Overlap Matching

Amihood Amir[*]
Bar-Ilan University & Georgia Tech

Richard Cole[†]
New York University

Ramesh Hariharan[‡]
Indian Institute of Science

Moshe Lewenstein[§]
Bar-Ilan University

Ely Porat[¶]
Bar-Ilan University & Weizmann Institute

Abstract

We propose a new paradigm for string matching, namely *structural matching*. In structural matching, the text and pattern contents are not important. Rather, some areas in the text and patterns are singled out, say intervals. A "match" is a text location where a specified relation between the text and pattern areas is satisfied.

In particular we define the structural matching problem of *Overlap (Parity) Matching*. We seek the text locations where all overlaps of the given pattern and text intervals have even length. We show that this problem can be solved in time $O(n \log m)$, where the text length is n and the pattern length is m.

As an application of overlap matching, we show how to reduce the *String Matching with Swaps* problem to the overlap matching problem. The *String Matching with Swaps* problem is the problem of string matching in the presence of local swaps. The best known deterministic upper bound for this problem was $O(nm^{1/3} \log m \log \sigma)$ for a general alphabet Σ, where $\sigma = min(m, |\Sigma|)$.

Our reduction provides a solution to the pattern matching with swaps problem in time $O(n \log m \log \sigma)$.

Key Words: Design and analysis of algorithms, combinatorial algorithms on words, pattern matching, pattern matching with swaps, non-standard pattern matching.

1 Introduction

The last few decades have prompted the evolution of pattern matching from a combinatorial solution of the exact string matching problem [12, 16] to an area concerned with approximate matching of

[*] Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8770; amir@cs.biu.ac.il. Partially supported by NSF grant CCR-96-10170, BSF grant 96-00509, and a BIU internal research grant.

[†] Courant Institute, New York University, cole@cs.nyu.edu. Supported in part by NSF grants CCR-95-03309 and CCR-98-00085.

[‡] Indian Institute of Science, Bangalore, ramesh@csa.iisc.ernet.in. Work done in part while visiting NYU and supported by NSF grants CCR-95-03309 and CCR-98-00085.

[§] Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8407; moshe@cs.biu.ac.il. Partially supported by the Israel Ministry of Science Eshkol Fellowship 061-1-97.

[¶] Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8407; porately@cs.biu.ac.il.

various relationships motivated by computational molecular biology, computer vision, and complex searches in digitized and distributed multimedia libraries [11, 7]. To this end, two new paradigms were needed – *"Generalized matching"* and *"Approximate matching"*.

In generalized matching the input is still a text and pattern but the "matching" relation is defined differently. The output is all locations in the text where the pattern "matches" under the new definition of match. The different applications define the matching relation. Examples are string matching with "don't cares" [12], parameterized matching [8, 4], less-than matching [3], and swapped matching [21, 2, 9]. Lower bound results on generalized matching can be found in [22].

Even under the appropriate matching relation there is still a distinction between *exact matching* and *approximate matching*. In the latter case, a distance function is defined on the text. A text location is considered a match if the distance between it and the pattern, under the given distance function, is within the tolerated bounds. Below are some examples that motivate approximate matching. In computational biology one may be interested in finding a "close" mutation, in communications one may want to adjust for transmission noise, in texts it may be desirable to allow common typing errors. In multimedia one may want to adjust for lossy compressions, occlusions, scaling, affine transformations or dimension loss.

The earliest and best known distance functions are Levenshtein's *edit distance* [19] and the *Hamming distance*. Let n be the text length and m the pattern length. Lowrance and Wagner [20, 24] proposed an $O(nm)$ dynamic programming algorithm for the extended edit distance problem. In [13, 17, 18] $O(kn)$ algorithms are given for the edit distance with only k allowed edit operations. Recently, Cole and Hariharan [10] presented an $O(nk^4/m \ + n + m)$ algorithm for this problem. Amir, Lewenstein and Porat [6] presented faster algorithms for the Hamming distance case.

Both above paradigms have an important trait in common – matching is dependent on the *alphabet symbols* in the respective pattern and text locations. In this paper we propose a new paradigm – **Structural Matching**. In this model, the content of the pattern and text is not important. What is important is the structure of these strings. Certain areas in the text and pattern are identified and a "match" of the pattern in the text is a location where these special areas satisfy a required relation. Structural Matching is motivated by two reasons. The first one is an "end" and the second one is a "means".

In molecular biology, it has long been a practice to consider special areas by their structure. Examples are repetitive genomic structures [14] such as *tandem repeates*, *LINEs* (Long Interspersed Nuclear Sequences) and *SINEs* (Short Interspersed Nuclear Sequences) [15]. Many problems in biology can be expressed as structural matching problems, thus streamlining and identifying the combinatorial nature of the problem.

The second reason is a functional one. The rich repertoire of relations between areas in the text and pattern can offer interesting tools for the solution of hitherto unresolved problems. In this paper we demonstrate such a use of structural matching for providing the fastest known algorithm for *swap matching*.

The *Pattern Matching with Swaps* problem (the *Swap Matching* problem, for short) requires finding all occurrences of a pattern of length m in a text of length n. The pattern is said to match the text at a given location i if adjacent pattern characters can be swapped, if necessary, so as to make the pattern identical to the substring of the text starting at location i. All the swaps are constrained to be disjoint, i.e., each character is involved in at most one swap.

The importance of the swap matching problem lies in recent efforts to understand the complexity of various generalized pattern matching problems. Until recently there were no known upper bounds better than the naive $O(nm)$ algorithm for the swap matching problem. Indeed, this problem was described in 1995 as one of the open problems in non-standard string matching [21].

Amir et al [2] obtained the first non-trivial results on this problem. They showed that the case when the size of the alphabet set Σ exceeds 2 can be reduced to the case when it is exactly 2 with a time overhead of $O(\log^2 \sigma)$. (The reduction overhead was reduced to $O(\log \sigma)$ in the journal version [1].) They then showed how to solve the problem for alphabet sets of size 2 in time $O(nm^{1/3} \log m)$, which is the best deterministic time bound known to date. Amir et al. [5] also give certain special cases for which $O(m\text{polylog}(m))$ time can be obtained. However, these cases are rather restrictive. Cole and Hariharan [9] give a randomized algorithm that solves the swap matching problem over a binary alphabet in time $O(n \log n)$.

In this paper we define a structural matching problem – the *Overlap (Parity) Matching Problem* – as follows.
INPUT: Text T of length n with marked intervals (substrings), and pattern P of length m with marked intervals (substrings).
OUTPUT: The text locations ℓ for which *all* overlaps of the marked intervals of T and marked intervals of P have *even*-length overlap.

We present a deterministic algorithm that solves the overlap matching problem in time $O(n \log m)$.

We then reduce the swap matching problem over binary alphabet to the overlap parity problem. Coupled with the alphabet reduction of [1, 9] it gives an algorithm for swap matching over general alphabet whose running time is $O(n \log m \log \sigma)$.

There are three main contributions in this paper.

1. The introduction of a new model in pattern matching, that of structural matching.
2. An efficient solution of the overlap matching problem.
3. The surprising time complexity of $O(n \log m)$ for solving the swap matching problem over binary alphabets. Until recently it was open whether the problem had a $o(nm)$ solution.

Paper organization. This paper is organized in the following way. In section 2 we give basic definitions. In sections 3, 4, 5 and 6 we solve the overlap matching problem. In section 7 we define the swap matching problem. In section 8 we prove a key lemma and show how to utilize this lemma in order to reduce swap matching to overlap matching.

2 Problem Definition

Consider a linear structure composed of contiguous units, for short a *segment*, e.g. an array. We say that a segment can be *marked* or *unmarked*. A *structural string* is a concatenation of marked and unmarked segments.

Overlap Matching is defined as follows:

Input: A structural string, P, which we will call the *pattern*, of length m units (i.e. the sum of the segment lengths), and a structural string, T, which we call the *text*, of length $n \geq m$ units.

Output: All text locations k, where, when P is aligned to start at k, each pair of marked text segment-marked pattern segment that overlap have *even*-length overlap.

Alternatively, we can replace the even-length overlap requirement with an *odd*-length overlap requirement. Since this problem can be reduced to the even-length case without much difficulty, we will only consider the even-length case.

Another way to visualize the pattern and the text is as regular strings partitioned into "segments" with several segments marked. Thus we can consider P to be $p_1 p_2 ... p_m$ and T to be $t_1 t_2 ... t_n$, the usual way of viewing a pattern and a text in string matching. However, it must be noted that the Overlap Matching problem is a structural problem and not a contextual problem, whereas, the problems in string matching are contextual problems and not structural. Therefore, even though we use the standard string matching notation for P and T the content of the elements is irrelevant to the problem.

3 Algorithm Outline for Overlap Matching

In the problem we desire only even-length overlap of marked segments. In other words, there is a match at a text location iff there is no odd-length overlap of any of the pairs of marked segments. This overlap parity property allows us to consider pairs of marked segments *separately*. As soon as a pair is found with odd-length overlap, it immediately leads to the conclusion that there is no match in that location.

The main idea of the algorithm is to separate the marked segments of the text and pattern into a small number of groups. In each of these groups it will be possible to check for the overlap parity property in time $O(n \log m)$ using polynomial multiplications (which can be done in time $O(n \log m)$ using FFT in a model with word length m bits). In the following sections we handle the different cases. Some of these cases necessitate new and creative uses of convolutions.

3.1 Grouping Text Segments by Parity of Starting and Ending Location

It will be important for us to know whether the marked segment we are dealing with starts at an odd or even text location. We also would like to know whether it ends at an odd or even text location. Consequently, we define new texts where each text has *exactly* those marked segments of a given start and end parity, with all other text elements defined as ϕ (*don't care*) and never contribute an error. (In the polynomial multiplication there will always be a 0 in these text locations.)

Definition: T^{oo} is a string of length n where for every location i, if t_i is in a marked segment whose first element is in an odd location and whose last element is in an odd location, $T^{oo}[i] = 1$. In all other locations j, $T^{oo}[j] = \phi$.

In a similar fashion define T^{oe},T^{eo},T^{ee}.

Example:

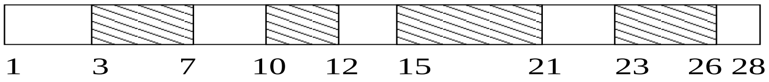

Figure 1: An example text.

$T^{oo} = \phi\phi11111\phi\phi\phi\phi\phi\phi\phi1111111\phi\phi\phi\phi\phi\phi\phi$
$T^{ee} = \phi\phi\phi\phi\phi\phi\phi\phi\phi111\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi$
$T^{oe} = \phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi1111\phi\phi$
$T^{eo} = \phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi\phi.$

Note that the segments of T^{oo} are exactly the marked segments of T that start and end at an odd location. Thus, it is clear that in every location where *none* of the above four text strings (T^{oo},T^{oe},T^{eo},T^{ee}) violate the parity property there is a match.

3.2 Grouping the Pattern Segments

The pattern segments are defined in exactly the same way as the text segments and we would like to group them in the same way. However, there is a difficulty here in "nailing down" the parity of a location, since the pattern is shifted and compared to every text location. However, since the only property we have used in our grouping was the parity of the text location, it is clear that in all the pattern alignments that start in odd locations, the parity of the start and end segment locations will be the same. Similarly, these parities will be the same for all pattern alignments that start in even text locations.

Thus we limit ourselves to two cases. The *Odd Result Case* and the *Even Result Case.* The *Odd Result* case is simply the pattern alignments starting in odd text locations. The *Even Result* case is pattern alignments starting in even text locations.

When we fix an alignment result case it is possible to define the parity of the starting and ending locations of each pattern segment. Thus we can get, for the odd result case, patterns $PO^{oo}, PO^{oe}, PO^{eo}$ and PO^{ee}. Similarly, for the even result case we consider patterns $PE^{oo}, PE^{oe}, PE^{eo}$ and PE^{ee}. We are now ready for the algorithm.

begin Algorithm

```
for $X = O, E$ do:
  for $ti = o, e$ do
    for $tj = o, e$ do
      for $pi = o, e$ do
        for $pi = o, e$ do
          check Odd Overlap property for $T^{ti,tj}$ and $PX^{pi,pj}$
          { Note that when $X = O$ we only consider the results in the odd text locations
          and when $X = E$ we only consider the results in the even text locations. }
```

end Algorithm

In the subsequent sections we will show how to implement the check for odd-length overlap (the forbidden property) for each of the cases in time $O(n \log m)$. The intrepid reader will realize that there are 32 cases to consider. Don't panic! Because of symmetry reasons, there are only three cases we need to describe. All others are similar.

The cases we describe are the following. First fix the result parity to the Odd Result case. It is clear that the the Even Result case is symmetric. Since we don't need the X parameter in the algorithm, we will henceforth ignore it for the sake of a simpler notation. We are now down to the combinations $T^{ti,tj}$ and $P^{pi,pj}$. This gives us 16 cases. We will handle separately only the following three types of cases:

1. $T^{ti,tj}$ and $P^{pi,pj}$ where either $ti = pi$ or $tj = pj$. (This type covers 12 cases.) These situations are handled in section 4.
2. $T^{ti,tj}$ and $P^{pi,pj}$ where $ti, tj = oe$ and $pi, pj = eo$; or where $ti, tj = eo$ and $pi, pj = oe$. These cases are handled in section 5.
3. $T^{ti,tj}$ and $P^{pi,pj}$ where $ti, tj = oo$ and $pi, pj = ee$; or where $ti, tj = ee$ and $pi, pj = oo$. These cases are handled in section 6.

4 Segments with Equal Parity Start

Consider the case $T^{ti,tj}$ and $P^{pi,pj}$ where $ti = pi$.

Observation 1 : *For every two segments, S_t in $T^{ti,tj}$, starting at location x and S_p in $P^{pi,pj}$, starting at location y, $|x - y|$ is always even. See all possibilities in figure 2 below.*

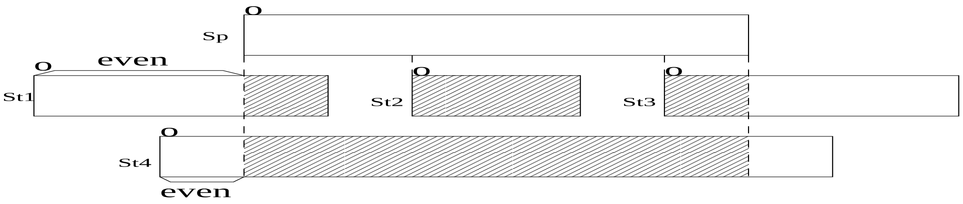

Figure 2: The cases of both text and pattern segments starting in locations with the same parity.

We are interested in the length of the segment overlaps (the shaded areas in figure 2). A segment overlap has odd length iff the property of lemma 3 holds. We now show a convolution for which the resulting value at location i is 0 iff there does not exists an odd-length segment overlap.

The Convolution: Construct a pattern $P' = p'_1 \cdots p'_m$ where

$$p'_i = \begin{cases} 0, & \text{if } P^{pi,pj}[i] = \phi; \\ 1, & \text{otherwise.} \end{cases}.$$

Construct a text $T' = t'_1 \cdots t'_n$ where every ϕ in $T^{ti,tj}$ is replaced by 0; and every segment in $T^{ti,tj}$ is replaced by an alternating segment of 1 and -1, starting at 1.

It is clear that for all cases where the starting location of a pattern segment is smaller than the starting location of a text segment the result of the convolution will be 1 if the length of the overlap is odd and 0 if it is even (since every text segment starts with a 1 and then alternates between -1 and 1). Because of observation 1, even when the text segment starts at a smaller location than the pattern segment, the difference between the starting locations has even length. Therefore in the area of the overlap, the text starts with a 1 and alternates between -1 and 1. Thus the convolution gives us the desired result.

This solves all eight cases of $T^{ti,tj}$ and $P^{pi,pj}$ where $ti = pi$. For the additional four cases where $tj = pj$ simply reverse the text and pattern and achieve the case considered above.

5 The Odd-Even Even-Odd Segments

Consider the case T^{oe} and P^{eo} (the case of T^{eo} and P^{oe} is symmetric).

Terminology: Let S_t be a text segment whose starting location is s_1 and whose ending location is f_1. Let S_p be a pattern segment being compared to the text at starting position s_2 and ending position f_2. If $s_1 < s_2 < f_2 < f_1$ then we say that S_t *contains* S_p. If $s_2 < s_1 < f_1 < f_2$ then we say that S_p *contains* S_t. If $s_1 < s_2 < f_1 < f_2$ then we say that S_t has a *left overlap* with S_p. If $s_2 < s_1 < f_2 < f_1$ then we say that S_t has a *right overlap* with S_p. We will sometimes refer to a left or right overlap as a *side overlap.*

Observation 2 *: For every two segments, S_t in T^{oe} and S_p in P^{eo} if either S_p is contained in S_t or S_t is contained in S_p then the overlap is of even length. If the overlap is a left overlap or right overlap then it is of odd length. See all possibilities in figure 3 below.*

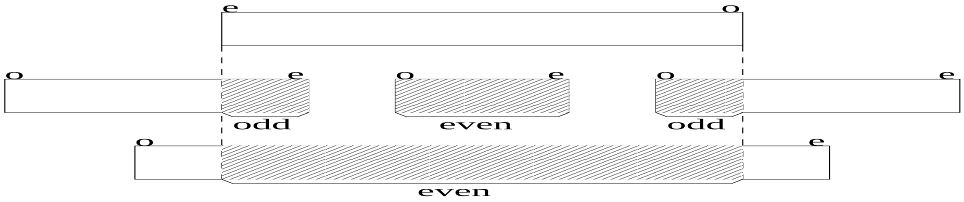

Figure 3: The cases where the text segment starts at an odd location and ends at an even location; the pattern segment does the opposite.

The correctness of the observation is immediate. Segments of these types have even length thus if one contains the other the overlap is necessarily of even length. Conversely, in case of a left or right overlap then the overlap starting and ending locations have the same parity, making the length of the overlap odd.

We will now show a convolution where segments that are contained in each other contribute a 0, and side overlaps contribute positive numbers. This convolution will be more complex than the previous one. The reason is that there is some inherent relation between text segments that *start* within a pattern segment. Likewise, there is a commonality between text segments that *end* within a pattern segment. Our problem is that we need to differentiate between cases that are naturally easier handled together.

Consider the following case. Let S_t be a segment of T^{oe} starting at text location s_1 and having length ℓ_t. Assume that the pattern P^{eo} of length ℓ_p is aligned to start at the text location such that the first symbol of segment S_p occurs at location s_2. Further assume that one of the segments is contained in the other. If we replace S_t by $s_1 1^{\ell_t - 2} - s_1$ and S_p by $s_2 1^{\ell_p - 2} - s_2$ (and ϕ by 0) then multiplication of these two segments will yield $k-2$, where k is the size of the overlap. See figure 4.

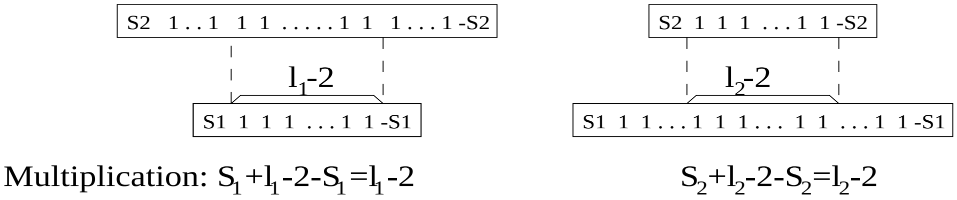
S2 1 . . 1 1 1 . . . . . 1 1 1 . . . 1 -S2
l1-2
S1 1 1 1 . . . 1 1 -S1
Multiplication: S1+l1-2-S1=l1-2
S2 1 1 1 . . . 1 1 -S2
l2-2
S1 1 1 . . . 1 1 1 . . . 1 1 . . . 1 1 -S1
S2+l2-2-S2=l2-2

Figure 4: Containment cases for text T^{oe} and pattern P^{oe}.

Conversely, if there is a side overlap of S_t with S_p then the multiplication will yield $\max(s_1, s_2) - \min(s_1, s_2) + k - 2$, where k is the size of the overlap. See figure 5.

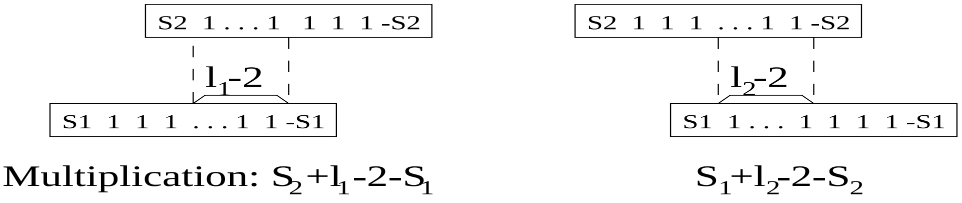

Figure 5: Edge intersection cases for text T^{oe} and pattern P^{eo}.

If we could solve two problems, we are done.

1. Remove the size of the overlap -2 from the result. (This will make the multiplication result be 0 when all overlaps are containments.)

2. Be able to actually replace every segment by its starting point, 1's and the negation of its starting point. (This will make the multiplication result positive when there exist side overlaps.) Being able to insert the starting point of the segment for the sake of multiplication is not at all simple to do since the same segment has $n-m$ different starting points!

However, if the above two problems could be solved by a convolution, then the result would be 0 iff all overlaps are containments iff all overlaps have even length (by observation 2).

Solution 1. The first problem can be easily solved. The convolution below provides, for every text location i, the sum $\sum_j (k_j - 2)$, where k_j are all the overlap lengths. All that is necessary, then, is to subtract the result of this convolution from the value obtained for each text location.

The Overlap Length Convolution: Replace every segment of length ℓ in both text and pattern by $01^{\ell-2}0$. Replace all ϕ's by 0's. Multiply.

Solution 2. The starting locations of all text segments are known in advance and thus the substitution of the segment by $s_1 1^{\ell_t - 2} - s_1$ can be done in linear time for the entire text. The pattern segment is problematic since it has $n-m$ different alignments. Therefore, we treat the pattern segments as if they all start relative to location 1.

The Zero Containment Convolution: Replace every segment of length ℓ_t starting in location s_1 of the text by $s_1 1^{\ell_t - 2} - s_1$. Replace every segment of length ℓ_p starting at location sp_1 of the *pattern* by $sp_1 1^{\ell_p -2} - sp_1$. Replace all ϕ's by 0's. Multiply.

The problem is that now the result of the multiplication of side overlaps (after subtraction of the overlap length convolution) may sometimes be positive and sometimes be negative. This may cause the zero containment convolution result to be a 0 even for overlaps that are not contained, i.e. of odd length. We will then not be able to distinguish between them and the cases of all overlaps being containments.

We now show how to adjust the zero containment convolution to give the desired result. The zero containment convolution (after subtracting the result of the overlap length convolution) causes all contained segments to give the result 0. Thus in location i_0 we get the value

$$\sum_{\ell \in A} sp_\ell - \sum_{k \in B} sp_k + \sum_{i \in C} s_i - \sum_{j \in D} s_j,$$

where A is the the set of pattern segments that have text segments overlapping them from the left, B is the the set of pattern segments that have text segments overlapping them from the right, C is the set of text segments that have right overlap with pattern segments and D is the set of text segments that have left overlap with pattern segments.

The problem is that the result we really want is

$$\sum_{\ell \in A} s_\ell - \sum_{k \in B} s_k + \sum_{i \in C} s_i - \sum_{j \in D} s_j.$$

In other words, we want the starting positions of the aligned pattern segments, rather than their starting positions relative to location 1. However, note that for location i_0, $s_x = i_0 + sp_x$. This means that the value we want in location i_0 is

$$\sum_{\ell \in A} (sp_\ell + i_0) - \sum_{k \in B} (sp_k + i_0) + \sum_{i \in C} s_i - \sum_{j \in D} s_j$$

This equals

$$\sum_{\ell \in A} sp_\ell - \sum_{k \in B} sp_k + \sum_{i \in C} s_i - \sum_{j \in D} s_j + \sum_{\ell \in A} i_0 - \sum_{k \in B} i_0.$$

We conclude that the desired value will be obtained if we add to the result of the zero containment convolution at every location i_0 the value $\sum_{\ell \in A} i_0 - \sum_{k \in B} i_0$. These values can be obtained by the following convolution.

The Shifting Convolution: Replace every pattern segment of length ℓ_p by $10^{\ell_p -2} - 1$ and every text segment of length ℓ_t by $01^{\ell_t -2}0$. Replace every ϕ by 0. Multiply.

It is easy to see that the result of this convolution in location i_0 is $\sum_{\ell \in A} i_0 - \sum_{k \in B} i_0$.

Definition: Let T be a text with segments $St_1, \ldots, St_x$ starting at locations $s_1, \ldots, s_x$, respectively. Let P be a pattern such that, when placed at location i_0, its segments $Sp_1, \ldots, Sp_y$ start at locations $v_1, \ldots, v_y$. Let $\{s_{i_1}, \ldots, s_{i_z}\}$ be the starting locations of text segments that have a left overlap with pattern segments, and let $\{v_{j_1}, \ldots, v_{j_z}\}$ be the starting location of the corresponding pattern segments. Similarly let $\{s_{k_1}, \ldots, s_{k_w}\}$ be the starting locations of text segments that have a right overlap with pattern segments, and $\{v_{\ell_1}, \ldots, v_{\ell_w}\}$ be the starting locations of the corresponding pattern segments.

The *external side length* of the pattern at location i_0 is

$$\sum_{r=1}^{y}(v_{i_r} - s_{j_r}) + \sum_{r=1}^{z}(s_{k_r} - v_{\ell_r}).$$

Note that containments do not contribute to the external side length. See figure 6 for an illustration.

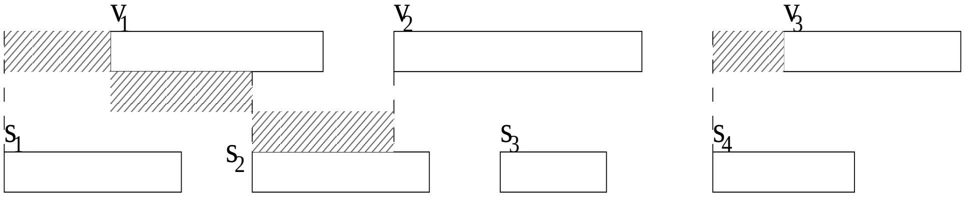

Figure 6: The sum of the shaded areas is the external side length:
$$(v_1 - s_1) + (v_2 - s_2) + (v_3 - s_4) + (s_2 - v_1).$$
Note that we are not counting external lengths of containments.

Lemma 1 *It is possible in time $O(n \log m)$ to provide, for every text location i, a value that is 0 if all overlaps of the pattern and text segments are containments, and positive otherwise.*

Proof: Add the results of the shifting convolution and the zero containment convolution and subtract from it the result of the overlap length convolution. We get the external side length which is 0 if all overlaps are containments and positive, otherwise. □

6 The Odd-Odd Even-Even Segments

Consider the case T^{oo} and P^{ee} (the case of T^{ee} and P^{oo} is symmetric).

Observation 3 *: For every two segments, S_t in T^{oo} and S_p in P^{ee} if either S_p is contained in S_t or S_t is contained in S_p then the overlap is of odd length. If the overlap is a left overlap or right overlap then it is of even length. See all possibilities in figure 7 below.*

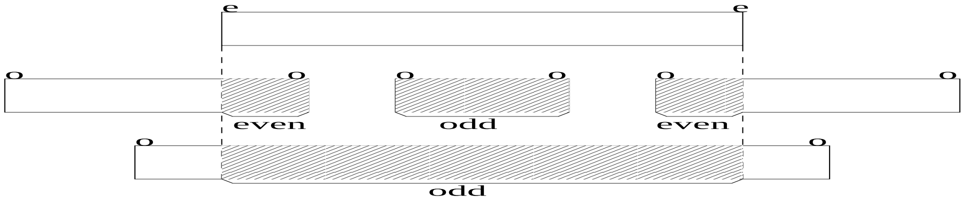

Figure 7: Every containment has odd length; every side overlap has even length.

The correctness of the observation is immediate. Segments of these types have odd lengths thus if one contains the other the overlap is necessarily of odd length. Conversely, in case of a left or right overlap then the overlap starting and ending locations have the opposite parity, making the length of the overlap even.

Definition: Let T be a text with segments $St_1, \ldots, St_x$ starting at locations $s_1, \ldots, s_x$, respectively. Let P be a pattern such that, when placed at location i_0, its segments $Sp_1, \ldots, Sp_y$ start at locations $v_1, \ldots, v_y$. Let $\{s_{i_1}, \ldots, s_{i_z}\}$ be the starting locations of text segments that have a left overlap with pattern segments or that contain a pattern segments, and let $\{v_{j_1}, \ldots, v_{j_z}\}$ be the starting location of the corresponding pattern segments. Similarly let $\{s_{k_1}, \ldots, s_{k_w}\}$ be the starting locations of text segments that have a right overlap with pattern segments or that are contained in pattern segments, and $\{v_{\ell_1}, \ldots, v_{\ell_w}\}$ be the starting locations of the corresponding pattern segments.

The *external overlap length* of the pattern at location i_0 is

$$\sum_{r=1}^{y}(v_{i_r} - s_{j_r}) + \sum_{r=1}^{z}(s_{k_r} - v_{\ell_r}).$$

Note that unlike the external side length, containments also contribute to the external overlap length. See figure 8 for an illustration.

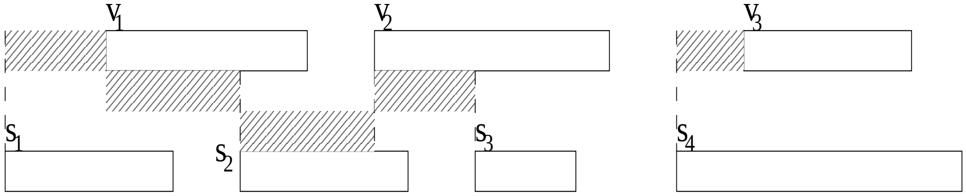

Figure 8: The sum of the shaded areas is the external overlap length:

$$(v_1 - s_1) + (v_2 - s_2) + (v_3 - s_4) + (s_2 - v_1) + (s_3 - v_2).$$

Note that $(s_3 - v_2) + (v_3 - s_4)$ would not be counted in the external side length.

We need a method to indicate whether there exist any overlaps. We will show that the external overlap length can be calculated by two convolutions. Since we know from section 5 how to compute the external side length, it suffices to see the difference between these two values. If it is 0 then there are no containments (and therefore no odd-length overlaps), otherwise there are containments.

The sum of the following two convolutions gives the external overlap length.

The External Contained and Right Length Convolution: Replace every segment of length ℓ in the pattern by $0123 \cdots \ell - 1$, and every segment of length ℓ in the text by $10^{\ell-1}$. Replace every ϕ by 0. Multiply.

The resulting value at every location i_0 is precisely the external length of the right overlaps and of the text segments contained in pattern segment overlaps. Left overlaps and text containing pattern give a 0. The next convolution computes precisely the remaining external lengths.

The External Containing and Left Length Convolution: Replace every segment of length ℓ in the text by $0123 \cdots \ell - 1$, and every segment of length ℓ in the pattern by $10^{\ell-1}$. Replace every ϕ by 0. Multiply.

Adding the results of the above two convolutions gives the external overlap length.

Lemma 2 *It is possible in time $O(n \log m)$ to provide, for every text location i, a value that is 0 if there are no containments either of text in pattern segments or of pattern in text segments, and positive otherwise.*

Proof: Subtract the external side length from the external overlap length. If there is a containment, we will get a positive number, otherwise we get 0. □

7 Swap Matching

Definition: Let $S = s_1 \ldots s_n$ be a string over alphabet Σ. A *swap permutation* for S is a permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$ such that

1. if $\pi(i) = j$ then $\pi(j) = i$ (characters are swapped).
2. for all i, $\pi(i) \in \{i-1, i, i+1\}$ (only adjacent characters are swapped).
3. if $\pi(i) \neq i$ then $s_{\pi(i)} \neq s_i$ (identical characters are not swapped).

For a given string $S = s_1 \ldots s_n$ and swap permutation π for S we denote $\pi(S) = s_{\pi(1)} s_{\pi(2)} \ldots s_{\pi(n)}$. We call $\pi(S)$ a *swapped version* of S.

For pattern $P = p_1 \ldots p_m$ and text $T = t_1 \ldots t_n$, we say that P *swap matches at location* i if there exists a swapped version P' of P that matches T starting at location i, i.e. $p'_j = t_{i+j-1}$ for $j = 1, \ldots, m$.

The *Swap Matching Problem* is the following:
INPUT: Pattern $P = p_1 \ldots p_m$ and text $T = t_1 \ldots t_n$ over alphabet Σ.
OUTPUT: All locations i where P swap matches T.

We note that the definition in [2] and the papers that followed is slightly different, allowing the swaps in the text rather than the pattern. However, it follows from Lemma 1 in [2] that both versions are of the same time complexity.

7.1 Reducing Large Alphabets to Binary Alphabets

In [2] it was shown how to reduce the swap matching problem over unbounded alphabets to the problem over a two letter alphabet with an $O(\log^2 |\Sigma|)$ multiplicative overhead. Here we outline a reduction requiring only an $O(\log |\Sigma|)$ factor detailed in [1]. Another unpublished $O(\log |\Sigma|)$ reduction appears in [9].

Definition: A $(\Sigma, 3)$-*universal set* is a set $S = \{\chi_1, \ldots, \chi_k\}$ of characteristic functions, $\chi_j : \Sigma \to \{0, 1\}$ such that for every $a, b, c \in \Sigma$, and for each of the eight possible combinations of $0-1$s, there exists χ_j such that $\chi_j(a), \chi_j(b), \chi_j(c)$ equals this combination.

We extend the definition of the functions χ_j to strings in the usual manner, i.e. for $S = s_1 \ldots s_n$, $\chi_j(S) = \chi_j(s_1)\chi_j(s_2) \ldots \chi_j(s_n)$.

Theorem 1 *[2] Let P be a pattern, T a text, both over an arbitrary alphabet Σ, and let $S = \{\chi_1, \ldots, \chi_k\}$ be a $(\Sigma, 3)$-universal set. P swap matches T at location i iff for all j, $\chi_j(P)$ swap matches $\chi_j(T)$ at location i.*

In [23] it was shown how to construct $(\Sigma, 3)$-universal set of cardinality $k = O(\log |\Sigma|)$ yielding the following.

Corollary 1 *A solution of the Swap Matching problem over alphabet $\{a, b\}$ of time $O(f(n, m))$ implies a solution of time $O(\log |\Sigma| f(n, m))$ over a general alphabet Σ.*

8 Reducing Swap Matching to Overlap Matching

Following the reduction to a binary alphabet, we assume that the text and the pattern both have only a's and b's. An alternating segment of a string $S \in \{a, b\}^*$ is a substring alternating between as and bs. A maximal alternating segment, or *segment* for short, is an alternating segment such that the character to the left of the leftmost character x in the alternating segment, if any, is identical to x, and similarly, the character to the right of the rightmost character y, if any, is identical to y.

We now show the key property necessary to reduce swap matching to overlap matching. To this end we partition the text and pattern into segments.

Lemma 3 *The pattern does not match in a particular alignment if and only if there exists a segment A in the text and a segment B in the pattern such that (1) the characters of A and B misalign in the overlap and (2) the overlap is of odd-length.*

Proof. First, we show the *if* part. For any overlap of segments that is of even length the overlapping portions of A and B can be made identical by swapping, if necessary. Note that these swaps stay within the overlapping portion so other segments are not affected. If the overlap is odd and the overlapping portions match exactly then nothing needs to be done. So clearly, if the pattern does not match it must be the case that there exists a segment A in the text and a segment B in the pattern that overlap with odd-length overlap and with misaligning characters in the overlap.

Consider the *only if* part now. Suppose A and B overlap such that the overlap is of odd length and, without loss of generality, the overlap has $u = (ab)^*a$ in the text and $v = (ba)^*b$ in the pattern. Then, for a match to occur, either the leftmost b in v must be swapped with the character to the left or the rightmost b in v must be swapped with the character to the right.

Clearly, the portion of B overlapping A cannot be a prefix of B for the above swap to be effective (the character preceding B, if any, is a b). It follows that the portion of B overlapping A must be a prefix of A. Once again the above swap is ineffective (the character preceding A is an a). Thus it follows that the pattern does not match at this alignment. $\Box$

Following Lemma 3, we would like to check each location whether there are overlapping segments with properties (1) and (2). To this end we will separate the segments of the text and pattern into two groups each, similar to what we did in the algorithm for the overlap matching.

We create these groups in such a way that the characters always misalign (property (1)) in the comparison of group vs. group. Moreover, the comparisons cover all possible misalignments of overlapping segments. Therefore, the necessary comparison is to check that the length of the overlap is odd (property (2)). We now describe the grouping.

8.1 Odd a and Even a Text Segments

The first two categories of text segments we consider are the *Even a segments*, where all the a's fall on even text locations, and the *Odd a segments*, where all the a's fall on odd text locations. We construct two new text (structural) strings, T_{odd-a} and T_{even-a}, each of them having length n. T_{odd-a} is a text whose marked segments are all the odd a segments of T in their exact locations. The text locations where there are even a segments are marked segments. Similarly, T_{even-a} is the (structural) text string whose marked segments are all the even a segments of T in their exact locations and the other segments are unmarked. In fact T_{even-a} and T_{odd-a} complement each other in the sense that they are exactly the same strings where each marked segment in one is the unmarked in the other.

Example: Let $T = ababbbaaababababbaababbaaabababa$.
The segments are: *abab b ba a ababab ba abab ba a abababa*
The odd a segments are the first, third, fifth, seventh, and ninth. Thus
T_{odd-a} = marked length 4, unmarked length 1, marked length 2, unmarked length 1, marked length 6, unmarked length 2, marked length 4, unmarked length 2, marked length 1, unmarked length 7.

8.2 Odd a and Even a Pattern Segments

The pattern segments are defined in exactly the same way as the text segments and we would like to group them in the same way. However, as in the overlap matching, the problem is to "nail down" the parity of a location, since the pattern is shifted and compared to every text location. However, since the only property we have used in our grouping was the parity of the text location, it is clear that in all the pattern alignments that start in odd locations, the parity of the a occurrences will be the same. Similarly, these parities will be the same for all pattern alignments that start in even text locations. Thus, we define P_{odd-a} and P_{even-a} in the same way we defined T_{odd-a} and T_{even-a}. The following lemma shows how to utilize the reduction to overlap matching and follows directly from the discussion.

Lemma 4 *P swap matches at an odd location, $2i+1$, of T iff P_{odd-a} overlap matches T_{even-a} at location $2i+1$ and P_{even-a} overlap matches T_{odd-a} at location $2i+1$.*
P swap matches at an even location, $2i$, of T iff P_{odd-a} overlap matches T_{odd-a} at location $2i$ and P_{even-a} overlap matches T_{even-a} at location $2i$.

Therefore, for swap matching we can state the following.

Theorem 2 *Swap Matching can be solved in $O(n \log m \log \sigma)$ for a general alphabet Σ, where $\sigma = min(m, |\Sigma|)$.*

References

[1] A. Amir, Y. Aumann, G. Landau, M. Lewenstein, and N. Lewenstein. Pattern matching with swaps. Submitted for publication.

[2] A. Amir, Y. Aumann, G. Landau, M. Lewenstein, and N. Lewenstein. Pattern matching with swaps. *Proc. 38th IEEE FOCS*, pages 144–153, 1997.

[3] A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation*, 118(1):1–11, April 1995.

[4] A. Amir, M. Farach, and S. Muthukrishnan. Alphabet dependence in parameterized matching. *Information Processing Letters*, 49:111–115, 1994.

[5] A. Amir, G.M. Landau, M. Lewenstein, and N. Lewenstein. Efficient special cases of pattern matching with swaps. *Information Processing Letters*, 68(3):125–132, 1998.

[6] A. Amir, M. Lewenstein, and E. Porat. Faster algorithms for string matching with k mismatches. In *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 794–803, 2000.

[7] A. Apostolico and Z. Galil (editors). *Pattern Matching Algorithms.* Oxford University Press, 1997.

[8] B. S. Baker. A theory of parameterized pattern matching: algorithms and applications. In *Proc. 25th Annual ACM Symposium on the Theory of Computation*, pages 71–80, 1993.

[9] R. Cole and R. Harihan. Randomized swap matching in $o(m \log m \log |\sigma|)$ time. Technical Report TR1999-789, New York University, Courant Institute, September 1999.

[10] R. Cole and R. Hariharan. Approximate string matching: A faster simpler algorithm. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 463–472, 1998.

[11] M. Crochemore and W. Rytter. *Text Algorithms.* Oxford University Press, 1994.

[12] M.J. Fischer and M.S. Paterson. String matching and other products. *Complexity of Computation, R.M. Karp (editor), SIAM-AMS Proceedings*, 7:113–125, 1974.

[13] Z. Galil and K. Park. An improved algorithm for approximate string matching. *SIAM J. Comp.*, 19(6):989–999, 1990.

[14] J. Jurka. Human repetitive elements. In R. A. Meyers, editor, *Molecular Biology and Biotechnology*, pages 438–441. VCH Publishers, New York, 1995.

[15] J. Jurka. Origin and evolution of alu repetitive elements. In R. J. Maraia, editor, *The Impact of Short Interspersed Elements (SINEs) on the Host Genome*, pages 25–41. R. G. Landes, New York, 1995.

[16] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Comp.*, 6:323–350, 1977.

[17] G. M. Landau and U. Vishkin. Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169, 1989.

[18] G.M. Landau, E. W. Myers, and J. P. Schmidt. Incremental string comparison. *SIAM J. Comp.*, 27(2):557–582, 1998.

[19] V. I. Levenshtein. Binary codes capable of correcting, deletions, insertions and reversals. *Soviet Phys. Dokl.*, 10:707–710, 1966.

[20] R. Lowrance and R. A. Wagner. An extension of the string-to-string correction problem. *J. of the ACM*, pages 177–183, 1975.

[21] S. Muthukrishnan. New results and open problems related to non-standard stringology. In *Proc. 6th Combinatorial Pattern Matching Conference*, pages 298–317. Lecture Notes in Computer Science 937, Springer-Verlag, 1995.

[22] S. Muthukrishnan and H. Ramesh. String matching under a general matching relation. *Information and Computation*, 122(1):140–148, 1995.

[23] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comp.*, pages 838–856, 1993.

[24] R. A. Wagner. On the complexity of the extended string-to-string correction problem. In *Proc. 7th ACM STOC*, pages 218–223, 1975.