Multi-seed Lossless Filtration
(Extended Abstract)

Gregory Kucherov[1], Laurent Noé[1], and Mikhail Roytberg[2]

[1] INRIA/LORIA, 615, rue du Jardin Botanique,
B.P. 101, 54602, Villers-lès-Nancy, France
{Gregory.Kucherov,Laurent.Noe}@loria.fr
[2] Institute of Mathematical Problems in Biology,
Pushchino, Moscow, Region, 142290, Russia
roytberg@impb.psn.ru

Abstract. We study a method of seed-based lossless filtration for approximate string matching and related applications. The method is based on a simultaneous use of several spaced seeds rather than a single seed as studied by Burkhardt and Karkkainen [1]. We present algorithms to compute several important parameters of seed families, study their combinatorial properties, and describe several techniques to construct efficient families. We also report a large-scale application of the proposed technique to the problem of oligonucleotide selection for an EST sequence database.

1 Introduction

Filtering is a widely-used technique in various string processing applications. Applied to the approximate string matching problem [2], it can be summarized by the following two-stage scheme: to find approximate occurrences (matches) of a given string in a text, one first quickly discards (filters out) those text areas where matches cannot occur, and then checks out the remaining parts of the text for actual matches. The filtering is done according to small patterns of a specified form that the searched string is assumed to share, in the exact way, with its approximate occurrences. A similar filtration scheme is used by heuristic local alignment algorithms ([3–6], to mention a few): they first identify potential similarity regions that share some patterns and then actually check whether those regions represent a significant similarity by computing a corresponding alignment.

Two types of filtering should be distinguished – *lossless* and *lossy*. A lossless filtration guarantees to detect *all* text fragments under interest, while a lossy filtration may miss some of them, but still tries to detect the majority of them. Local alignment algorithms usually use a lossy filtration. On the other hand, the lossless filtration has been studied in the context of approximate string matching problem [7, 1]. In this paper, we focus on the lossless filtration.

In the case of lossy filtration, its efficiency is measured by two parameters, usually called *selectivity* and *sensitivity*. The sensitivity measures the part of text fragments of interest that are missed by the filter (false negatives), and the

S.C. Sahinalp et al. (Eds.): CPM 2004, LNCS 3109, pp. 297–310, 2004.

selectivity indicates what part of detected fragments don't actually represent a solution (false positives). In the case of lossless filtration, only the selectivity parameter makes sense and is therefore the main characteristic of the filtration efficiency.

The choice of patterns that must be contained in the searched text fragments is a key ingredient of the filtration algorithm. *Gapped seeds* (spaced seeds, gapped q-grams) have been recently shown to significantly improve the filtration efficiency over the "traditional" technique of contiguous seeds. In the framework of lossy filtration for sequence alignment, the use of designed gapped seeds has been introduced by the PatternHunter method [4] and then used by some other algorithms (e.g. [5, 6]). In [8, 9], spaced seeds have been shown to improve indexing schemes for similarity search in sequence databases. The estimation of the sensitivity of spaced seeds (as well as of some extended seed models) has been subject of several recent studies [10–14]. In the framework of lossless filtration for approximate pattern matching, gapped seeds were studied in [1] (see also [7]) and have been also shown to increase the filtration efficiency considerably.

In this paper, we study an extension of the single-seed filtration technique [1]. The extension is based on using *seed families* rather than individual seeds. In Section 3, we present dynamic programming algorithms to compute several important parameters of seed families. In Section 4, we first study several combinatorial properties of families of seeds, and, in particular, seeds having a periodic structure. These results are used to obtain a method for constructing efficient seed families. We also outline a heuristic genetic programming algorithm for constructing seed families. Finally, in Section 5, we present several seed families we computed, and we report a large-scale experimental application of the method to the practical problem of oligonucleotide design.

2 Multiple Seed Filtering

A *seed* Q (called also *spaced seed* or *gapped q-gram*) is a list $\{p_1, p_2, \ldots, p_w\}$ of positive integers, called *matching positions*, such that $p_1 < p_2 < \ldots < p_w$. By convention, we always assume $p_1 = 0$. The *span* of a seed Q, denoted $s(Q)$, is the quantity $p_w + 1$. The number w of positions is called the *weight* of the seed and denoted $w(Q)$. Often we will use a more visual representation of seeds, adopted in [1], as words of length $s(Q)$ over the two-letter alphabet {#,-}, where # occurs at all matching positions and - at all positions in between. For example, seed $\{0, 1, 2, 4, 6, 9, 10, 11\}$ of weight 8 and span 12 is represented by word ###-#-#--###. The character - is called a *joker*. Note that, unless otherwise stated, the seed has the character # at its first and last positions.

Intuitively, a seed specifies the set of patterns that, if shared by two sequences, indicate a possible similarity between them. Two sequences are similar if the Hamming distance between them is smaller than a certain threshold. For example, sequences CACTCGT and CACACTT are similar within Hamming distance 2 and this similarity is detected by the seed ##-# at position 2. We are interested in seeds that detect *all* similarities of a given length with a given Hamming distance.

Formally, a *gapless similarity* (hereafter simply similarity) of two sequences of length m is a binary word $w \in \{0,1\}^m$ interpreted as a sequence of matches (1's) and mismatches (0's) of individual characters from the alphabet of input sequences. A seed $Q = \{p_1, p_2, \ldots, p_w\}$ *matches* a similarity w at position i, $1 \le i \le m - p_w + 1$, iff $\forall j \in [1..w],\ \ w[i + p_j] = 1$. In this case, we also say that seed Q *has an occurrence* in similarity w at position i. A seed Q is said to *detect a similarity* w if Q has at least one occurrence in w.

Given a similarity length m and a number of mismatches k, consider all similarities of length m containing k 0's and $(m - k)$ 1's. These similarities are called (m, k)-similarities. A seed Q *solves the detection problem* (m, k) (for short, the (m, k)-problem) iff for all $\binom{m}{k}$ (m, k)-similarities w, Q detects w. For example, one can check that seed `#-##--#-##` solves the $(15, 2)$-problem.

Note that the weight of the seed is directly related to the *selectivity* of the corresponding filtration procedure. A larger weight improves the selectivity, as less similarities will pass through the filter. On the other hand, a smaller weight reduces the filtration efficiency. Therefore, the goal is to solve an (m, k)-problem by a seed with the largest possible weight.

Solving (m, k)-problems by a single seed has been studied by Burkhardt and Kärkkäinen [1]. An extension we propose here is to use a *family of seeds*, instead of a single seed, to solve the (m, k)-problem. Formally, a finite family of seeds $F =< Q_l >_{l=1}^{L}$ *solves the* (m, k)*-problem* iff for all $\binom{m}{k}$ (m, k)-similarities w, there exists a seed $Q_l \in F$ that detects w.

Note that the seeds of the family are used in the complementary (or disjunctive) fashion, i.e. a similarity is detected if it is detected by *one of the seeds*. This differs from the conjunctive approach of [7] where a similarity should be detected by two seeds *simultaneously*.

The following example motivates the use of multiple seeds. In [1], it has been shown that a seed solving the $(25, 2)$-problem has the maximal weight 12. The only such seed (up to reversal) is `###-#--###-#--###-#`. However, the problem can be solved by the family composed of the following two seeds of weight 14: `#####-##---#####-##` and `#-##---#####-##---####`.

Clearly, using these two seeds increases the selectivity of the search, as only similarities having 14 or more matching characters pass the filter vs 12 matching characters in the case of single seed. On uniform Bernoulli sequences, this results in the decrease of the number of candidate similarities by the factor of $|A|^2/2$, where A is the input alphabet. This illustrates the advantage of the multiple seed approach: it allows to increase the selectivity while preserving a lossless search. The price to pay for this gain in selectivity is a double work spent on identifying the seed occurrences. In the case of large sequences, however, this is largely compensated by the decrease in the number of false positives caused by the increase of the seed weight.

3 Computing Properties of Seed Families

Burkhardt and Kärkkäinen [1] proposed a dynamic programming algorithm to compute the *optimal threshold* of a given seed – the minimal number of its oc-

currences over all possible (m,k)-similarities. In this section, we describe an extension of this algorithm for seed families and, on the other hand, describe dynamic programming algorithms for computing two other important parameters of seed families that we will use in a latter section.

Consider an (m,k)-problem and a family of seeds $F =< Q_l >_{l=1}^{L}$. We need the following notation.

- $s_{max} = max\{s(Q_l)\}_{l=1}^{L}$, $s_{min} = min\{s(Q_l)\}_{l=1}^{L}$,
- for a binary word w and a seed Q_l, $suff(Q_l, w) = 1$ if Q_l matches w at position $(|w| - s(Q_l) + 1)$ (i.e. matches a suffix of w), otherwise $suff(Q_l, w) = 0$,
- $last(w) = 1$ if the last character of w is 1, otherwise $last(w) = 0$,
- $zeros(w)$ is the number of 0's in w.

3.1 Optimal Threshold

Given an (m,k)-problem, a family of seeds $F =< Q_l >_{l=1}^{L}$ has the *optimal threshold* $T_F(m,k)$ if every (m,k)-similarity has at least $T_F(m,k)$ occurrences of seeds of F and this is the maximal number with this property. Note that overlapping occurrences of a seed as well as occurrences of different seeds at the same position are counted separately. As an example, the singleton family {`###-##`} has threshold 2 for the $(15,2)$-problem.

Clearly, F solves an (m,k)-problem if and only if $T_F(m,k) > 0$. If $T_F(m,k) > 1$, then one can strengthen the detection criterion by requiring several seed occurrences for a similarity to be detected. This shows the importance of the optimal threshold parameter.

We now describe a dynamic programming algorithm for computing the optimal threshold $T_F(m,k)$. For a binary word w, consider the quantity $T_F(m,k,w)$ defined as the minimal number of occurrences of seeds of F in all (m,k)-similarities which have the suffix w. By definition, $T_F(m,k) = T_F(m,k,\varepsilon)$. Assume that we precomputed values $\mathcal{T}_F(j,w) = T_F(s_{max}, j, w)$, for all $j \leq \max\{k, s_{max}\}$, $|w| = s_{max}$. The algorithm is based on the following recurrence relations on $T_F(i,j,w)$, for $i \geq s_{max}$.

$$T_F(i,j,w[1..n]) = \begin{cases} \mathcal{T}_F(j,w), & \text{if } i = s_{max}, \\ T_F(i-1, j-1, w[1..n-1]), & \text{if } w[n] = 0, \\ T_F(i-1, j, w[1..n-1]) + [\sum_{l=1}^{L} suff(Q_l, w)], & \text{if } n = s_{max}, \\ \min\{T_F(i,j,\mathtt{1}.w), T_F(i,j,\mathtt{0}.w)\}, & \text{if } zeros(w) < j, \\ T_F(i,j,\mathtt{1}.w), & \text{if } zeros(w) = j. \end{cases}$$

The first relation is an initial condition of the recurrence. The second one is based on the fact that if the last symbol of w is 0, then no seed can match a suffix of w (as the last position of a seed is always a matching position). The third relation reduces the size of the problem by counting the number of suffix seed occurrences. The fourth one splits the counting into two cases, by considering two possible characters occurring on the left of w. If w already contains j 0's, then only 1 can occur on the left of w, as stated by the last relation.

A dynamic programming implementation of the above recurrence allows to compute $T_F(m,k,\varepsilon)$ in a bottom-up fashion, starting from initial values $\mathcal{T}_F(j,w)$ and applying the above relations in the order in which they are written. A straightforward dynamic programming implementation requires $O(m \cdot k \cdot 2^{(s_{max}+1)})$ time and space. However, the space complexity can be immediately improved: if values of i are processed successively, then only $O(k \cdot 2^{(s_{max}+1)})$ space is needed. Furthermore, for each i and j, it is not necessary to consider all $2^{(s_{max}+1)}$ different strings w, but only those which contain up to j 0's. The number of those w is $g(j,s_{max}) = \sum_{i=0}^{j} \binom{s_{max}}{i}$. For each i, j ranges from 0 to k. Therefore, for each i, we need to store $f(k,s_{max}) = \sum_{j=0}^{k} g(j,s_{max}) = \sum_{i=0}^{k} \binom{s_{max}}{i} \cdot (k-i+1)$ values. This yields the same space complexity as for computing the optimal threshold for one seed [1].

The quantity $\sum_{l=1}^{L} suff(Q_l,w)$ can be precomputed for all considered words w in time $O(L \cdot g(k,s_{max}))$ and space $O(g(k,s_{max}))$, under the assumption that checking an individual match is done in constant time. This leads to the overall time complexity $O(m \cdot f(k,s_{max}) + L \cdot g(k,s_{max}))$ with the leading suff $m \cdot f(k,s_{max})$ (as L is usually small compared to m and $g(k,s_{max})$ is smaller than $f(k,s_{max})$.

3.2 Number of Undetected Similarities

We now describe a dynamic programming algorithm that computes another characteristic of a seed family, that will be used later in Section 4.4. Consider an (m,k)-problem. Given a seed family $F =< Q_l >_{l=1}^{L}$, we are interested in the number $U_F(m,k)$ of (m,k)-similarities that are not detected by F. For a binary word w, define $U_F(m,k,w)$ to be the number of undetected (m,k)-similarities that have the suffix w.

Similar to [10], let $X(F)$ be the set of binary words w such that *(i)* $|w| \leq s_{max}$, *(ii)* for any $Q_l \in F$, $suff(Q_l, 1^{s_{max}-|w|}w) = 0$, and *(iii)* no proper suffix of w verifies *(ii)*. Note that word 0 belongs to $X(F)$, as the last position of every seed is a matching position.

The following recurrence relations allow to compute $U_F(i,j,w)$ for $i \leq m$, $j \leq k$, and $|w| \leq s_{max}$.

$$U_F(i,j,w[1..n]) = \begin{cases} \binom{i-|w|}{j-zeros(w)}, & \text{if } i < s_{min}, \\ 0, & \text{if } \exists l \in [1..L],\ suff(Q_l,w) = 1, \\ U_F(i-1, j-last(w), w[1..n-1]), & \text{if } w \in X(F), \\ U_F(i,j,\mathtt{1}.w) + U(i,j,\mathtt{0}.w), & \text{if } zeros(w) < j, \\ U_F(i,j,\mathtt{1}.w), & \text{if } zeros(w) = j. \end{cases}$$

The first condition says that if $i < s_{min}$, then no word of length i will be detected, hence the binomial formula. The second condition is straightforward. The third relation follows from the definition of $X(F)$ and allows to reduce the size of the problem. The last two conditions are similar to those from the previous section.

The set $X(F)$ can be precomputed in time $O(L \cdot g(k,s_{max}))$ and the worst-case time complexity of the whole algorithm remains $O(m \cdot f(k,s_{max}) + L \cdot g(k,s_{max}))$.

3.3 Contribution of a Seed

Using a similar dynamic technique, one can compute, for a given seed of the family, the number of (m,k)-similarities that are detected only by this seed and not by the others. Together with the number of undetected similarities, this parameter will be used later in Section 4.4.

Given an (m,k)-problem and a family $F =< Q_l >_{l=1}^{L}$, we define $S_F(m,k,l)$ to be the number of (m,k)-similarities detected by the seed Q_l exclusively (through one or several occurrences), and $S_F(m,k,w,l)$ to be the number of those similarities ending with the suffix w. A dynamic programming algorithm similar to the one described in the previous sections can be applied to compute $S_F(m,k,l)$. Below we give only the main recurrence relations for $S_F(m,k,w,l)$ and leave out initial conditions.

$$S_F(i,j,w[1..n],l) = \begin{cases} \sum_{x\in\{0,1\}} S_F(i-1,j-1+x,x.w[1..n-1],l) & \text{if } \mathit{suff}(Q_l,w)=1 \text{ and} \\ \quad + \ U_F(i-1,j-1+x,x.w[1..n-1]), & \forall l' \neq l,\ \mathit{suff}(Q_{l'},w)=0, \\ \sum_{x\in\{0,1\}} S_F(i-1,j-1+x,x.w[1..n-1],l) & \text{if } \forall l',\ \mathit{suff}(Q_{l'},w)=0. \end{cases}$$

The first relation allows to reduce the problem when Q_l matches a suffix of w, but not the other seeds of the family. The second one applies if no seed matches a suffix of w. The complexity of computing $S_F(m,k,l)$ for a given l is the same as the complexity of dynamic programming algorithms from the previous sections.

4 Seed Design

In the previous Section we showed how to compute various useful characteristics of a given family of seeds. A much more difficult task is to find an efficient seed family that solves a given (m,k)-problem. Note that there exists a trivial solution where the family consists of all $\binom{m}{k}$ position combinations, but this is in general unacceptable in practice because of a huge number of seeds. Our goal is to find families of reasonable size (typically, with the number of seeds smaller than ten), with a good filtration efficiency.

In this section, we present several results that contribute to this goal. In Section 4.1, we start with the case of single seed with a fixed number of jokers and show, in particular, that for one joker, there exists one best seed in a sense that will be defined. We then show in Section 4.2 that a solution for a larger problem can be obtained from a smaller one by a regular expansion operation. In Section 4.3, we focus on seeds that have a periodic structure and show how those seeds can be constructed by iterating some smaller seeds. We then show a way to build efficient families of periodic seeds. Finally, in Section 4.4, we briefly describe a heuristic approach to constructing efficient seed families that we used in the experimental part of this work presented in Section 5.

4.1 Single Seeds with a Fixed Number of Jokers

Assume that we fixed a class of seeds under interest (e.g. seeds of a given minimal weight). One possible way to define the seed design problem is to fix the similarity

length m and find a seed that solves the (m, k)-problem with the largest possible value of k. A complementary definition is to fix k and minimize m provided that the (m, k)-problem is still solved. In this section, we adopt the second definition and present an optimal solution for one particular case.

For a seed Q and a number of mismatches k, define the *k-critical value* for Q as the minimal value m such, that Q solves the (m, k)-problem. For a class of seeds $\mathcal{C}$ and a value k, a seed is k-optimal in $\mathcal{C}$ if Q has the minimal *k-critical value* among all seeds of $\mathcal{C}$.

One interesting class of seeds $\mathcal{C}$ is obtained by putting an upper bound on the possible number of jokers in the seed, i.e. on the number $(s(Q) - w(Q))$. We have found a general solution of the seed design problem for the class $\mathcal{C}_1(n)$ consisting of seeds of weight n with only one joker.

Theorem 1. *Let n be an integer and $r = \lfloor n/3 \rfloor$. For every $k \geq 2$, seed $Q(n) =$ `#`$^{n-r}$`-#`^{r} is k-optimal among the seeds of $\mathcal{C}_1(n)$.*

To illustrate Theorem 1, seed `####-##` is optimal among all seeds of weight 6 with one joker. This means that this seed solves the $(m, 2)$-problem for all $m \geq 16$ and this is the smallest possible bound over all seeds of this class. Similarly, this seed solves the $(m, 3)$-problem for all $m \geq 20$, which is the best possible bound, etc.

4.2 Regular Expansion and Contraction of Seeds

We now show that seeds solving larger problems can be obtained from seeds solving smaller problems, and vice versa, using a regular expansion and regular contraction operations.

Given a seed Q , its *i-regular expansion* $i \otimes Q$ is obtained by multiplying each matching position by i. This is equivalent to inserting $i - 1$ jokers between every two successive positions along the seed. For example, if $Q = \{0, 2, 3, 5\}$ (or `#-##-#`), then the 2-regular expansion of Q is $2 \otimes Q = \{0, 4, 6, 10\}$ (or `#---#-#---#`). Given a family F, its i-regular expansion $i \otimes F$ is the family obtained by applying the i-regular expansion on each seed of F.

Lemma 1. *If a family F solves the (m, k)-problem, then the $(im, (i+1)k-1)$-problem is solved both by family F and by its i-regular expansion $F_i = i \otimes F$.*

Proof. Consider an $(im, (i+1)k-1)$-similarity w. By the pigeon hole principle, it contains at least one substring of length m with k mismatches or less, and therefore F solves the $(im, (i+1)k-1)$-problem. On the other hand, consider i disjoint subsequences of w each one consisting of m positions equal modulo i. Again, by the pigeon hole principle, at least one of them contains k mismatches or less, and therefore the $(im, (i+1)k-1)$-problem is solved by $i \otimes F$.

The following lemma is the inverse of Lemma 1, it states that if seeds solving a bigger problem have a regular structure, then a solution for a smaller problem can be obtained by the regular contraction operation, inverse to the regular expansion.

Lemma 2. *If a family $F_i = i \otimes F$ solves the (im, k)-problem, then F solves both the (im, k)-problem and the $(m, \lfloor k/i \rfloor)$-problem.*

Proof. Similar to Lemma 1.

Example 1. To illustrate the two lemmas above, we give the following example pointed out in [1]. The following two seeds are the only seeds of weight 12 that solve the $(50, 5)$-problem: `#-#-#---#-----#-#-#---#-----#-#-#---#` and `###-#--###-#--###-#`. The first one is the 2-regular expansion of the second. The second one is the only seed of weight 12 that solves the $(25, 2)$-problem.

The regular expansion allows, in some cases, to obtain an efficient solution for a larger problem by reducing it to a smaller problem for which an optimal or a near-optimal solution is known.

4.3 Periodic Seeds

In this section, we study seeds with a periodic structure that can be obtained by iterating a smaller seed. Such seeds often turn out to be among maximally weighted seeds solving a given (m, k)-problem. Interestingly, this contrasts with the lossy framework where optimal seeds usually have a "random" irregular structure.

Consider two seeds Q_1,Q_2 represented as words over {`#`,`-`}. We denote $[Q_1,Q_2]^i$ the seed defined as $(Q_1Q_2)^iQ_1$. For example, $[$`###-#`, `--`$]^2$ = `###-#--###-#--###-#`.

We also need a modification of the (m, k)-problem, where (m, k)-similarities are considered modulo a cyclic permutation. We say that a seed family F solves a *cyclic (m, k)-problem*, if for every (m, k)-similarity w, F detects one of cyclic permutations of w. Trivially, if F solves an (m, k)-problem, it also solves the cyclic (m, k)-problem. To distinguish from a cyclic problem, we call sometimes an (m, k)-problem a *linear* problem.

We first restrict ourselves to the single-seed case. The following lemma demonstrates that iterating smaller seeds solving a cyclic problem allows to obtain a solution for bigger problems, for the same number of mismatches.

Lemma 3. *If a seed Q solves a* cyclic *(m, k)-problem, then for every $i \geq 0$, the seed $Q_i = [Q, -^{(m-s(Q))}]^i$ solves the linear $(m \cdot (i+1) + s(Q) - 1, k)$-problem. If $i \neq 0$, the inverse holds too.*

Example 2. Observe that the seed `###-#` solves the cyclic $(7, 2)$-problem. From Lemma 3, this implies that for every $i \geq 0$, the $(11 + 7i, 2)$-problem is solved by the seed $[$`###-#`, `--`$]^i$ of span $5 + 7i$. Moreover, for $i = 1, 2, 3$, this seed is optimal (maximally weighted) over all seeds solving the problem.

By a similar argument based on Lemma 3, the periodic seed $[$`#####-##`, `---`$]^i$ solves the $(18 + 11i, 2)$-problem. Note that its weight grows as $\frac{7}{11}m$ compared to $\frac{4}{7}m$ for the seed from the previous paragraph. However, this is not an asymptotically optimal bound, as we will see later.

The $(18 + 11i, 3)$-problem is solved by the seed $(\texttt{###-\#--\#}, \texttt{---})^i$, as seed `###-#--#` solves the cyclic $(11, 3)$-problem. For $i = 1, 2$, the former is a maximally weighted seed among all solving the $(18 + 11i, 3)$-problem.

One question raised by these examples is whether there exists a general periodic seed which is asymptotically optimal, i.e. has a maximal asymptotic weight. The following theorem establishes a tight asymptotic bound on the weight of an optimal seed, for a fixed number of mismatches. It gives a negative answer to this question, as it shows that the maximal weight grows faster than any linear fraction of the problem size.

Theorem 2. *Consider a fixed k. Let $w(m)$ be the maximal weight of a seed solving the cyclic (m, k)-problem. Then $(m - w(m)) = \Theta(m^{\frac{k-1}{k}})$.*

The following simple lemma is also useful for constructing efficient seeds.

Lemma 4. *Assume that a family F solves an (m, k)-problem. Let F' be the family obtained from F by cutting out l characters from the left and r characters from the right of each seed of F. Then F' solves the $(m - r - l, k)$-problem.*

Example 3. The $(9 + 7i, 2)$-problem is solved by the seed $[\texttt{###}, \texttt{-\#--}]^i$ which is optimal for $i = 1, 2, 3$. Using Lemma 4, this seed can be immediately obtained from the seed $[\texttt{###-\#}, \texttt{--}]^i$ from Example 2, solving the $(11 + 7i, 2)$-problem.

We now apply the above results for the single seed case to the case of multiple seeds.

For a seed Q considered as a word over {#,-}, we denote by $Q_{[i]}$ its cyclic shift to the left by i characters. For example, if $Q =$ `####-#-##--`, then $Q_{[5]} =$ `#-##--####-`. The following lemma gives a way to construct seed families solving bigger problems from an individual seed solving a smaller cyclic problem.

Lemma 5. *Assume that a seed Q solves a cyclic (m, k) problem and assume that $s(Q) = m$ (otherwise we pad Q on the right with $(m - s(Q))$ jokers). Fix some $i > 1$. For some $L > 0$, consider a list of L integers $0 \le j_1 < \cdots < j_L < m$, and define a family of seeds $F =< \|(Q_{[j_l]})^i\| >_{l=1}^{L}$, where $\|(Q_{[j_l]})^i\|$ stands for the seed obtained from $(Q_{[j_l]})^i$ by deleting the joker characters at the left and right edges. Define $\delta(l) = ((j_{l-1} - j_l) \mod m)$ (or, alternatively, $\delta(l) = ((j_l - j_{l-1}) \mod m)$) for all l, $1 \le l \le L$. Let $m' = \max\{s(\|(Q_{[j_l]})^i\|) + \delta(l)\}_{l=1}^{L} - 1$. Then F solves the (m', k)-problem.*

We illustrate Lemma 5 with two examples that follow.

Example 4. Let $m = 11$, $k = 2$. Consider the seed $Q =$ `####-#-##--` solving the cyclic $(11, 2)$-problem. Choose $i = 2$, $L = 2$, $j_1 = 0$, $j_2 = 5$. This gives two seeds $Q_1 = \|(Q_{[0]})^2\| =$ `####-#-##--####-#-##` and $Q_2 = \|(Q_{[5]})^2\| =$ `#-##--####-#-##--####` of span 20 and 21 respectively, $\delta(1) = 6$ and $\delta(2) = 5$. $\max\{20 + 6, 21 + 5\} - 1 = 25$. Therefore, family $F = \{Q_1, Q_2\}$ solves the $(25, 2)$-problem.

Example 5. Let $m = 11$, $k = 3$. The seed $Q =$ `###-#--#---` solving the cyclic $(11, 3)$-problem. Choose $i = 2$, $L = 2$, $j_1 = 0$, $j_2 = 4$. The two seeds are $Q_1 = \|(Q_{[0]})^2\| =$ `###-#--#---###-#--#` (span 19) and $Q_2 = \|(Q_{[4]})^2\| =$ `#--#---###-#--#---###` (span 21), with $\delta(1) = 7$ and $\delta(2) = 4$. $\max\{19 + 7, 21 + 4\} - 1 = 25$. Therefore, family $F = \{Q_1, Q_2\}$ solves the $(25, 3)$-problem.

4.4 Heuristic Seed Design

Results of Sections 4.1-4.3 allow to construct efficient seed families in certain cases, but still do not allow to perform a systematic seed design. In this section, we briefly outline a heuristic genetic programming algorithm for designing seed families. The algorithm was used in the experimental part of this work, that we present in the next section. Note that this algorithm uses dynamic programming algorithms of Section 3.

The algorithm tries to iteratively improve the characteristics of a *population* of seed families. A sample of family candidates are processed, then some of them are *mutated* and *crossed over* according to the set of (m, k)-similarities they do not detect.

The first step of each iteration is based on screening current families against sets of *difficult similarities*, which are similarities that have been detected by fewer families. These sets are permanently reordered and updated according to the number of families that don't detect those similarities.

For those families that pass through the screening filter, the number of undetected similarities is computed by the dynamic programming algorithm of Section 3.2. The family is kept if it produces a smaller number than the families currently known. To detect seeds to be improved inside a family, we compute the contribution of each seed by the dynamic programming algorithm of Section 3.3. The seeds with the least contribution are then modified.

The entire heuristic procedure does not guarantee finding optimal seeds families but often allows to compute efficient or even optimal solutions in a reasonable time. For example, in ten runs of the algorithm we found 3 of the 6 possible families of two seeds of weight 14 solving the $(25, 2)$-problem. The whole computation took less than 1 hour, compared to a week of computation needed to exhaustively test all seed pairs.

5 Experiments

We describe two groups of experiments we have made. The first one concerns the design of efficient seed families, and the second one applies a multi-seed lossless filtration to the identification of unique oligos in a large set of EST sequences.

Seed Design Experiments

We considered several (m, k)-problems. For each problem, and for a fixed number of seeds in the family, we computed families solving the problem and realizing

as large seed weight as possible (under a natural assumption that all seeds in a family have the same weight). We also kept track of the ways (periodic seeds, genetic programming heuristics, exhaustive search) in which those families can be computed.

Tables 1 and 2 summarize some results obtained for the $(25, 2)$-problem and the $(25, 3)$-problem respectively. Families of periodic seeds (that can be found using Lemma 5) are marked with ^{p}, those that are found using a genetic algorithm are marked with ^{g}, and those which are obtained by an exhaustive search are marked with ^{e}. Only in this latter case, the families are guaranteed to be optimal. Families of periodic seeds are shifted according to their construction (see Lemma 5).

Moreover, to compare the selectivity of different families solving a given (m, k)-problem, we estimated the probability δ for at least one of the seeds of the family to match at a given position of a uniform Bernoulli four-letter sequence.

Note that the simple fact of passing from a single seed to a two-seed family results in a considerable gain in efficiency: in both examples shown in the tables there a change of about one order magnitude in the selectivity estimator δ.

Table 1. Seed families for $(25, 2)$-problem

size	weight	family seeds	δ
1	$12^{e,p,g}$	`###-#--###-#--###-#`	$5.96 \cdot 10^{-8}$
2	$14^{e,p,g}$	`####-#-##--####-#-##` `#-##--####-#-##--####`	$7.47 \cdot 10^{-9}$

3	15^{p}	`#--##-#-######--##-#-##`
		`#-######--##-#-#####`
		`####--##-#-######--##`
			$2.80 \cdot 10^{-9}$

4	16^{p}	`###-##-#-###--#######`
		`##-#-###--#######-##-#`
		`###--#######-##-#-###`
		`#######-##-#-###--###`
			$9.42 \cdot 10^{-10}$

6	17^{p}	`##-#-##--#######-####-#`
		`#-##--#######-####-#-##`
		`#######-####-#-##--###`
		`###-####-#-##--#######`
		`####-#-##--#######-###`
		`##--#######-####-#-##--#`
			$3.51 \cdot 10^{-10}$

Oligo Design Using Multi-seed Filtering

An important practical application of lossless filtration is the design of reliable oligonucleotides for DNA micro-array experiments. Oligonucleotides (oligos) are small DNA sequences of fixed size (usually ranging from 10 to 50) designed to hybridize only with a specific region of the genome sequence. In micro-array experiments, oligos are expected to match ESTs that stem from a given gene and not to match those of other genes. The problem of oligo design can then be formulated as the search for strings of a fixed length that occur in a given sequence but do not occur, within a specified distance, in other sequences of a

Table 2. Seed families for $(25, 3)$-problem

size	weight	family seeds	δ
1	$8^{\,e,p,g}$	`###-#-----###-#`	$1.53 \cdot 10^{-5}$
2	10^{p}	`####-#-##--#---##`	$1.91 \cdot 10^{-6}$
		`##--#---####-#-##`	
3	11^{p}	`#---####-#-##--#---##`	$7.16 \cdot 10^{-7}$
		`###-#-##--#---####`	
		`##--#---####-#-##--#`	
4	12^{p}	`#---####-#-##--#---###`	$2.39 \cdot 10^{-7}$
		`###-#-##--#---####-#`	
		`#-##--#---####-#-##--#`	
		`##--#---####-#-##--#---#`	

given (possibly very large) sample. Different approaches to oligo design apply different distance measures and different algorithmic techniques [15–18]. The experiments we briefly present here demonstrate that the multi-seed filtering provides an efficient solution to the oligo design problem.

family size	weight	δ
1	7^{e}	$6.10 \cdot 10^{-5}$
2	8^{e}	$3.05 \cdot 10^{-5}$
3	9^{e}	$1.14 \cdot 10^{-5}$
4	10^{g}	$3.81 \cdot 10^{-6}$
6	11^{g}	$1.43 \cdot 10^{-6}$
10	12^{g}	$5.97 \cdot 10^{-7}$

```
{ ####---#---------#---#--#### ,
  ###--#--##--------#-#### ,
  ####----#--#--##-### ,
  ###-#-#---##--#### ,
  ###-##-##--#-#-## ,
  ####-##-#-#### }
```

Fig. 1. Computed seed families for the considered $(32, 5)$-problem and the chosen family (6 seeds of weight 11)

We adopt the formalization of the oligo design problem as the problem of identifying in a given sequence (or a sequence database) all substrings of length m that have no occurrences elsewhere in the sequence within the Hamming distance k. The parameters m and k were set to 32 and 5 respectively. For the $(32, 5)$-problem, different seed families were designed and their selectivity was estimated. Those are summarized in the table in Figure 1, using the same conventions as in Tables 1 and 2 above. The family composed of 6 seeds of weight 11 was selected for the filtration experiment (shown in Figure 1).

The filtering has been applied to a database of rice EST sequences composed of 100015 sequences for a total length of 42.845.242 Mb[1]. Substrings matching other substrings with 5 substitution errors or less were computed. The computation took slightly more than one hour on a Pentium 4 3GHz computer. Before applying the filtering using the family for the $(32, 5)$-problem, we made a rough pre-filtering using one spaced seed of weight 16 to detect, with a high selectivity, almost identical regions. As a result of the whole computation, 87% percent of the database were removed by the filtering procedure, leaving the remaining part as oligo candidates.

[1] source : `http://bioserver.myongji.ac.kr/ricemac.html`, The Korea Rice Genome Database

6 Conclusion

In this paper, we studied a lossless filtration method based on multi-seed families and demonstrated that it represents an improvement compared to the single-seed approach considered in [1]. We showed how some important characteristics of seed families can be computed using the dynamic programming. We presented several combinatorial results that allow one to construct efficient families composed of seeds with a periodic structure. Finally, we described experimental results providing evidence of the applicability and efficiency of the whole method.

The results of Sections 4.1-4.3 establish several combinatorial properties of seed families, but many more of them remain to be elucidated. We conjecture that the structure of optimal or near-optimal seed families can be studied using the combinatorial design theory, but this relation remains to be clearly established. Another direction is to consider different distance measures, especially the Levenstein distance, or at least to allow some restricted insertion/deletion errors. The method proposed in [19] does not seem to be easily generalized to multi-seed families, and a further work is required to improve lossless filtering in this case.

After this work was completed, it has been brought to our attention that in the context of lossy filtering for local alignement algorithms, the use of optimized multi-seed families has been recently proposed in PatternHunter II software [20], and the design of those families has been also studied in paper [21].

Acknowledgements

G. Kucherov and L. Noé have been supported by the French *Action Spécifique "Algorithmes et Séquences"* of CNRS. A part of this work has been done during a stay of M. Roytberg at LORIA, Nancy, supported by INRIA. M. Roytberg has been supported by the Russian Foundation for Basic Research (project nos. 03-04-49469, 02-07-90412) and by grants from the RF Ministry for Industry, Science, and Technology (20/2002, 5/2003) and NWO.

References

1. Burkhardt, S., Kärkkäinen, J.: Better filtering with gapped q-grams. Fundamenta Informaticae **56** (2003) 51–70 Preliminary version in Combinatorial Pattern Matching 2001.
2. Navarro, G., Raffinot, M.: Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences. Cambridge University Press (2002) ISBN 0-521-81307-7. 280 pages.
3. Altschul, S., Madden, T., Schäffer, A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Research **25** (1997) 3389–3402
4. Ma, B., Tromp, J., Li, M.: PatternHunter: Faster and more sensitive homology search. Bioinformatics **18** (2002) 440–445

5. Schwartz, S., Kent, J., Smit, A., Zhang, Z., Baertsch, R., Hardison, R., Haussler, D., Miller, W.: Human–mouse alignments with BLASTZ. Genome Research **13** (2003) 103–107
6. Noe, L., Kucherov, G.: YASS: Similarity search in DNA sequences. Research Report RR-4852, INRIA (2003) `http://www.inria.fr/rrrt/rr-4852.html`.
7. Pevzner, P., Waterman, M.: Multiple filtration and approximate pattern matching. Algorithmica **13** (1995) 135–154
8. Califano, A., Rigoutsos, I.: Flash: A fast look-up algorithm for string homology. In: Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology. (1993) 56–64
9. Buhler, J.: Provably sensitive indexing strategies for biosequence similarity search. In: Proceedings of the 6th Annual International Conference on Computational Molecular Biology (RECOMB02), Washington, DC (USA), ACM Press (2002) 90–99
10. Keich, U., Li, M., Ma, B., Tromp, J.: On spaced seeds for similarity search. Discrete Applied Mathematics (2004) to appear.
11. Buhler, J., Keich, U., Sun, Y.: Designing seeds for similarity search in genomic DNA. In: Proceedings of the 7th Annual International Conference on Computational Molecular Biology (RECOMB03), Berlin (Germany), ACM Press (2003) 67–75
12. Brejova, B., Brown, D., Vinar, T.: Vector seeds: an extension to spaced seeds allows substantial improvements in sensitivity and specificity. In Benson, G., Page, R., eds.: Proceedings of the 3rd International Workshop in Algorithms in Bioinformatics (WABI), Budapest (Hungary). Volume 2812 of Lecture Notes in Computer Science., Springer (2003) 39–54
13. Kucherov, G., Noe, L., Ponty, Y.: Estimating seed sensitivity on homogeneous alignments. In: Proceedings of the IEEE 4th Symposium on Bioinformatics and Bioengineering (BIBE2004), May 19-21, 2004, Taichung (Taiwan), IEEE Computer Society Press (2004) to appear.
14. Choi, K., Zhang, L.: Sensitivity analysis and efficient method for identifying optimal spaced seeds. Journal of Computer and System Sciences (2003) to appear.
15. Li, F., Stormo, G.: Selection of optimal DNA oligos for gene expression arrays. Bioinformatics **17** (2001) 1067–1076
16. Kaderali, L., Schliep, A.: Selecting signature oligonucleotides to identify organisms using DNA arrays. Bioinformatics **18** (2002) 1340–1349
17. Rahmann, S.: Fast large scale oligonucleotide selection using the longest common factor approach. Journal of Bioinformatics and Computational Biology **1** (2003) 343–361
18. Zheng, J., Close, T., Jiang, T., Lonardi, S.: Efficient selection of unique and popular oligos for large est databases. In: LNCS 2676, Proceedings of Symposium on Combinatorial Pattern Matching (CPM'03), Springer (2003) 273–283
19. Burkhardt, S., Karkkainen, J.: One-gapped q-gram filters for Levenshtein Distance. In: Proceedings of the 13th Symposium on Combinatorial Pattern Matching (CPM'02). Volume 2373., Springer (2002) 225–234
20. Li, M., Ma, B., Kisman, D., Tromp, J.: PatternHunter II: Highly sensitive and fast homology search. Journal of Bioinformatics and Computational Biology (2004) Earlier version in GIW 2003 (International Conference on Genome Informatics).
21. Sun, Y., Buhler, J.: Designing multiple simultaneous seeds for DNA similarity search. In: Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2004), ACM Press (2004)