Technical Report #2008-01
Department of Computer Science
Memorial University of Newfoundland
St. John's, NL, Canada

PATCHWORK VERIFICATION IN A FILTERING APPROACH TO APPROXIMATE PATTERN MATCHING

by

Christoph J. Richter (1) and Wolfgang Banzhaf (2)

[1]Department of Computer Science, University of Dortmund, 44221 Dortmund, Germany, Email: christoph.richter@cs.uni-dortmund.de

[2] Department of Computer Science, Memorial University of Newfoundland, St. John's, NL, Canada A1C 5S7, Email: banzhaf@cs.mun.ca

Department of Computer Science
Memorial University of Newfoundland
St. John's, NF, Canada A1B 3X5

January 2008

Patchwork Verification in a Filtering Approach to Approximate Pattern Matching

Christoph J. Richter
University of Dortmund
Dept. of Computer Science
44221 Dortmund, Germany
christoph.richter@cs.uni-dortmund.de

Wolfgang Banzhaf
Memorial University of Newfoundland
Dept. of Computer Science
St. John's, NL, Canada A1C 5S7
banzhaf@cs.mun.ca

A verification method for the partitioning into exact search filtering approach in approximate pattern matching will be shown here. We will calculate the limits of applicability and demonstrate the usefulness for longer patterns and smaller alphabets (e. g. DNA), when searching with high error levels.

Keywords: algorithms, approximate pattern matching, filtering, verification

1 Introduction

Approximate pattern matching is the problem of finding all positions in a text T, where a pattern P matches with at most k errors. In a more formal way, the problem of approximate pattern matching can be defined as follows: Given a text $T = t_1 \ldots t_n$, and a pattern $P = p_1 \ldots p_m$ $(t_i, p_j \in \Sigma)$, return the set $\{|x\bar{P}|, T = x\bar{P}y \wedge d(P, \bar{P}) \leq k\}$, where $x, \bar{P}$ and y are substrings of T, $|.|$ returns the length of a string and $d(P, \bar{P})$ gives the edit distance between P and $\bar{P}$. The edit distance between two strings characterizes the number of transformation operations (insertion, deletion and replacement) that are necessary to transform one string into an other one.

Since this problem has a great variety of applications in different areas like computational biology, text retrieval, and others [4, 6, 10], a lot of algorithm have been designed to solve this problem. Good overviews are given in [6] and [9].

The general solution principle for this problem utilizes dynamic programming and takes $O(nm)$ time [12, 13]. Based on this principle, many improved algorithms have been developed achieving up to $O(kn)$ in the worst case and $O(kn/\sqrt{\sigma})$ in the average case, where σ is the size of the alphabet Σ (like the algorithm of Chang and Lampe [2]).

A more advanced class of algorithms for approximate pattern matching solves the problem in two phases. The idea of the first phase, the filtering or search phase, is to

identify areas in the text, where an approximate matching possibly may occur (this may happen via discarding areas, where no approximate matching can appear at all). The second phase, the checking or verification phase, then checks all these areas separately with one of the basic algorithms for approximate pattern matching. While first phase can be done in $O(n)$, the verification cost of every area is basically quadratic during the second phase. Thus, the applicability of a filtering algorithm depends on the domination of the first phase. Naturally, for higher error levels $\alpha = k/m$ more verification in the second phase is expected. To be more robust to higher error levels, it is necessary to reduce the amount of verification.

There are a few approaches that deal with the issue of lowering the share of verification. Besides these approaches, in section 2 the specific kind of filtering used in this paper is discussed briefly also. In section 3 we present the approach of *patchwork verification*, which basically tries to avoid checking of overlapping areas. This approach is general in the sense that every approximate matching of the pattern within the given interval is found, but it is easily extendable to return new hits only, if the text is processed linearly. Patchwork verification is analyzed in section 4, before we draw conclusions in the last section.

2 Related Work

To improve filtering algorithms with lowering the share of the verification phase, different ideas has been presented. Giegerich et al. [3] mixed both phases of the filtering algorithm. With the information of the search phase about the maximal number of errors left, the checking phase can be stopped prematurely if in the progress of checking the actual number of errors shows that an approximate matching is not possible anymore.

With *hierarchical verification* another idea was presented by Navarro and Baeza-Yates [5, 7, 8]. The basis of this method is a simple fact: If a pattern of length m matches with k errors and the pattern is split into j pieces, at least one of these pieces matches with $\lfloor k/j \rfloor$ errors. For hierarchical verification, the pattern is recursively halved and thus split into 2^j pieces until the pieces are small enough to be searched with $\lfloor k/2^j \rfloor$ errors conveniently. If one of the pieces is found, for complete verification the process of halving is reverted and level by level two pieces are united and checked for an occurrence with twice as many errors as before. If on every level the verification is positive, the occurrence of the whole pattern is verified on the last level. In case of a negative verification on any level, the whole verification process is stopped, because it is sure that the pattern does not occur with at most k errors in the text a this position.

A different approach to reduce the overall amount of checking was presented in [**?**]. There, a grammar of the text is used to identify repetitions that only needs to be checked once. The same grammar is also used to skip searching in some areas of the text.

Though it may be difficult in some cases, in general all these approaches could be

combined with any filtering method. Here, we work with a partitioning into exact search filter. The filtering phase is determined by the same fact that builds the basis for hierarchical verification. The pattern is split into $k+s$ pieces and each of the pieces can be searched with $\lfloor \frac{k}{k+s} \rfloor$ errors. For $s \geq 1$, exact search of each piece is possible. This kind of filter was proposed by Wu and Manber [15] with $s = 1$ and is taken in this paper also. Other than Wu and Manber, who used an extension of shift-or [1], we use a multi pattern variant of the Boyer–Moore–Sunday algorithm [14] for exact searching. Here, the algorithm of Chang and Lampe [2] is applied in the verification phase, though in principle any approximate pattern matching algorithm could be used.

3 Patchwork Verification

The general idea of *patchwork verification* is to extend the verification algorithm for a better handling of overlapping calls. Whenever an exact matching subpattern could be localized in the filtering phase, the verification algorithm is called to check a certain area. Naturally, when the exact matchings of two subpatterns are too close to each other, the verification areas may overlap. In the following, with patchwork verification a method is described that considers these overlaps.

Assuming the subpattern P_i of P matches at position t in T. Without any additional knowledge, the area $[pos_b, pos_e]$ that is necessary to be verified can be determined as follows. It could be possible that P_i is located at the very beginning of P and thus

$$pos_e = t + k + m - 1 \tag{3.1}$$

it is obtained. On the other hand, P_i could be the last symbol of P and thus it is:

$$pos_b = t - k - (m - 1) \tag{3.2}$$

Considering additionally the length $|P_i|$ of the matched subpattern, the start of the interval can be defined more precise as at least the symbols of P_i follow t. This results in:

$$pos_b = t - k - (m - |P_i|) \tag{3.3}$$

If a verification algorithm is called frequently with overlapping areas, checking the given interval $[pos_b, pos_e]$ completely results in multiple verification of some positions. To avoid this, here, the idea is to remember the calculation state and the results of the last call of the verification algorithm and to reuse this information to reduce the calculation effort if possible.

During the calculation of an approximate matching with a pattern of length m, every state loses its influence after $m - 1$ positions. For a seamless continuation of the last verification this influence must not exists anymore during the actual verification at the

end position of the last verification. Assuming the last verified area was $[oldpos_b, oldpos_e]$, the condition for the possibility of taking advantage of overlappings with copying hits is

$$(oldpos_b \leq pos_b) \wedge (pos_b \leq oldpos_e - m + 1) \wedge (oldpos_e < pos_e). \tag{3.4}$$

Considering this situation, there are four different areas (cf. Figure 1):

- $[oldpos_b, pos_b - 1]$: This area is not of interest for the actual verification.
- $[pos_b, pos_b + m - 2]$: If hits were found in this area during the last verification, the influence of former positions may be the reason and so this area needs to be verified separately (without considering overlaps). If no hits were found during the last verification, this area can be ignored.
- $[pos_b + m - 1, oldpos_e]$: If hits were found in this area during the last verification, these hits can be copied for the actual verification.
- $[oldpos_e + 1, pos_e]$: Nothing is known about this area, so the verification can be continued here using the final state of the last verification.

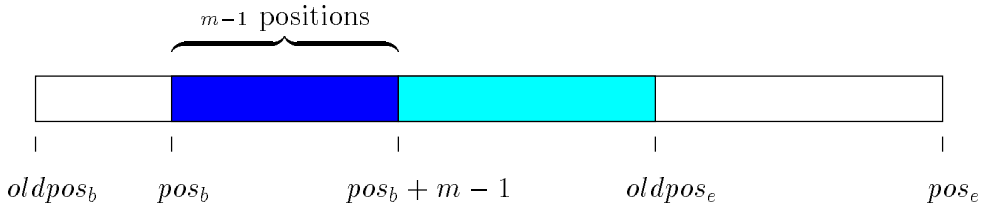


Figure 1: Patchwork verification. The influence areas of the previous verification are colored in gray. In the light gray area, hits from the previous verification can be copied. If there are hits in the dark gray area, this area needs to be verified separately again.

Integrating the distinction of these areas into a basic approximate pattern matching algorithm, a new verification algorithm is obtained, which considers overlapped checking areas. This resulting algorithm provides exactly all approximate matchings between the pattern and the text in the given area.

4 Analysis

In this section, patchwork verification will be evaluated. First, we estimate where patchwork is better than plain verification with the basic algorithm and we analyze its performance. Then, the slightly modified version of hierarchical verification is explained, which finally is used for the comparison with patchwork verification.

In general, any approximate string matching algorithm could serve as basic verification algorithm. Here, the algorithm of Chang and Lampe [2] was implemented. Both, hierarchical and patchwork verification were implemented as exhaustive checking algorithms, i. e. every approximate matching between the given text interval and the pattern is reported.

The algorithms were implemented in Java and all experiments were done on a 2.4 GHz Linux PC with 1 GB RAM. Within the experiments at least 20 repetitions were done, building a basis for the standard deviation bars in the figures. The search patterns were selected randomly from the text. The text was either random with varying alphabet sizes σ, or DNA (the complete genome of Haemophilus influenzae Rd, 1.77 MB in size, 70 characters per line) with an alphabet size of four, save for a few exceptions.

Compared to just plain verification of the given interval $[pos_b, pos_e]$, patchwork verification is of advantage, if overlappings occur, i. e. if it is $pos_b \leq oldpos_e$ with regard to the last interval checked $[oldpos_b, oldpos_e]$. Using the interval limits of equations 3.1 and 3.2, we achieve

$$t_{new} \leq t_{old} + 2m + 2k - 2. \tag{4.1}$$

In principle, for patchwork verification to outstand plain verification, only two consecutive calls of the verification algorithm on intervals fulfilling condition 4.1 are necessary. If the average distance of two exact matching subpatterns falls below $2m + 2k - 2$, advantage can be taken from the overlappings about every time, the verification algorithm is called. Assuming an equal distribution of text characters and with a subpattern length of $m/(k+1)$, the average distance between two of the $k+1$ exact matching subpatterns is

$$t_{new} - t_{old} = \frac{\sigma^{\frac{m}{k+1}}}{k+1}. \tag{4.2}$$

Thus, condition 4.1 can be transformed to

$$\sigma^{\frac{m}{k+1}} \leq (2m + 2k - 2)(k+1). \tag{4.3}$$

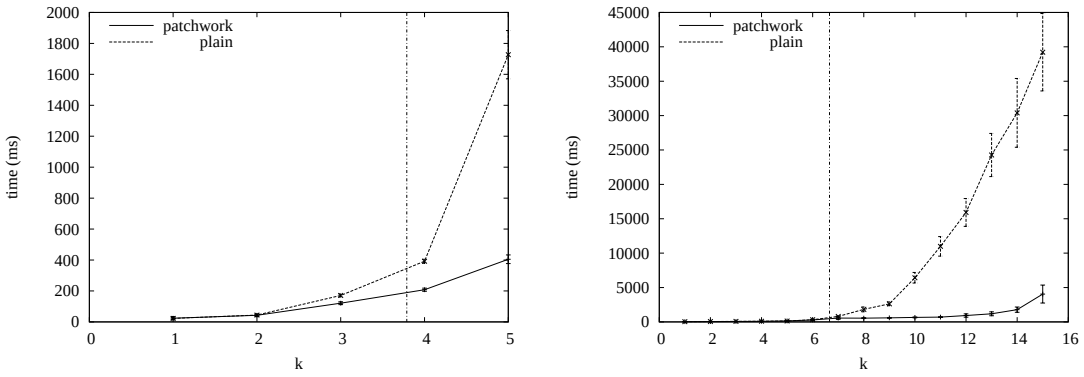
Figure 2 shows random text examples illustrating this condition. In the examples, it is obvious that patchwork verification outstands plain verification even before the limit given with condition 4.3 is reached for k.

If two consecutive verification intervals $[oldpos_b, oldpos_e]$ and $[pos_b, pos_e]$ are close together, some results of the first interval also hold for the second interval and thus can be copied (cf. Figure 1). This happens for

$$pos_b + m - 1 \leq oldpos_e. \tag{4.4}$$

Using the interval limits of equations 3.1 and 3.2 and the average exact matching distance of equation 4.2, we achieve

$$\frac{\sigma^{\frac{m}{k+1}}}{k+1} \leq m + 2k - 1 \tag{4.5}$$



(a) $\sigma = 10, n = 1000000, m = 10$ (b) $\sigma = 5, n = 1000000, m = 30$

Figure 2: Condition 4.3 in different random text searches. The condition is fulfilled for $k \geq 4.1342$ in (a) and for $k \geq 7.2154$ in (b).

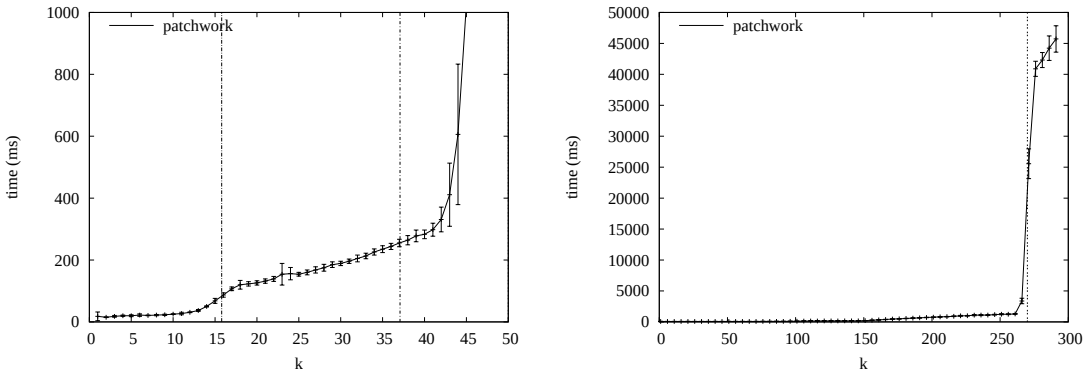
as the condition for reusing of results.

Furthermore, the most restricting limit is reached, when the interval $[pos_b, pos_b + m - 1]$ needs to be checked everytime. This interval is checked only, if a hit was already found within this interval, otherwise it is skipped. A hit exists in this interval, if the pattern matches in $[oldpos_b, pos_b + m - 1]$ with at most k errors. The average edit distance between two patterns of length m is between $m(1 - e/\sqrt{\sigma})$ and $2m(1 - 1/\sqrt{\sigma})$ [6] and it is conjectured that the true value is $m(1 - 1/\sqrt{\sigma})$ for large σ [11]. Since the interval is approximately of size m (for large error levels k/m it becomes $m - 1$), there is a hit in the interval approximately if it is

$$m(1 - 1/\sqrt{\sigma}) \leq k. \tag{4.6}$$

Figure 3 shows exemplarily the general runtime performance of patchwork verification and the calculated limits. Obviously, the impact of condition 4.5 can not be noticed, because only hits are copied for the considered interval. Since the number of hits increases with the error level, no clear influence at a certain point can be expected.

To achieve expected linear time, for a filtering algorithm it is important that the verification phase does not dominate, i. e. the overall verification costs are $O(n)$. Patchwork verification is called for every of the $k + 1$ exact matching subpatterns of length $\frac{m}{k+1}$. At every text position, the probability for this to happen is $(k + 1)/\sigma^{\frac{m}{k+1}}$. Assuming patchwork verification is used (i. e. condition 4.3 holds), assuming further interval $[pos_b, pos_b + m - 1]$ does not need checking (i. e. condition 4.6 holds) and neglecting the costs for copying hits in the interval $[pos_b + m - 1, oldpos_e]$ (cf. Figure 1) the costs per



(a) $\sigma = 4, m = 100, n = 100000$, maximum time restriced to 1000ms

(b) $\sigma = 100, m = 300, n = 100000$

Figure 3: In (a) the limit for k fulfilling condition 4.3 is marked with the left plumb line, while the right plumb line marks condition 4.5. Condition 4.6 is fulfilled for $k \geq 50$. In (b) only condition 4.6 is marked showing the general conformity with the measured result.

call are quadratic in the length of the verified area and thus they are $(\sigma^{\frac{m}{k+1}}/(k+1))^2$. To achieve linear expected time, we should have

$$n \frac{k+1}{\sigma^{\frac{m}{k+1}}} \left(\frac{\sigma^{\frac{m}{k+1}}}{k+1} \right)^2 \leq cn \tag{4.7}$$

which is equivalent to

$$\frac{\sigma^{\frac{m}{k+1}}}{k+1} \leq c. \tag{4.8}$$

With $m/(k+1) \approx 1/\alpha$ it is

$$\frac{\sigma^{\frac{1}{\alpha}}}{k+1} \leq c. \tag{4.9}$$

To solve for α, the following weaker inequality is used (replacing k by $m-1$):

$$\frac{\sigma^{\frac{1}{\alpha}}}{m} \leq c. \tag{4.10}$$

Together with equation 4.6 this results in a gross approximation of the interval where linearity is expected (for any constant c):

$$\frac{1}{\log_\sigma cm} \leq \alpha < 1 - \frac{1}{\sqrt{\sigma}}. \tag{4.11}$$

Of course, the limit for linearity of the partitioning into $k+1$ pieces filter still holds additionally and thus [6], linearity is also expected for

$$\alpha < 1/(3 \log_\sigma m). \tag{4.12}$$

Considering equation 4.11, for patchwork verification reasonable values of α are reached for small alphabets and longer patterns.

To integrate hierarchical verification (section 2) into the general filtering approach, a slight modification was necessary. Originally, hierarchical verification successively halves the pattern until the subpatterns are small enough to be searched with an appropriate number of errors. Since the smallest subpatterns are already given here with splitting the pattern into $k+1$ pieces, these subpatterns are successively melted until the whole pattern is reassembled. For melting, the idea of successive halving is reverted, i.e. in every step the subpattern results from a calculation, dividing the whole pattern into a power of 2 pieces, where each piece consists of a number of smallest subpatterns, originating from the $k+1$ partitioning.

A practical problem with hierarchical verification that needs to be considered, is the possible ambiguity of basic subpatterns. If a basic subpattern exists more than once in the pattern, hierarchical verification can not be stopped as long as not every possible occurrence is checked. If, for instance, the pattern `aaxxaaaa` was split into 4 pieces ($k=3$) and `aa` was found without error, it is not only necessary to check for `aaxx` with $\lfloor 3/2 \rfloor = 1$ errors, but also `aaaa`.

For hierarchical verification linear time is expected for

$$\alpha < 1/log_\sigma m \tag{4.13}$$

when used with the partitioning into $k+1$ pieces filter [6, 5, 7]. This limit equals the lower bound of the interval given in equation 4.11, when setting $c=1$.

Figure 4 shows patchwork and hierarchical verification on DNA data with a search-pattern of length $m=300$. The linear time limits given in equations 4.12 and 4.11 easily can be identified in the run of the curves. Moreover, this example clearly shows the good performance of patchwork verification on longer patterns and smaller alphabets.

5 Conclusions

A verification method making the partitioning into exact search filtering approach in approximate pattern matching applicable for higher error levels α was presented. Further, we have calculated and demonstrated the limits of linearity when using this method. The method of patchwork verification is especially useful for longer patterns and basically small alphabets like it is in DNA data.

We have seen that the linearity when using patchwork verification for higher error levels starts where linearity for hierarchical verification ends. Thus, it is an obvious idea,

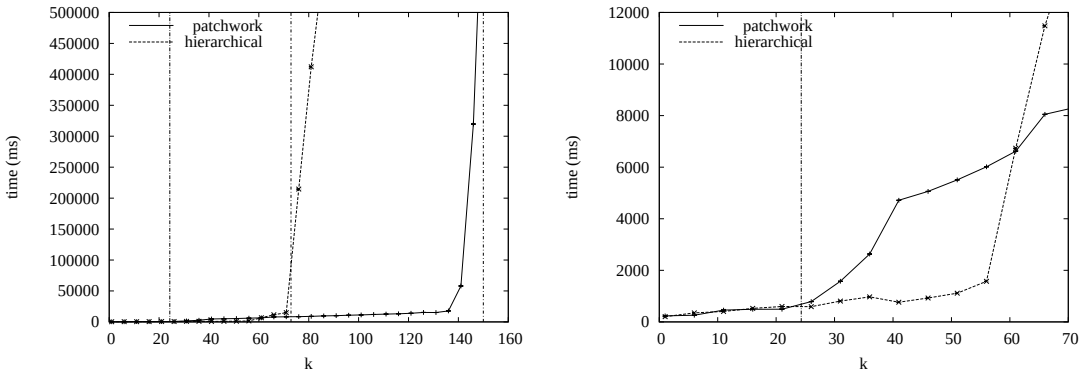


Figure 4: Patchwork and hierarchical verification on DNA data with a longer pattern ($m = 300$). (a) The leftmost plumb line gives the limit for α for normal filtering (equation 4.12). The middle line marks the limit given in equation 4.13 which equals the lower bound of the interval in condition 4.11. The upper border of this interval is marked with the rightmost line. (b) Zoom in on the first part of the graph shown in (a).

to switch at least between these methods depending on the actual error level to achieve an overall good filtering algorithm.

References

[1] R. A. Baeza-Yates and G. H. Gonnet, *A new apporach to text searching*, Communications of the ACM, 35 (1992), pp. 74–82. Preliminary version in ACM SIGR'89.

[2] W. I. Chang and J. Lampe, *Theoretical and empirical comparisons of approximate string matching algorithms*, in Proceedings of the 3rd Annual Symposium on Combinatorial Pattern Matching (CPM'92), A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, eds., vol. 644 of LNCS, Tucson, Arizona, USA, April/May 1992, Springer, pp. 175–184.

[3] R. Giegerich, F. Hischke, S. Kurtz, and E. Ohlebusch, *A general technique to improve filter algorithms for approximate string matching*, in Proceedings of the Fourth South American Workshop on String Processing (WSP'97), R. Baeza-Yates, ed., Valparaiso, Chile, November 1997, Carleton University Press, pp. 38–52.

[4] K. Kukich, *Techniques for automatically correcting words in text*, ACM Computing Surveys, 24 (1992), pp. 377–439.

[5] G. Navarro, *Approximate Text Searching*, PhD thesis, Department of Computer Science, University of Chile, Santiago, Chile, December 1998.

[6] G. Navarro, *A guided tour to approximate string matching*, ACM Computing Surveys, 33 (2001), pp. 31–88.

[7] G. Navarro and R. Baeza-Yates, *Very fast and simple approximate string matching*, Information Processing Letters (IPL), (1999), pp. 65–70.

[8] G. Navarro and R. Baeza-Yates, *Improving an algorithm for approximate pattern matching*, Algorithmica, 30 (2001), pp. 473–502. Previous version: Tech. Rep. TR/DCC-98-5, Dept. of Computer Science, University of Chile.

[9] G. Navarro, R. Baeza-Yates, E. Sutinen, and J. Tarhio, *Indexing methods for approximate string matching*, IEEE Data Engineering Bulletin, 24 (2001), pp. 19–27. Special issue on Managing Text Natively and in DBMSs. Invited paper.

[10] D. Sankoff and J. B. Kruskal, eds., *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, Massachusetts, 1983.

[11] D. Sankoff and S. Mainville, *Common Subsequences and Monotone Subsequences*, in Sankoff and Kruskal [10], 1983, ch. 17, pp. 363–365.

[12] P. H. Sellers, *The theory and computation of evolutionary distances: Pattern recognition*, Journal of Algorithms, 1 (1980), pp. 359–373.

[13] T. F. Smith and M. S. Waterman, *Identification of common molecular subsequences*, Journal of Molecular Biology, 147 (1981), pp. 195–197.

[14] D. M. Sunday, *A very fast substring search algorithm*, Communications of the ACM, 33 (1990), pp. 132–142.

[15] S. Wu and U. Manber, *Fast text searching allowing errors*, Communications of the ACM, 35 (1992), pp. 83–91.