full-scale search that involves sequences with several million letters cannot be carried out in a reasonable time frame. For large alignment problems, other solutions are needed that may sacrifice some sensitivity for speed, i.e., that may miss some local alignments but run reasonably fast. Heuristic search programs such as FASTA [6] and BLAST [7] were introduced at the end of the 1980s. They rely on the so-called *hit-and-extend* heuristic, which can be implemented using hashing and lookup tables, introduced in this context by Wilbur and Lipman [8]. The majority of modern local alignment programs [9–13] exploit some variant of this idea. Some more recent alternatives are based on suffix trees [14, 15].

This paper concentrates on hit-and-extend methods. Hit-and-extend methods rely on a hash function $h\colon \Sigma^\ell \to \{0, \dots, H-1\}$. Local alignments are found by first identifying *hits*, which are pairs of positions (i, j) where $h(S[i..i+\ell-1]) = h(T[j..j+\ell-1])$. The most obvious choice for hashing is to use the identity function, when hits are defined by identical substrings of length ℓ, called ℓ*-mers*. In fact, this strategy is used by BLAST. All the hits can be found efficiently by using a lookup table that stores the *occurrence*

lists $\mathsf{Occ}(g) = \{i\colon h(S[i..i+\ell-1]) = g\}$ for every key g. Subsequently, hits are detected and extended by consulting the occurrence list for $h(T[j..j+\ell-1])$ in each position j. Figure 1 outlines this concept. This strategy is often called "seeding" and the hash function or its representation is called as a seed. The sensitivity of a seed measures its ability to hit a homology, and the specificity of a seed characterizes its ability to filter out a random region.

```
     Algorithm HIT-AND-EXTEND
     Input sequences S, T;
     hash function h: Σ^ℓ → {0,...,H−1}
H1   for i = 1,...,|S| − ℓ + 1 do
H2       set g ← h(S[i..i+ℓ−1])
H3       add i to the list Occ(g)
H4   for j = 1,...,|T| − ℓ − 1 do
H5       set g ← h(T[j..j+ℓ−1])
H6       process the hits (i, j): i ∈ Occ(g)
```

```
     Algorithm X-DROP
     Input sequences S, T; start (i, j); allowed drop X
X1   set s ← 0; max ← 0
X2   while s > max − X and i ≤ |S| and j ≤ |T| do
X3       if S[i] = T[j] then s ← s + 1 else s ← s − 1
X4       if s > max then set max ← s
X5       set i ← i + 1; j ← j + 1
X6   report max
```

Fig. 1. Basic hit-and-extend procedure. Algorithm HIT-AND-EXTEND outlines the method. Hits are extended in Line H6 by exploring the dynamic programming table around the hits. X-DROP is a popular extension algorithm, used in BLAST [7, 9] and many other alignment programs. The extension is shown only in the forward direction. An analogous extension process is carried out in the reverse direction where i and j decrease by 1 in every step.

It was recently discovered [12] that *spaced seeds* provide very good sensitivity and specificity. A spaced seed is defined by a set $\mathcal{S} = \{s_1, \ldots, s_k\} \subseteq \{1, \ldots, \ell\}$. In practice, a spaced seed is often denoted by the characteristic vector for the set, defined as the length-ℓ binary string in which the bits at the positions specified by the seed have value 1. The corresponding hash function concatenates the characters in positions specified by the seed, and encodes the resulting string $u[s_1] \cdot u[s_2] \cdots u[s_k]$ by an integer in the range $\{0, \ldots, |\Sigma|^k - 1\}$. Such a seed is called an (ℓ, k)-seed, and has *weight* k. The initial discovery led to a number of results on selecting spaced seeds [16–18] in various statistical or empirical alignment models. Additional references with a thorough discussion are offered in [19]. Spaced seeds or similar indexing constructs have been studied also in the contexts of lossless filtration [20, 21] and sequencing by hybridization [22].

There exist several generalizations of spaced seeds, which include multiple seeds [13, 23], and vector seeds [10, 24]. Multiple seeds are a set of carefully selected spaced seeds $\mathcal{S}_1, \ldots, \mathcal{S}_M$. The set of hits for such a set is the union of the hits found by every single seed. A vector seed is defined by a vector of non-negative weights $(w_1, \ldots, w_\ell)$ and a threshold t: there is a hit at (i, j) if $t \leq \sum_{\delta=1}^{\ell} w_\delta I\{S[i+\delta-1] = T[j+\delta-1]\}$, where $I\{C\}$ is 1 if and only if condition C is true, otherwise it is 0. (The slightly more general definition of [24] allows for a scoring matrix.) A vector seed can be viewed as a well-structured set of multiple seeds.

The time complexity increase of using multiple seeds can be offset by using higher-weighted seeds. It was shown that higher-weighted multiple seeds and vector seeds may offer superior sensitivity [13, 24] to that of a single seed at the same specificity. However, they can hardly reach their theoretical potential due to their memory requirements. In case of multiple seeds [13], a lookup table is constructed for every seed. Vector seeds rely on a hash table for the spaced seed defined by the positions with non-zero weights. As a consequence, memory usage is exponential in the number of positive weights. Vector seeds with widely varying weighting schemes proved prohibitive due to their demands on memory.

We first propose in Section 2 a novel two-stage extension procedure that improves the efficiency of hit-and-extend methods. Rather than being a trivial heuristic, extensive optimization is needed to maximize the sensitivity of the two-stage extension. The concept of daughter seeds is introduced in Section 3. Daughter seeds allow us to attain or surpass the sensitivity and speed of multiple and vector seeds, and pose only modest demands on memory. We discuss the advantages of combining two-stage extension and the daughter seeds. Section 4 explores some practical techniques of space reduction, which include an implementation of 11-mer based hashing with 1.5 bytes per base pair for the purposes of comparing mammalian-sized genomes. Section 5 concludes the paper.

2 Two-stage extension

2.1 Average complexity of the classic hit-and-extend method

In this study, we restrict our attention to gapless local alignments. The presented techniques are, however, relevant also for gapped alignments, as most programs perform gapped extension only if a high-scoring gapless alignment is found. For simplicity, we consider the alignment scoring policy that rewards an identity with $+1$ and penalizes a mismatch with -1. Thus, without loss of generality, each local alignment between $S[i..i+n-1]$ and $T[j..j+n-1]$ can be represented by a 0-1 string R of length n, where $R[k]=1$ if and only if $S[i+k-1]$ matches $T[i+k-1]$. Let n' be the number of ones in R. Then the score of the alignment is $(2n'-n)$. The *similarity* of the local alignment is then ratio n'/n.

If $S[i..i+n-1]$ and $T[j..j+n-1]$ are random unrelated sequences, then the similarity is expected to be $\beta=\sum_{a\in\Sigma}p(a)q(a)$, where $p(a)$ and $q(a)$ are the background frequencies for the letter a in the two sequences. For DNA sequences with alphabet size 4, $\beta=\frac{1}{4}$ if the letter occurrences are uniform random in at least one of S and T. For simplicity, we mostly focus on such a model of random sequences. Nonetheless, the analyses can be easily extended to arbitrary background frequencies.

A heuristic local alignment method can be assessed by evaluating its specificity and sensitivity. *Specificity* is measured by the average running time on random unrelated sequences. *Sensitivity* is measured by the probability of detecting a homology under a probabilistic model of homologies.

Since the introduction of spaced seeds, there has been much work on finding variants of hit conditions and hash functions to gain better sensitivity and specificity. In this section we scrutinize the extension instead. The usual method is to extend a hit in each of the two directions along the diagonal until the score drops by a specified amount. In each direction, the position where the maximum score is reached is recorded and gives the extent of the local alignment. Figure 1 shows the X-DROP extension procedure in one direction.

If we ignore the boundary effects of S and T, the average running time of a hit-and-extend method for random sequences is $f\times t\times|S|\times|T|$, where f is the probability of a hit at a fixed position pair (i,j), and t is the average time spent on a hit extension. The probability f is called the *false positive rate* in [24]. In what follows, we analyze t more closely for the X-drop algorithm of Fig. 1. In Line X3, the score decreases with an expected value of $(1-2\beta)$ in each step. Therefore the extension will stop after around $X/(1-2\beta)$ steps. The following lemma formalizes this argument.

Lemma 1. *Suppose that $\beta<1/2$ holds for the match probability, and* X-DROP *is invoked with a positive integer X. Let $n=\min\{|S|,|T|\}$. If τ denotes the number of times the loop body X3–X5 is executed, then*

$$\lim_{n\to\infty}\mathbb{E}\tau=\frac{X-\beta\Big(1-\big(\frac{\beta}{1-\beta}\big)^X\Big)}{1-2\beta}=\frac{X-\sum_{t=1}^{X}\big(\frac{\beta}{1-\beta}\big)^t}{1-2\beta}. \tag{1}$$

Proof. Let $Y_1,Y_2,\ldots$ be the series of ±1-valued random variables that track the changes of the score s in Line X3 so that $s=\sum_{\delta=1}^{t}Y_\delta$ after t comparisons. Formally, $Y_\delta=1$ if $S[i+\delta-1]=T[j+\delta-1]$, otherwise $Y_\delta=-1$, and let $s(t)=\sum_{\delta=1}^{t}Y_\delta$. Clearly, $\{Y_\delta\}$ are independent and identically distributed (iid) random variables with expected value $\mathbb{E}Y=-(1-2\beta)$, and $s(\delta)$ form a simple random walk. The number τ is a stopping time for $Y_1,Y_2,\ldots$ and thus Wald's Equation applies [25]:

$$\mathbb{E}s(\tau)=(\mathbb{E}Y)(\mathbb{E}\tau). \tag{2}$$

We need therefore to determine $\mathbb{E}s(\tau)$, the expected final value of the score s when the condition fails in Line X2. The key idea is to consider *ladder points* [26] which are the places where max is updated in Line X4. Specifically, the ladder points $\tau_0=0,\tau_1,\tau_2,\ldots$ are defined in the following manner. For every $m>0$, $\tau_m=\min\{t\colon s(t)=m\}$, with the convention that if $s(t)$ never reaches m, then $\tau_m=\infty$. Let L be the maximum value of $s(t)$ before the score drops by X for the first time: $L=\min\{m\colon s(t)=m-X \text{ for some } \tau_m<t<\tau_{m+1}\}$. The stopping time is $\tau=\min\{t\colon \tau_L<t, s(t)=L-X\}$. Consequently, $s(\tau)=L-X$ and the value $\mathsf{max}=L$ is returned at the end of the extension. We need to compute $\mathbb{E}L$ to obtain $\mathbb{E}\tau$ through Eq. (2). Consider the probability $\mathbb{P}\{L>0\}$. It equals the probability that the random

walk $s(t)$ attains the value 1 before $-X$. By standard results on random walks [25], this probability equals $\rho = \frac{\alpha^{-X}-1}{\alpha^{-X-1}-1}$ with $\alpha = \frac{\beta}{1-\beta}$. Since $\{Y_\delta\}$ are independent, the distribution of L is a (shifted) geometric distribution, and thus $\mathbb{P}\{L = m\} = (1-\rho)\rho^m$ for all $m \in \mathbb{N}$. Hence, $\mathbb{E}L = \frac{\rho}{1-\rho} = \frac{\alpha}{1-\alpha}(1-\alpha^X)$. By Eq. (2),

$$\mathbb{E}\tau = \frac{X - \mathbb{E}L}{1-2\beta} = \frac{X - \frac{\alpha}{1-\alpha}(1-\alpha^X)}{1-2\beta},$$

and Eq. (1) follows by plugging in the value of α. □

Corollary 1. *If the alphabet is of size 4 and one of the sequences is uniform random, then $\beta = \frac{1}{4}$, and the expected number of comparisons at a hit is $4X - 2 + o(1)$.*

Proof. Substituting $\frac{1}{4}$ for β in Lemma 1, $\mathbb{E}\tau = 2X - 1 + 3^{-X}$. Noticing that the extension is done in both directions, the corollary is proved. □

With a typical choice $X = 16$, a hit extension entails approximately 62 character comparisons on average.

2.2 Two-stage hit extension

We propose the following two-stage extension process. Let $\mathcal{S} = \{s_1, \ldots, s_k\}$ be an (ℓ, k)-seed, and let $\mathcal{S}' = \{s'_1, \ldots, s'_m\}$ be a set of positive integers with $\mathcal{S} \cap \mathcal{S}' = \emptyset$. Furthermore, let $0 < t \leq m$ be a threshold. The triple $(\mathcal{S}, \mathcal{S}', t)$ defines a *relaxed seed* employed in the following manner. Hits are found as if the spaced seed $\mathcal{S}$ were used. When a hit is found, the positions of $\mathcal{S}'$ are tested, and the full extension is performed if at least t matches are found. In particular, let (i, j) be a hit position. Full extension is performed only if $S[i + s' - 1] = T[j + s' - 1]$ for at least t of $s' \in \mathcal{S}'$. A relaxed seed may significantly increase the specificity, which can be seen in Theorem 1. As we will see in Table 1, the sensitivity of a relaxed seed varies very much for different choices of $\mathcal{S}'$ even with the same size and threshold. Therefore, the optimization of the positions in $\mathcal{S}'$ should be done together with $\mathcal{S}$.

Theorem 1. *Suppose that the two-stage extension method is employed with $|\mathcal{S}'| = m$. Let $b(m,t) = \sum_{i=t}^{m} \binom{m}{i} (\frac{1}{4})^i (\frac{3}{4})^{m-i}$. The average number of character comparisons performed during a bi-directional hit extension is $C = \big(m + b(m,t)(4X-2)\big) + o(1)$.*

Proof. The preliminary test on $\mathcal{S}'$ compares m pairs of characters. A full extension is performed with probability $b(m,t) \leq 1$. A full extension performs an average of $(2X - 1 + 3^{-X})$ comparisons in each of the forward and backward directions. □

Seed	Sens.(64)	Sens.(100)	C	T
`111001001001010111`	0.618	0.838	62	4
`111010010100110111`	0.451	0.685	62	1
`1111111111`	0.391	0.578	62	4

Seed & threshold		Sens.(64)	Sens.(100)	C	T
`111xx1xx1x01010111x`	3	0.555	0.791	14.50	0.94
`x1110x10x10x1010111`	2	0.550	0.787	20.22	1.30
`111001001001010111xxxx`	2	0.528	0.777	20.22	1.30

Table 1. Comparison of some relaxed and spaced seeds. Relaxed seeds are encoded by `0`–`1`–`x` strings: position i has `1` if it is in $\mathcal{S}$, and it has `x` if it is in $\mathcal{S}'$. Sensitivity values are calculated at a 70% similarity level for homology regions of lengths 64 and 100. Column C shows the expected number of comparisons in hit extension, when $X = 16$, and column T lists the expected time spent on finding and extending hits, defined as C times the false positive rate. T is normalized for weight-11 seeds. The two spaced seeds on the left are the most sensitive weight-10 and weight-11 seeds. (Notice that they are much better than a 10-mer.) The table on the right-hand side lists some relaxed seeds. It illustrates that the placement of relaxed positions has a non-negligible effect on sensitivity.

Sensitivity is assessed in the following manner. Let R be a binary representation of a homology region with a given similarity. In order to have a hit with the spaced seed $\mathcal{S}$, R has to contain a substring u such

that $\forall s \in \mathcal{S}\colon u[s] = 1$. In order to have a hit with the relaxed seed $(\mathcal{S}, \mathcal{S}', t)$, R has to contain a substring u such that $\forall s \in \mathcal{S}\colon u[s] = 1$ and $\sum_{j=1}^{m} u[s'_j] \geq t$. The sensitivity is defined to be the hit probability under a specific probabilistic model for homologies. The first such model was introduced in [12]: it imposes that local alignments are created by independent equiprobable substitutions. Here we consider similarity patterns drawn uniformly from the set of length-n binary strings in which there is a 1 in exactly k positions. Computing the sensitivity of spaced seeds in a similar model was considered by Kucherov et al. [27]. Theirs and earlier algorithms for computing the sensitivity of spaced seeds [16, 18] can be readily adapted to relaxed seeds. As an alternative, one can convert a relaxed seed to an equivalent set of multiple seeds and compute the sensitivity by employing the algorithm in [13] that calculates the sensitivity of an arbitrary set of seeds.

Table 1 compares relaxed and spaced seeds. It turns out that the sensitivity of a $(k-1)$-weight seed can be approached while the running time stays close to that of a weight-k spaced seed. It is noteworthy that the last two seeds, `x1110x10x10x1010111` and `1110010010010101111xxxx` have the same $\mathcal{S}$, same size $|\mathcal{S}'|$ and same threshold t. At the same time, they have very different sensitivities. The example demonstrates that the two-stage extension is not a trivial extension heuristic. Indeed, it can be fully profited of only after a meticulous optimization step, in which the threshold and the relaxed positions are selected. This observation is epitomized by the extreme case of a relaxed seed $(\mathcal{S}, \mathcal{S}', t)$ where $t = |\mathcal{S}'|$. This relaxed seed is equivalent to the spaced seed $\mathcal{S} \cup \mathcal{S}'$, and the necessity to optimize the spaced seed is well-known [12].

2.3 Implementation and memory usage

The data structure for the basic algorithm of Fig. 1 has to support the operation $\textsc{Add}(g, i)$ that records the position i as one belonging to $\mathsf{Occ}(g)$, and the operation $\textsc{reportAll}(g)$ that returns the list $\mathsf{Occ}(g)$ as a set. For a spaced seed with weight k, a rather straightforward implementation was introduced in [12]. An integer array, *head*, of length 4^k was used to record the first occurrence of each hash value. Then another integer array, *next*, of length $|S|$ is used to retrieve all the other occurrences. *next*$[i]$ records the next occurrence of the same hash value as position i. The two arrays *head* and *next* form a hash table that requires memory for $(4^k + |S|)$ integers. In a direct manner, a relaxed seed $(\mathcal{S}, \mathcal{S}', t)$ can be implemented by relying on the hash table for the spaced seed $\mathcal{S}$.

3 Daughter seeds

The vector seed idea [24] is very effective for improving sensitivity. Every vector seed corresponds to a particular set of ordinary spaced seeds defined as follows. Let $(w_1, \ldots, w_\ell)$ be the weights and t be the threshold of the vector seed. Let $\mathcal{P} = \{\delta\colon w_\delta > 0\}$ be the set of positively weighted positions, and $K = |\mathcal{P}|$. The vector seed is equivalent to the set of seeds $\{\mathcal{S}_1, \ldots, \mathcal{S}_M\}$ where $\mathcal{S}_i$ are the subsets of $\mathcal{P}$ in which $t \leq \sum_{\delta \in \mathcal{S}_i} w_\delta$. For convenience, we call $\mathcal{P}$ the *parent seed*, and the multiple seeds produced from the parent seed are called *daughter seeds*. When all vector weights are 0 or 1, daughter seeds are generated from the parent by removing up to $(K - t)$ elements from the parent's set of sampled positions. In fact, an equivalent set can be created by removing exactly that many positions.

Our relaxed seeds also define sets of daughter seeds: a relaxed seed $(\mathcal{S}, \mathcal{S}', t)$ is equivalent to the family of $(k + t)$-element subsets of the parent seed $\mathcal{S} \cup \mathcal{S}'$, in which only t elements of $\mathcal{S}'$ are present. The vector seed is different from the multiple seeds introduced in [13], which selects seeds from the complete seed space by a greedy algorithm. The advantage of multiple seeds over the vector seed is that the selected seeds are not dependent on each other. As a result, more local alignments may be hit by a constant number of seeds. On the other hand, multiple seeds require one hash table for each seed, which increases the memory requirements, and therefore only a few number of seeds can be used in practice. The vector seed only requires one hash table for the parent seed. As pointed out in [24], memory will still be a problem when there are more than 14 positive weights.

In what follows, we examine daughter seed sets without constraints on which positions may vary, otherwise imposed by vector seeds and relaxed seeds. We show that daughter seeds can have superior sensitivity to

those of spaced seeds and vector seeds with comparable specificity, while using a reasonable amount of memory (one hash table for the parent seed).

The problem of selecting an optimal set of daughter seeds is likely to be intractable, based on NP-hardness results of selecting multiple seeds [13] or even one optimal seed (M. Li and B. Ma, manuscript in preparation). Computing the false positive rate involves some further complications. Let $\mathcal{S}_1, \ldots, \mathcal{S}_M \subseteq \mathcal{P}$ be a set of daughter seeds. In order to obtain the false positive rate, one needs to count subsets of $\{1, \ldots, \ell\}$ that are supersets of at least one of the seeds, where ℓ is the common seed length (i.e., $\ell = \max \cup_{i=1}^{M} \mathcal{S}_i$). Since neither the sensitivity nor the specificity changes when we add a new daughter seed $\mathcal{D} \supset \mathcal{S}_i$, we can safely assume that the daughter set is *complete* in the sense that if $\mathcal{D} \subseteq \mathcal{P}$ is included, then so are all its supersets $\mathcal{D}'$ with $\mathcal{D} \subseteq \mathcal{D}' \subseteq \mathcal{P}$.

In order to select complete daughter sets, we used the same greedy algorithm as in Li *et al.* [13], adding daughters one by one. Let f be the false positive rate of the parent seed and $K = |\mathcal{P}|$. Suppose now that one daughter seed $\mathcal{S}_1$ is selected by removing one element of $\mathcal{P}$. Obviously, the false positive rate of that single daughter seed increases the false positive rate by $3f$. By adding another daughter seed $\mathcal{S}_2$ of the same weight, the total false positive rate becomes $7f$. Now, consider the choice between adding additional three $(K-1)$-size daughter seeds, or the $(K-2)$-daughter seed $\mathcal{S}_1 \cap \mathcal{S}_2$. Both choices increase the false positive rate by $9f$, and thus the question is which one increases the sensitivity more. Intuitively, adding three $(K-1)$-sized daughter seeds is a better choice, and we observed that behavior in experiments. We thus considered selecting complete daughter sets by first including the parent, then daughter seeds of weight $(K-1)$ until all of them are included, then daughter seeds of weight $(K-2)$ until all of them are included, and so on, down to weight-K' daughters among which not all are selected necessarily. In practice, good daughter seed sets are found with $K = 13$ or $K = 14$, and $K' \geq K - 2$, and thus selecting a quasi-optimal set is feasible. If the number of weight K' daughters is M', then the false positive rate of such a set is $\left(3^{K-K'}M' + \sum_{i=0}^{K-K'+1} \binom{K}{i} 3^i\right) f$. Table 2 shows some daughter seeds. The table illustrates that daughter seeds have better sensitivity than spaced seeds, or practically implementable vector seeds with comparable false positive rates.

Name	Parent weight	Daughters	Sensitivity		False positive rate
MD-3-13	13	$3 \times$ wt12	0.473	*	0.625
MD-5-13	13	$5 \times$ wt12	0.593	*	1.0
MD-11-14	14	$2 \times$ wt12 + $9 \times$ wt13	0.729	*	0.95
VS-12-13	13	$13 \times$ wt12	0.835		2.5
MD-16-14	14	$13 \times$ wt12 + $3 \times$ wt13	0.841	*	2.5
MD-11-13	13	$3 \times$ wt11 + $8 \times$ wt12	0.884	*	4.19
MD-25-14	14	$23 \times$ wt12 + $2 \times$ wt13	0.902	*	3.91
VS-11-12	12	$12 \times$ wt11	0.927		9.25
MD-14-13	13	$12 \times$ wt11 + $2 \times$ wt12	0.941	*	9.25

Table 2. Daughter seeds. Sensitivity values are given for length-64 regions at 70% similarity level. In case of MD seeds (sensitivity marked with *), the values are calculated from simulations involving one million random similarity regions: the accuracy is thus within ± 0.002 with probability 99.9%. The "Daughters" column describes a minimal daughter set (the largest antichain) selected from the complete daughter set: MD-11-13 for instance is a set of 8 weight-12 daughters and 3 weight-11 daughters. The weight-13 parent is the spaced seed `1110110010110101111`; the weight-14 parent is `11110010101100110101111`. Both of these have maximum sensitivity among spaced seeds with equal weight. False positive rate is normalized by that of a weight-11 spaced seed. The VS seeds are vector seeds from [24]. For comparison, the spaced seeds of weight 10 and 11 in Table 1 have sensitivity of 0.618 and 0.451, respectively.

The idea of daughter seeds and the two-stage extension can be combined together to further improve the sensitivity and specificity. Because the two-stage extension of different daughter seeds can be done separately, we can choose different checkpoints, $\mathcal{S}'$, for different daughter seeds. This resembles the multiple seeds idea and gives the flexibility to minimize the dependency between different daughter seeds. Therefore,

the sensitivity can be maximized. For instance, a 17-element set of weight-11 daughters of a weight-13 parent used in conjunction with two-hit extension has a sensitivity of 0.917 at the same speed as MD-25-14, which is faster than a weight-10 spaced seed. More results about the combination will be included in the full version of this extended abstract.

4 Two ideas for reducing memory usage

Daughter seeds rely on a single hash table for the parent seed, and avoid this way the impractical memory requirements of general seed sets. Further memory reductions can be achieved by storing the hash table in a more compact fashion. We need a data structure that supports the operation REPORTALL discussed in §2.3. If k-mers are used, then a suffix array can provide the functionality, which can be implemented using $O(|S|)$ bits [28] in addition to storing the sequence S. Various self-indexing methods [29, 30] promise even better compression by storing S and the indexing structure together. These latter, however, are still impractical for genomic DNA sequence comparisons, since the amount of time they spend on retrieving each hit is measured in milliseconds [30]. Given that the number of hits between two sequences of length 10^8 is about $2.4 \cdot 10^9$ (when using a 11-weight key), the implied running time (in the order of several weeks) is unacceptable. When changing the data structure, even a four-fold increase in the execution of REPORTALL is undesirable, since in a conventional implementation shorter hash keys imply the same increase in running time, with the added benefits of reduced memory usage and improved sensitivity. One can also attempt to eliminate some keys from the table while preserving a good level of sensitivity. Roberts et al. [31] offer an elegant method of selecting the keys to keep, which can lead to a tenfold memory reduction in the overlap detection phase of shotgun sequence assembly, but it is not clear whether the method is equally effective for local alignment tasks that require high sensitivity.

The data structure for the hashing is typically implemented using 32-bit integers [12], as outlined in §2.3. Consequently, a table for a k-weight key occupies $4(4^k+|S|)$ bytes. We describe a way of saving space without much sacrifice in either speed or ease of implementation. In particular, we show how to replace 32-bit integers with $(2k)$-bit integers. For seeds of weights 10–13, this means a memory reduction of 37.5–18.75%. The idea is fairly simple: choose a large integer Q and store the modulo Q remainders in both head and next. The integer value Q is reserved for marking ends of lists, so $\lceil \log_2(Q+1) \rceil$-bit integers suffice. Figure 2 shows the data structure. Since $\textsc{Add}(g, i)$ is called in increasing order of i (cf. Fig. 1), key occurrences are restored correctly.

Theorem 2. *(a)* REPORTALL *of Fig. 2 correctly enumerates the occurrences of a key g, provided that the calls* $\textsc{Add}(g, i)$ *were made in increasing order of i.*

(b) Suppose that S is a uniform random string, and the hash function is such that all keys occur with equal probability. If $Q \gg 1$ and $|S| \to \infty$, then the hash function is evaluated in the loop of Line R5 $(1 - e^{-Q/H})^{-1}$ times on average. If h is defined by a weight-k spaced seed, then, for each occurrence of a key g, $\textsc{ReportAll}(g)$ *performs an expected number of $k + \frac{4/3}{e^{Q/H}-1}$ character comparisons.*

Proof. Claim (a) holds since g occurs in positions $q_1 Q + \mathsf{head}[g], q_2 Q + \mathsf{next}[\mathsf{head}[g]], q_3 Q + \mathsf{next}[\mathsf{next}[\mathsf{head}[g]]], \ldots$ with $q_1 \geq q_2 \geq \cdots$. The variable q always stores the current value of q_i until all the occurrences are enumerated. In order to prove Claim (b), we use the fact that the set positions in which a particular key occurs can be modeled using a Poisson process [32]. Let Δ be the number of times the **while** condition is evaluated in Line R5 before continuing with Line R6. Let X be the distance between the previously found occurrence and the one the loop is looking for. Then $\mathbb{P}\{\Delta = q\} = \mathbb{P}\{X \in [1 + (q-1)Q, qQ]\}$ for all $q = 1, 2, \ldots$. Using the Poisson process approximation, $\mathbb{P}\{\Delta = q\} = (1-\gamma)\gamma^{q-1}$ with $\gamma \approx (1 - H^{-1})^Q \approx e^{-Q/H}$. Consequently, $\mathbb{E}\Delta = \frac{1}{1-\gamma}$ as claimed. For spaced seeds in particular, when the loop condition evaluates to true, an expected number of 4/3 positions are looked at, and when the condition finally fails, k comparisons are made. The expected number of tested positions is therefore $k + \frac{4}{3}(\mathbb{E}\Delta - 1)$, as claimed.

By Theorem 2, using $Q = 4^k - 1$ with a weight-k seed entails an expected number of $(k + 0.77)$ character comparisons. (One can even get away with not comparing all k positions in Line R5 but only some $k' < k$

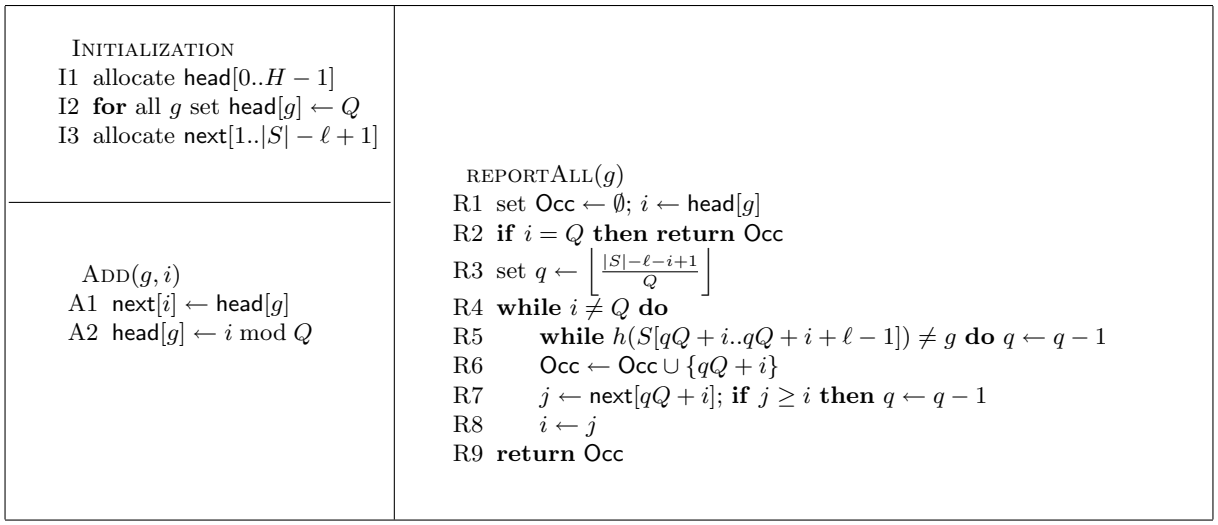

Fig. 2. Data structure for occurrence lists that uses integers in the range $\{0, \ldots, Q\}$. The value Q represents a null pointer.

of them. There is a small chance ($0.25^{k'}$) that we switch to enumerating occurrences of a different hash key g'. The key g', however, matches key g in k' positions, and so the generated hits are not completely arbitrary. The advantage is the lower number of comparisons per hit.) As an alternative to the (mod$Q$) representation, one can avoid the character comparisons by using run-length encoding [33] of the distances between consecutive occurrences, which reduces the space equivalently at the price of having to handle bit vectors of varying length.

Suffix trees or arrays can be employed to enumerate occurrences of k-mers. To our knowledge, there is no efficient way of retrieving occurrences of spaced seeds from a suffix array, and thus their use is limited to k-mers. At the same time, suffix tree-based local alignment methods use at least 12.5–15.6 bytes [15] per base pair. Here we describe a simple method of reducing storage for hashing with k-mers in genome-size local alignments. The idea is to use a hash table for longer $(k + d)$-mers sampled in every $(d + 1)$-th position of S. The occurrences of a key g can be retrieved by listing the occurrences of the keys $a_1 a_2 \cdots a_d \cdot g$, $a_1 \cdots a_{d-1} g a_d$, $\ldots$, $g a_1 a_2 \cdots a_d$ for all choices of $a_1, \ldots, a_d \in \Sigma$. With a judicious choice of d, the running time remains essentially the same, while the memory usage is reduced. Table 3 shows some numerical values, for a typical mammalian chromosome or genome. For instance, about 1.5 bytes/nucleotide suffice for 11-mer based alignment of a whole mammalian genome, if the sequence is stored in 2 bits/nucleotide and the table is stored in less than 10 bits/nucleotide. This memory usage is better than that of the currently most space-efficient suffix array representation [34], which uses 12 bits per nucleotide in addition to the sequence storage. At the same time, the hash table takes considerably less effort to implement.

5 Conclusion

We introduced novel ideas on selecting a structured set of spaced seeds to gain superior sensitivity and speed in hit-and-extend methods of local alignment. Our guideline in designing the techniques was to minimize memory usage, in order to avoid the main obstacle encountered by other methods such as multiple seeds and vector seeds. We described some additional, easily implementable ways to lower memory demands. Memory usage is a key factor in the efficiency of homology search algorithms, and is likely to become even more important in the future. Both the number and total length of DNA sequences in Genbank has doubled about every 17 months since 1983. This rate of increase is comparable to the popular version of Moore's law about computing power doubling every 18 months, and thus powerful heuristics are likely to remain highly valued

table	chromosome		genome	
	int32	mod$Q$	int32	mod$Q$
11-mers	33	22.69	32.07	22.04
every 2^{nd} 12-mer	20	14.375	16.25	11.68
every 4^{th} 14-mer	136	110.5	12	9.75
12-mers	36	27	32.5	24.38
every 2^{nd} 13-mer	32	25	17	13.28

Table 3. Number of bits used per character when storing a k-mer table. The traditional implementation uses 32-bit integers; the implementation of Fig. 2 uses $2k$-bit integers. Notice that $(2k-1)$ or $(2k-2)$ bits suffice for storing occurrences restricted to every second or fourth position, respectively. Sequence lengths are $|S| = 2^{27}$ for a chromosome, and $|S| = 2^{31}$ for a genome, based on the human genome.

in the comparison of molecular sequences. Our methods are memory efficient and offer practical solutions for the alignment of large genomic sequences in terms of speed and sensitivity.

Acknowledgments

This work was supported by grants from the Natural Sciences and Engineering Research Council of Canada, and the *Fonds québécois de la recherche sur la nature et les technologies*.

References

1. Miller, W., Makova, K.D., Nekrutenko, A., Hardison, R.C.: Comparative genomics. Annual Review of Genomics and Human Genetics **5** (2004) 15–56
2. Frazer, K.A., Elnitski, L., Church, D.M., Dubchak, I., Hardison, R.C.: Cross-species sequence comparisons: A review of methods and available resources. Genome Research **13** (2003) 1–12
3. Smith, T.F., Waterman, M.S.: Identification of common molecular subsequences. Journal of Molecular Biology **147** (1981) 195–197
4. Gotoh, O.: An improved algorithm for matching biological sequences. Journal of Molecular Biology **162** (1982) 708–708
5. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A sub-quadratic sequence alignment algorithm for unrestrictred cost matrices. In: Proc. 11th ACM-SIAM Symposium on Discrete Algorithms (SODA). (2002) 679–688
6. Pearson, W.R., Lipman, D.J.: Improved tools for biological sequence comparison. Proceedings of the National Academy of Sciences of the USA **85** (1988) 2444–2448
7. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of Molecular Biology **215** (1990) 403–410
8. Wilbur, W.J., Lipman, D.J.: Rapid similarity searches of nucleic acid and protein data banks. Proceedings of the National Academy of Sciences of the USA **80** (1983) 726–730
9. Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Research **25** (1997) 3389–3402
10. Schwartz, S., Kent, W.J., Smit, A., Zhang, Z., Baertsch, R., Hardison, R.C., Haussler, D., Miller, W.: Human-mouse alignments with BLASTZ. Genome Research **13** (2003) 103–107
11. Ning, Z., Cox, A.J., Mullikin, J.C.: SSAHA: A fast search method for large DNA databases. Genome Research **11** (2001) 1725–1729
12. Ma, B., Tromp, J., Li, M.: PatternHunter: faster and more sensitive homology search. Bioinformatics **18** (2002) 440–445
13. Li, M., Ma, B., Kisman, D., Tromp, J.: PatternHunter II: highly sensitive and fast homology search. Journal of Bioinformatics and Computational Biology **2** (2004) 411–439
14. Delcher, A.L., Kasif, S., Fleischmann, R.D., Peterson, J., White, O., Salzberg, S.L.: Alignment of whole genomes. Nucleic Acids Research **27** (1999) 2369–2376
15. Kurtz, S., Phillippy, A., arthur L. Delcher, Smoot, M., Shumway, M., Antonescu, C., Salzberg, S.L.: Versatile and open software for comparing large genomes. Genome Biology **5** (2004) R12
16. Buhler, J., Keich, U., Sun, Y.: Designing seeds for similarity search in genomic DNA. Journal of Computer and System Sciences **70** (2005) 342–363

17. Choi, K.P., Zhang, L.: Sensitivity analysis and efficient method for identifying optimal spaced seeds. Journal of Computer and System Sciences **68** (2004) 22–40
18. Keich, U., Li, M., Ma, B., Tromp, J.: On spaced seeds for similarity search. Discrete Applied Mathematics **138** (2004) 253–263
19. Brown, D.G., Li, M., Ma, B.: A tutorial of recent developments in the seeding of local alignment. Journal of Bioinformatics and Computational Biology **2** (2004) 819–842
20. Pevzner, P., Waterman, M.S.: Multiple filtration and approximate pattern matching. Algorithmica **13** (1995) 135–154
21. Burkhardt, S., Kärkkäinen, J.: Better filtering with gapped q-grams. Fundamenta Informaticae **23** (2003) 1001–1008
22. Frieze, A.M., Preparata, F.P., Upfal, E.: Optimal reconstruction of a sequence from its probes. Journal of Computational Biology **6** (1999) 361–368
23. Sun, Y., Buhler, J.: Designing multiple simultaneous seeds for DNA similarity search. In: Proc. 8th Annual International Conference on Computational Molecular Biology (RECOMB). (2004) 76–84
24. Brejová, B., Brown, D., Vinař, T.: Vector seeds: An extension to spaced seeds. Journal of Computer and System Sciences **70** (2005) 364–380
25. Ross, S.M.: Stochastic Processes. Second edn. Wiley & Sons (1996)
26. Ewens, W.J., Grant, G.R.: Statistical Methods in Bioinformatics: An Introduction. Springer-Verlag (2001)
27. Kucherov, G., Noé, L., Ponty, Y.: Estimating seed sensitivity on homogeneous alignments. In: Proc. 4th IEEE Symposium on Bioinformatics and Bioengineering (BIBE). (2004) 387–394
28. Grossi, R., Vitter, J.S.: Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In: Proc. 32nd ACM Symposium on Theory of Computing (STOC). (2000) 397–406
29. Ferragina, P., Manzini, G.: Opportunistic data structures with applications. In: Proc. 41st Annual Symposium on Foundations of Computer Science (FOCS). (2000) 390–398
30. Mäkinen, V., Navarro, G.: Compressed compact suffix arrays. In: Combinatorial Pattern Matching: 15th Annual Symposium. Volume 3109 of LNCS., Springer-Verlag (2004) 421–433
31. Roberts, M., Hayes, W., Hunt, B.R., Mount, S.M., Yorke, J.A.: Reducing storage requirements for biological sequence comparison. Bioinformatics **20** (2004) 3363–3369
32. Waterman, M.S.: Introduction to Computational Biology: Maps, Sequences and Genomes. CRC Press (1995)
33. Golomb, S.W.: Run-length encodings. IEEE Transactions on Information Theory **12** (1966) 399–401
34. Hon, W.K., Sadakane, K.: Space-economical algorithms for finding maximal unique matches. In: Combinatorial Pattern Matching: 13th Annual Symposium. Volume 2373 of LNCS. (2002) 144–152

A Multiple daughter seeds in Table 2

```
--------- MD-3-13
123 45  6 78 9 0123
1110110010110101111 parent
1110110010110101110 13
1110110010010101111  7
1110010010110101111  4

--------- MD-5-13
MD-3-13+
1110110010110001111  9
1110110010110100111 10

--------- MD-11-14
12345 5 6 78  90 1 234
11110010101100110101111 parent
11110010101100110101100 13,14
11110010101000110100111  8,12
01110010101100110101111  1
10110010101100110101111  2
11010010101100110101111  3
11100010101100110101111  4
11110000101100110101111  5
11110010000100110101111  6
11110010100010011010111  7
11110010101100010101111  9
11110010101100100101111 10
11110010101100110000111 11

--------- MD-16-14
12345 5 6 78  90 1 234
11110010101100110101111 parent
11110010101100110101100 13,14
11110010101000110101110  8,14
11110010000100100101111  6,10
11110010000100110101101  6,13
11110010100000011010111  7, 8
11110010100010011010101  7,13
11010000101100110101111  3, 5
11100010101100100101111  4,10
11110010100010010010111  7,10
11110000101100110101110  5,14
11010010101100110000111  3,11
11010010000100110101111  3, 6
11010010101100010101111  3, 9
01110010101100110101111  1
10110010101100110101111  2
11110010101100110100011 12

--------- MD-11-13
123 45  6 78 9 0123
1110110010110101111 parent
1110110010110001101  9,12
1110110010100100111  8,10
1110100010110100111  5,10
```

```
0110110010110101111  1
1010110010110101111  2
1000110010110101111  3
1110010010110101111  4
1110110000110101111  6
1110110010010101111  7
1110110010110101011 11
1110110010110101110 13

--------- MD-25-14
12345 5 6 78  90 1 234
11110010101100110101111 parent
11110010101100110101100 13,14
1111001010100011010110  8,14
1111001000110010010111  6,10
1111001000110011010101  6,13
1111001010000011010111  7, 8
1111001010010011010101  7,13
1101000010110011010111  3, 5
1110001010110010010111  4,10
1111001010010010010111  7,10
1111000010110011010110  5,14
1101001010110011000111  3,11
1101001000110011010111  3, 6
1101001010110001010111  3, 9
1111001000100011010111  6, 8
1110001010100011010111  4, 8
1111001010110010010110 10,14
1111001010100011000111  8,11
1101001010100011010111  3, 8
1111001000110011010110  6,14
1111001010100011010011  8,12
1111001010110010010011 10,12
1111001010110010000111 10,11
1101001010110010010111  3,10
0111001010110011010111  1
1011001010110011010111  2

--------- MD-14-13
123 45  6 78 9 0123
1110110010110101111 parent
1110110010110001101  9,12
1110100010110100111  5,10
1110110010100100111  8,10
1110110010000101111  7, 8
1110110010110101100 12,13
1110110000110100111  6,10
0110110010110101110  1,13
1110110010110101010 11,13
1110110000110101110  6,13
1110110010100101110  8,13
1110110010110100110 10,13
1100110010110101011  3,11
1010110010110101111  2
1110010010110101111  4
```