One-Gapped q-Gram Filters for Levenshtein Distance

Stefan Burkhardt[1]⋆ Juha Kärkkäinen[2]⋆⋆

[1] Center for Bioinformatics, Saarland University
Postfach 151150, 66041 Saarbrücken, Germany; stburk@mpi-sb.mpg.de
[2] Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany; juha@mpi-sb.mpg.de

Abstract. We have recently shown that q-gram filters based on gapped q-grams instead of the usual contiguous q-grams can provide orders of magnitude faster and/or more efficient filtering for the Hamming distance. In this paper, we extend the results for the Levenshtein distance, which is more problematic for gapped q-grams because an insertion or deletion in the gap affects a q-gram while a replacement does not. To keep these effects under control, we concentrate on gapped q-grams with just one gap. We demostrate with experiments that the resulting filters provide a significant improvement over the contiguous q-gram filters. We also develop new techniques for dealing with complex q-gram filters.

1 Introduction

Given a *pattern* string P and a *text* string T, the *approximate string matching problem* is to find all substrings of the text (*matches*) that are within a *distance* k of the pattern P. The most commonly used distance measure is the *Levenshtein distance*, the minimum number of single character insertions, deletions and replacements needed to change one string into the other. A simpler variant is the *Hamming distance*, that does not allow insertions and deletions, i.e., it is the number of nonmatching characters for strings of the same length. The *indexed* version of the problem allows preprocessing the text to build an index while the *online* version does not. Good surveys are given in [10, 11].

Filtering is a way to speed up approximate string matching, particularly in the indexed case but also in the online case. A *filter* is an algorithm that quickly discards large parts of the text based on some *filter criterium*, leaving the remaining part to be checked with a proper (online) approximate string matching algorithm. A filter is *lossless* if it never discards an actual match; we consider only lossless filters. The ability of a filter to reduce the text area is called its *(filtration) efficiency*.

Many filters are based on *q-grams*, substrings of length q. The q-gram similarity (defined as a distance in [13]) of two strings is the number of q-grams shared by the strings. The q-gram filter is based on the *q-gram lemma*:

Lemma 1 ([6]). *Let P and S be strings with (Levenshtein or Hamming) distance k. Then the q-gram similarity of P and S is at least $t = |P| - q + 1 - kq$.*

⋆ Supported by the DFG 'Initiative Bioinformatik' grant BIZ 4/1-1.

⋆⋆ Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

The value t in the lemma is called the *threshold* and gives the minimum number of q-grams that an approximate match must share with the pattern, which is used as the filter criterium. There are actually many possible ways to count the number of shared q-grams offering different tradeoffs between speed and filtration efficiency (see, e.g., [6, 13, 5, 1]). However, in all cases the value of the threshold is the one given by the lemma.

A generalization of the q-gram filter uses *gapped* q-grams, subsets of q characters of a fixed non-contiguous *shape*. For example, the 3-grams of shape `##-#` in the string `ACAGCT` are `ACG`, `CAC` and `AGT`. In [2], we showed that the use of gapped q-grams can significantly improve the filtration efficiency and/or speed of the q-gram filter *for the Hamming distance*. The result cannot be trivially extended to the Levenshtein distance due to the effect of insertions and deletions on the gaps. In the above example, a replacement of `G` would leave the 3-gram `CAC` unaffected but the deletion of `G` or an insertion of a character before `G` would change all 3-grams.

In this paper, we study gapped q-gram filters for the Levenshtein distance based on an idea already suggested in [2]. The idea is to use multiple shapes that differ only in the length of the gap(s), for example, the shapes `##--#`, `##-#` and `###`. We restrict our consideration to q-grams with only one gap, which minimizes the number of different shapes needed. Our experimental results show that with the right choice of the shape a significant improvement over conventional q-gram filters can be achieved for Levenshtein distance, too.

Even with the restriction to one-gapped q-grams, a major obstacle in implementing the filters is determining the value of the threshold. As already noted in [2], for gapped q-gram filters, there is no simple threshold formula like the one given by Lemma 1. Indeed, even defining the threshold precisely is far from trivial. The definition we give here not only solves the problem for the one-gapped q-gram filters but provides a framework for solving it for other even more complicated filters.

Gapped q-grams have also been used in [3, 12, 9]. In [3, 12], the motivation is to increase the filtration efficiency by considering multiple shapes. Pevzner and Waterman [12] use q-grams containing every $(k+1)$st character together with contiguous q-grams for the Hamming distance. Califano and Rigoutsos [3] describe a *lossy* filter for the Levenshtein distance that uses as many as 40 different random shapes. Their approach is effective for high k but they need a huge index (18GB for a 100 million nucleotide DNA database). The Grampse system of Lehtinen et al. [9] uses a shape containing every hth character for some h (similar to [12]) for exact matching. Their motivation of using gapped q-grams is to reduce dependencies between the characters of a q-gram.

2 The filter algorithm

The basic idea behind the gapped q-gram filters for the Levenshtein distance was already suggested in [2]. The filters use a basic shape with only one gap and two other shapes formed from the basic shape by increasing and decreasing the length of the gap by one. For example, with the basic shape `##-#` we would also use the shapes `##--#` and `###`. The filter compares the q-grams of all three shapes in the pattern to the q-grams of the basic shape in the text.[1] Then matching q-grams are found even if there is an insertion or deletion in the gap.

Otherwise the filter algorithm is similar to the one described in [5] for the contiguous q-grams. Let us define a *hit* as a pair (i, j) such that a q-gram (of any of the three shapes) starting at position i in P matches a q-gram (of the basic shape) starting at position j in T. The *diagonal* of a hit (i, j) is $j - i$. The diagonal represents the approximate starting position of the corresponding substring of T, or more accurately, the diagonal of the dynamic programming matrix corresponding to the beginning of the q-gram. The following result was shown for contiguous q-grams in [5] and it trivially extends to gapped q-grams.

Lemma 2. *If a substring S of T is within distance k of P, then S and P share at least t q-grams such that the diagonals of the corresponding hits differ by at most k.*

For contiguous q-grams, the threshold t in the lemma is the one in Lemma 1. For gapped q-grams, the threshold is defined in the next section.

Based on the lemma, it is enough to find all sets of $k + 1$ contiguous diagonals that contain at least t hits. We call such a set of diagonals a *match*.[2] The matches can be computed by finding all the hits with a q-gram index of the text, sorting them by diagonal, and scanning the sorted list. If radix sort is used, the time requirement is $O(q|P| + h)$, where h is the number of hits.

3 The threshold

For contiguous q-grams, there is a simple formula (Lemma 1) for computing the threshold, but this is not the case for gapped q-grams. In fact, even defining the threshold precisely is nontrivial. In this section, we give a formal definition of the threshold. The algorithm we used for computing the threshold is described in [7].

Following [4], we define an *edit transcript* as a string over the alphabet M(atch), R(eplace), I(nsert) and D(elete), describing a sequential character-by-character transformation of one string to another. For two strings P and S, let $\mathcal{T}(P, S)$ denote the set of all transcripts transforming P to S. For example, $\mathcal{T}(\texttt{actg}, \texttt{acct})$ contains `MMRR`, `MMIMD`, `MIMMD`, `IRMMD`, `IDIMDID`, etc.. For a transcript $\tau \in \mathcal{T}(P, S)$, the source length $slen(\tau)$ of τ is the length of P, i.e., the number of non-insertions in τ. The Levenshtein cost $c_L(\tau)$ is the number of non-matches. The Hamming cost $c_H(\tau)$ is infinite if τ contains insertions or deletions and the same as Levenshtein cost otherwise. The Levenshtein distance and Hamming distance of P and S are $d_L(P, S) = \min_{\tau \in \mathcal{T}(P,S)} c_L(\tau)$ and $d_H(P, S) = \min_{\tau \in \mathcal{T}(P,S)} c_H(\tau)$, respectively.

[1] An alternative would be to use two shapes, for example `##-#` and `##--#`, for both strings. The main advantage of the asymmetric approach is that it requires a text index for only one shape.

[2] Matches that overlap are merged into one larger match by the algorithm and are counted as one match in the experiments.

Here we defined distance measures for strings using cost functions for edit transcripts. Similarly, we define the q-gram similarity measures for strings using *profit functions* for edit transcripts. Then we can define the threshold using edit transcripts as follows.

Definition 1. *The* threshold *for a cost function c and a profit function p of edit transcripts is*

$$t_p^c(m,k) = \min_{\tau}\{p(\tau) \mid slen(\tau) = m, c(\tau) \leq k\}.$$

The following lemma gives the filter criterium.

Lemma 3. *Let c be a cost function and p a profit function for edit transcripts. Define a distance d of two strings P and S as $d(P,S) = \min_{\tau \in \mathcal{T}(P,S)} c(\tau)$ and a similarity s as $s(P,S) = \max_{\tau \in \mathcal{T}(P,S)} p(\tau)$. Now, if $d(P,S) \leq k$, then $s(P,S) \geq t_p^c(|P|,k)$.*

The lemma holds for any choice of cost c and profit p. The cost functions of interest to us were defined above. The profit functions that define the q-gram based similarity measures are described next.

Let I be a set of integers. The *span* of I is $span(I) = \max I - \min I + 1$, i.e., the size of the minimum contiguous interval containing I. The *position* of I is $\min I$, and the *shape* of I is the set $\{i - \min I \mid i \in I\}$. An integer set Q with position zero is called a *shape*. For any shape Q and integer i, let Q_i denote the set with shape Q and position i, i.e., $Q_i = \{i + j \mid j \in Q\}$. Let $Q_i = \{i_1, i_2, \ldots, i_q\}$, where $i = i_1 < i_2 < \cdots < i_q$, and let $S = s_1 s_2 \ldots s_m$ be a string. For $1 \leq i \leq m - span(Q) + 1$, the *$Q$-gram* at position i in S, denoted by $S[Q_i]$, is the string $s_{i_1} s_{i_2} \ldots s_{i_q}$. For example, if $S = \texttt{acagagtct}$ and $Q = \{0, 2, 3, 6\}$, then $S[Q_1] = S[Q_3] = \texttt{aagt}$ and $S[Q_2] = \texttt{cgac}$.

A *match alignment* M_τ of a transcript τ is the set of pairs of positions that are matched to each other. For example, $M_{\texttt{MIMRDMR}} = \{(1,1),(2,3),(5,5)\}$. For a set I of integers, let $M_\tau(I)$ be the set to which M_τ maps I, i.e., $M_\tau(I) = \{j \mid i \in I \text{ and } (i,j) \in M_\tau\}$. A *$Q$-hit* in a transcript τ is a pair (i,j) of integers such that $M_\tau(Q_i) = Q_j$. The *Q-profit* $p_Q(\tau)$ of a transcript τ is the number of its Q-hits, i.e., $p_Q(\tau) = |\{(i,j) \mid M_\tau(Q_i) = Q_j\}|$. Using p_Q as the profit function defines the *Q-similarity* of two strings P and S as $s_Q(P,S) = \max_{\tau \in \mathcal{T}(P,S)} p_Q(\tau)$.

For any $b_1, g, b_2 > 0$, let (b_1, g, b_2) denote the *one-gap shape* $\{0, \ldots, b_1 - 1, b_1 + g, \ldots, b_1 + g + b_2 - 1\}$. For a one-gap shape $Q = (b_1, g, b_2)$, let $Q^{+1} = (b_1, g+1, b_2)$ and $Q^{-1} = (b_1, g-1, b_2)$ (or $Q^{-1} = \{0, \ldots, b_1 + b_2 - 1\}$ if $g = 1$). Then, a *$Q \pm 1$-hit* in a transcript τ is a pair (i,j) of integers such that $Q_j \in \{M_\tau(Q_i^{-1}), M_\tau(Q_i), M_\tau(Q_i^{+1})\}$. The *$Q \pm 1$-profit* of τ, denoted by $p_{Q\pm1}(\tau)$, is the number $Q \pm 1$-hits in τ, i.e., $p_{Q\pm1}(\tau) = |\{(i,j) \mid Q_j \in \{M_\tau(Q_i^{-1}), M_\tau(Q_i), M_\tau(Q_i^{+1})\}\}|$. The *$Q \pm 1$-similarity* of two strings P and S is $s_{Q\pm1}(P,S) = \max_{\tau \in \mathcal{T}(P,S)} p_{Q\pm1}(\tau)$.

For $c = c_H$ and $p = p_Q$, Definition 1 gives the thresholds that were used by the Hamming distance filters in [2]. If Q is the contiguous shape $\{0, 1, \ldots, q-1\}$, the threshold is the same as given by Lemma 1 and also used in the Levenshtein distance filter. The corresponding similarity measure s_Q is not exactly the one used by the filter algorithms but there is a close connection. If $\tau \in \mathcal{T}(P,S)$, then a Q-hit (i,j) in τ implies $P[Q_i] = S[Q_j]$. Therefore, the number of matching Q-grams between P and S is at least $s_Q(P,S)$, and thus at least the threshold if $d(P,S) \leq k$. The number of matching Q-grams may be higher than $s_Q(P,S)$. For example, `ab` and `ba` have two matching $\{0\}$-grams (`a` and `b`) even though $S_{\{0\}}(\mathtt{ab},\mathtt{ba}) = 1$. The exact value of $s_Q(P,S)$ could be computed by a more careful analysis of the matching q-grams, but it may not be worth the computational effort.

For $c = c_L$ and $p = p_{Q\pm1}$, Definition 1 gives the thresholds used by the one-gapped q-gram filters for the Levenshtein distance. What was said about the relation of the filter and the similarity measure s_Q above, holds for this case, too.

4 The minimum coverage

The two main properties of a filter are filter speed and filtration efficiency. When using an index the dominating factor for the filter speed is the total number of hits that have to be accessed and processed to locate the potential matches in T. Predicting the number of hits is straighforward using the number of shapes in P and the value q of matching characters per shape with the following equation:

$$hits \approx \frac{1}{\Sigma^q} 3|P||T|$$

The factor of 3 comes in for one-gap shapes since we use 3 shapes per starting position in P.

The filtration efficiency of a filter, i.e., the probability of having a random match depends in a complex way on the basic shape, defining the *quality* of the shape. In [2], we used the notion of minimum coverage to predict the quality of shapes. This was vital due to the large number of available shapes. When only using shapes with one gap the number of possible choices is dramatically reduced which allows experimental evaluation of all candidate shapes. However we also wanted to analyse the general concept of quality prediction for this case, especially since it was easily verifiable with the experimental results.

For the case of the Hamming distance the *minimum coverage* was defined as the size of the smallest union of t shapes with different positions. This corresponds to the minimum number of 'fixed' characters required to match between a pattern and a text substring for there to be t matching Q-grams. The probability of matching a certain arrangement of t Q-grams depends exponentially on the number of fixed characters in that arrangement. To exactly compute the probability of a random match one would have to take into account all possible arrangements of t or more shapes. However, since those arrangements with the lowest or close to the lowest number of fixed characters are much more likely, one can compute a good approximation using only the most probable arrangements, i.e. those with the minimum coverage. The expected number of matches given a certain minimum coverage mc would therefore be roughly proportional to $1/\Sigma^{mc}$.

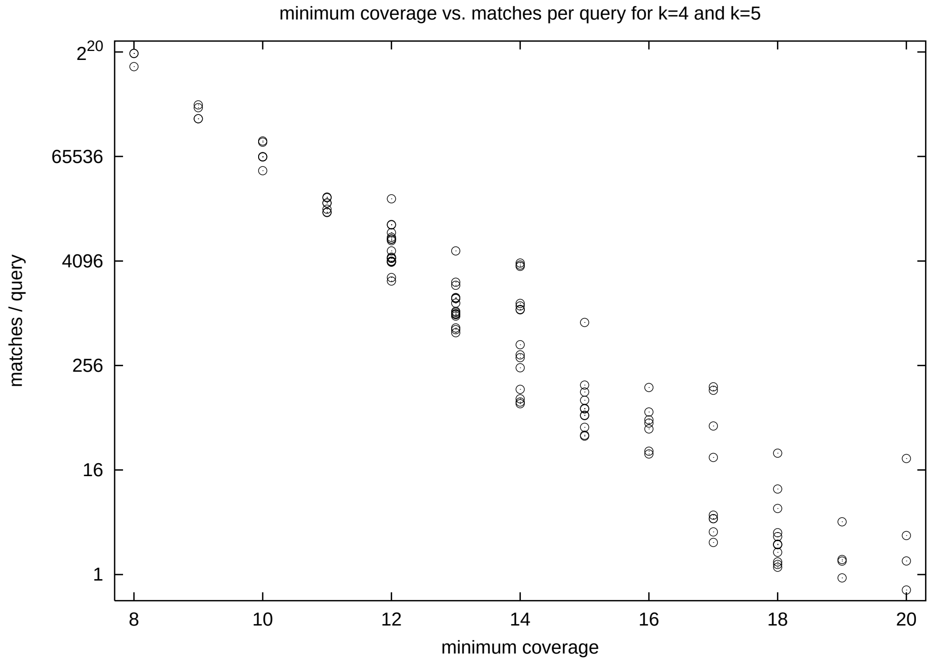

Fig. 1. Correlation between minimum coverage and filter efficiency.

Since we still only use one shape for matching in the text, the minimum coverage and the number of fixed characters remain the same here. The additional shapes used in the pattern can however increase the number of possible pattern substrings which match the fixed characters in the text. For cases where t shapes are arranged with a large overlap, this increase is negligible. In general this is the case for the minimum coverage. We evaluated the correlation between minimum coverage and the experimental results for all shapes and display them in Figure 1. It is obvious that while there is a small loss in correlation the overall predictive capability of the minimum coverage remains intact.

5 Experiments

To test the one-gap q-grams in practice, we performed some experiments on a randomly generated DNA database with 50 million basepairs (even and independent distribution of characters). The queries we used were 1000 random strings of length 500. The query length was chosen since it is a typical query length in many computational biology applications.

It should be noted that the threshold used in filtering was computed for $m = 50$ which is a typical value for finding local matches in DNA. This difference between query and window length has the effect that, while the filter is still guaranteed to report all positions where there is an approximate occurrence of a substring of length 50 of the query, it will also lead to a moderate increase in the number of potential matches reported due to the increase in the number of shapes for which one searches. [3]

For the edit distance k we used values of 4 and 5, making the experimental setting correspond to typical high similarity local alignment problems in shotgun sequencing [14] and EST clustering [8, 1]. The database contained no actual matches of this quality, i.e., all potential matches reported by the filter were false positives.

q	best for k=4
8	#####---###
9	#######----##
10	#######---###
11	#######--####
12	#########--###
13	#########-####
q	best for k=5
8	######---##
9	######---###
10	#######-###

Table 1. The gapped shapes used in Figure 2.

Like in our earlier paper we compared the gapped shapes with the contiguous shapes used in the classic q-gram lemma. For $k = 4$ and $k = 5$ we tested all shapes with $q \geq 8$. For $k = 4$ there are 87 such shapes and for $k = 5$ there are 35. From these shapes we picked, for each value of q, those with the best experimental performance and compared them with the filter based on the classic q-gram lemma. The best gapped shapes are shown in Table 1. Figure 2 compares them to contiguous q-grams both in theory (expected number of hits vs. minimum coverage) and in practice (hits vs. matches).

The top plot shows the values used to predict filtration efficiency (the minimum coverage) and filter speed (the expected number of hits) for both contiguous q-grams and the best one-gap shapes. The bottom graph contains the actual experimental results showing the average number of hits as well as the average number of matches per query (averaged across all 1000 queries). The expected number of hits in the first plot was computed taking

[3] A better filter efficiency can be achieved by counting the hits separately for each substring of length 50. This should not have a significant effect on the relative performance of the different shapes and we can therefore use this approximation for comparing them. Looking at the results one should keep in mind that the absolute values for filtration efficiency are not the best possible but slightly worse.

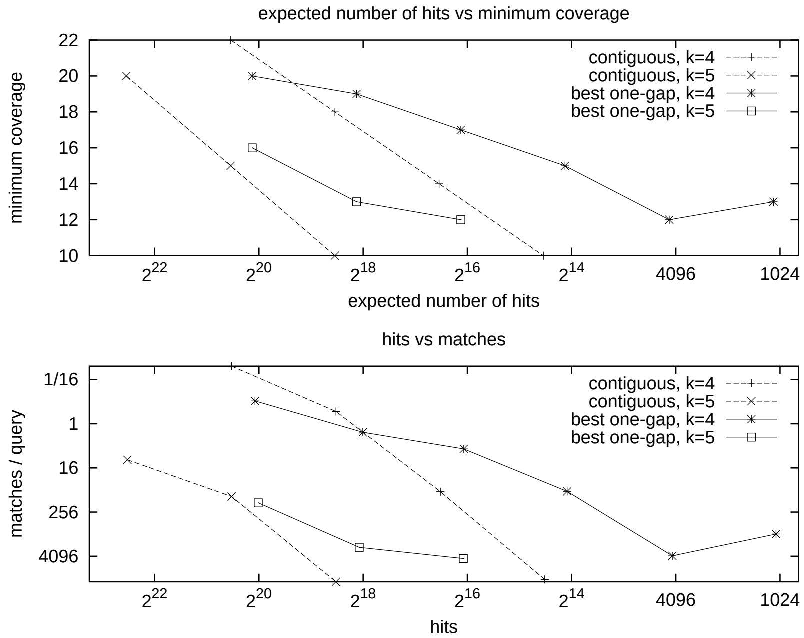

Fig. 2. Comparison of ungapped with gapped shapes.

into account the fact that we use 3 different shapes for each possible starting position in the query as described in Section 4.

Looking at Figure 2 one can observe that also for the Levenshtein distance the shapes with one gap show a better performance than ungapped shapes. In general they allow a substantial increases in the possible values of q and/or the filtration efficiency. The higher values of q make them prime candidates for index–based implementations. The comparison between predicted and actual performance reinforces the correlation described in Section 4 and with it the predictive capability of the minimum coverage.

Another point we want to make is that the proper choice of the shapes used for filtering is very important. To illustrate this we want to mention that for a fixed set of parameters the best shapes had filtration efficiencies that differed by as much as a factor of 10^6. The difference in filtration efficiency between the median and the best shape was still up to a factor of 50.

6 Concluding remarks

We have shown that suitably chosen one-gap q-grams combined with a simple technique to compensate for insertions and deletions in the gap can significantly improve the performance of the basic q-gram filter for the Levenshtein distance. This demonstrates that they are worth studying in further research on the problem of approximate string matching using the Levenshtein distance.

Aside from looking at shapes with only one gap it might be interesting to look at shapes with more than one gap. It remains to be seen wether the added number of shapes is worth the potential increase in filter quality. Also, techniques like generating word neighborhoods for q-grams, which are for example used in BLASTP, could perhaps be adapted to gapped q-grams. Other possibilities include combining two or more different shapes into one filter. The framework for computing thresholds for more complex filters has been provided with the definitions in this paper.

References

1. S. Burkhardt, A. Crauser, P. Ferragina, H.-P. Lenhof, E. Rivals, and M. Vingron. q-gram based database searching using a suffix array (QUASAR). In *Proc. 3rd Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 77–83. ACM Press, 1999.
2. Stefan Burkhardt and Juha Kärkkäinen. Better filtering with gapped q-grams. In *Proc. 12th Annual Symposium on Combinatorial Pattern Matching*, volume 2089 of *LNCS*, pages 73–85. Springer, 2001.
3. A. Califano and I. Rigoutsos. FLASH: A fast look-up algorithm for string homology. In *Proc. 1st International Conference on Intelligent Systems for Molecular Biology*, pages 56–64. AAAI Press, 1993.
4. D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.
5. N. Holsti and E. Sutinen. Approximate string matching using q-gram places. In *Proc. 7th Finnish Symposium on Computer Science*, pages 23–32, 1994.
6. P. Jokinen and E. Ukkonen. Two algorithms for approximate string matching in static texts. In *Proc. 16th Symposium on Mathematical Foundations of Computer Science*, volume 520 of *LNCS*, pages 240–248. Springer, 1991.
7. J. Kärkkäinen. Computing the threshold for q-gram filters, 2002. Manuscript.
8. A. Krause and M. Vingron. A set-theoretic approach to database searching and clustering. *Bioinformatics*, 14:430–438, 1998.
9. O. Lehtinen, E. Sutinen, and J. Tarhio. Experiments on block indexing. In *Proc. 3rd South American Workshop on String Processing (WSP)*, pages 183–193. Carleton University Press, 1996.
10. G. Navarro. *Approximate Text Searching.* PhD thesis, Dept. of Computer Science, University of Chile, 1998.
11. G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.
12. P. A. Pevzner and M. S. Waterman. Multiple filtration and approximate pattern matching. *Algorithmica*, 13(1/2):135–154, 1995.
13. E. Ukkonen. Approximate string matching with q-grams and maximal matches. *Theor. Comput. Sci.*, 92(1):191–212, 1992.
14. J. Weber and H. Myers. Human whole genome shotgun sequencing. *Genome Research*, 7:401–409, 1997.