Fourier transform methods for δ and (δ, γ) matching and other measures of string similarity

Peter Clifford[a], Raphaël Clifford[b,*] and Costas Iliopoulos[b]
[a] *Department of Statistics, 1 South Parks Road, Oxford, UK*
[b] *Algorithm Design Group, Department of Computer Science, King's College London, London, UK*

Technical Report TR-04-09, July 2004

Abstract

A unifying approach to the δ and (δ, γ) matching problems on strings of integers is developed using Fourier transform methods. For pattern length m and text length n, the running times for the proposed algorithms are shown to be $O(\delta n \log m)$ for both δ and (δ, γ) matching, effectively independent of the alphabet size. An $O(n\sqrt{m \log m})$ algorithm for the γ matching and total difference problems is also given thereby clarifying conjectures in the literature.

1 Introduction

The recognition of pattern and structure in large databases is a fundamental goal in the rapidly developing area of statistical data-mining. Applications include the analysis of high-frequency financial trading data, object recognition in image processing, the exploration of genomic data and the identification of melodic structure in musicology. An important class of problems can be specified in terms of a text string $t = t_1 \cdots t_n$ and a pattern string $p = p_1 \cdots p_m$ over an alphabet Σ. The classical string-matching problem is to find all occurrences of the pattern p in the text t, in other words to find

*Corresponding author. Email: raph@dcs.kcl.ac.uk

all indices i such that p and the substring $t^{(i)} \stackrel{\text{def}}{=} t_i \cdots t_{i+m-1}$ are identical. For many applications it is more common to define a measure of dissimilarity between strings with the object being to find which substrings of t match p approximately, in the sense that the dissimilarity between p and the substring is below some nominal value.

The focus of approximate string matching algorithms has traditionally been on symbolic data. However, in many cases the data are more naturally represented as a string of integer values. An important example is that of computer assisted music analysis. The pitch of each note can be represented on the chromatic scale or using MIDI notation as an integer chosen from some interval. A musical score can then be thought of as a set of strings of integers, each string representing a different instrument or *voice*. Approximate matching in this context, and in general when dealing with strings of numeric values, must take into account the distance between characters as well as any other considerations (see [9, 4] for surveys of string methods in music analysis). In this paper we present new algorithms for approximate matching on strings over an alphabet of integers. We focus on problems that can be described in the *generalised linear product* formalism of Fischer and Paterson [10]. Problems involving deletions, insertions and the associated edit distances that are important in bioinformatic applications (see e.g. [11]) are specifically excluded.

There is only one way in which two strings $p_1 \cdots p_m$ and $t_1 \cdots t_m$ can be the same but there are many ways of measuring their dissimilarity. The Hamming distance, for example, counts the number of mismatches, i.e. the number of locations where $p_j \neq t_j$. More generally, a numerical score $s(a, b)$ can be defined for a specific mismatch between symbols a and b and the *total score* calculated, i.e. $\sum_{j=1}^{m} s(p_j, t_j)$. When the alphabet is a subset of the real line, natural measures of dissimilarity are the *total difference*, $\sum_{j=1}^{m} |p_j - t_j|$; the *total squared difference*, $\sum_{j=1}^{m} (p_j - t_j)^2$; and the *maximum difference*, $\max_{j=1}^{m} |p_j - t_j|$. These are respectively, the L_1 distance, the square of the L_2 distance and the L_∞ distance between the vectors $(p_1, \ldots, p_m)$ and $(t_1, \ldots, t_m)$.

The fast Fourier transform (FFT) is the basis of several fast algorithms for approximate string-matching. The most important property of the FFT is that, for numerical strings, all the inner-products,

$$p \cdot t^{(i)} \stackrel{\text{def}}{=} \sum_{j=1}^{m} p_j t_{i+j-1}, \quad 1 \leq i \leq n - m + 1,$$

can be calculated accurately and efficiently in $O(n \log m)$ time (see e.g. [8], Chapter 32).

The basic idea when using FFTs to tackle approximate matching problems is to express the dissimilarity measure in terms of an inner-product. For example, since $(x-y)^2 = x^2 - 2xy + y^2$, we can express the total squared difference between p and $t^{(i)}$ as

$$\sum_{j=1}^{m} (p_j - t_{i+j-1})^2 = \sum_{j=1}^{m} p_j^2 - 2p \cdot t^{(i)} + \sum_{j=1}^{m} t_{i+j-1}^2.$$

The first and last terms can be calculated in $O(n+m)$ time and the middle terms can be calculated in $O(n \log m)$ time using the FFT. The exact matching problem is then solved immediately since there is an exact match at all values of i where the total squared difference is zero. Of course a similar argument can be made for any dissimilarity measure which is zero when there is an exact match and bounded away from zero otherwise. By using a score of $x/y + y/x - 2 = (x-y)^2/xy$, Cole and Hariharan [5] show that FFT methods can also solve the exact matching problem with wild cards in $O(n \log m)$ time, effectively independent of $|\Sigma|$.

In the δ, γ and (δ, γ) matching problems, the alphabet is assumed to be an interval of integers. The problems are defined as follows.

Problem 1 (δ matching) *Determine $\mathcal{I}_\delta$ where*

$$\mathcal{I}_\delta = \{i : \max_{j=1..m} \left|p_j - t_{i+j-1}\right| \leq \delta\}.$$

In other words, the problem is to find all indices i such that the maximum difference between p and $t^{(i)}$ is no larger than δ. For γ matching the problem is to find all indices i such that the total difference between p and $t^{(i)}$ is no larger than γ.

Problem 2 (γ matching) *Determine $\mathcal{J}_\gamma$ where*

$$\mathcal{J}_\gamma = \{i : \sum_{j=1}^{m} \left|p_j - t_{i+j-1}\right| \leq \gamma\}.$$

For (δ, γ) matching, we require both that the maximum difference is bounded by δ and that the total difference is no larger than γ. The problem is defined as follows.

Problem 3 ((δ, γ) matching) *Determine* $\mathcal{I}_\delta \cap \mathcal{J}_\gamma$.

An $o(nm)$ solution for δ matching can be derived by a reduction to the *less-than matching problem* [3]. In less-than matching the task is to find all i such that every value in p is less than or equal to its corresponding value in $t^{(i)}$. δ matching can be solved using two instances of less-than matching giving an overall time complexity of $O(n\sqrt{m \log m})$ using FFTs [2]. An alternative approach is given in [7] where an instance of δ matching is reduced to at most $2\delta + 1$ instances of exact matching with wild cards. Using FFTs again, the total time required is therefore $O(\delta n \log m)$. For many applications, including musical ones, δ does not depend on the size of the input and can be regarded as a constant. We present here a different and direct $O(\delta n \log m)$ solution to δ matching. This method is asymptotically faster than a reduction to less-than matching when δ is $o(\sqrt{m/\log m})$. For (δ, γ) matching we give a similar $O(\delta n \log m)$ solution which is faster than the current known $O(nm)$ bound if δ is $o(m/\log m)$.

An $o(nm)$ solution for γ matching, for arbitrary γ, can be derived from the total difference problem.

Problem 4 (Total difference) *Determine the total differences*

$$M_i = \sum_{j=1}^{m} \left| p_j - t_{i+j-1} \right|, \quad \textit{for } 1 \leq i \leq n - m + 1.$$

We will show that the total difference problem can be solved in $O(n\sqrt{m \log m})$ running time. Our method is based on the divide and conquer approach introduced by Abrahamson and developed by Amir [1, 3]. We believe this to be the first $o(nm)$ solution for this total problem.

The plan of the paper is as follows. In Section 2 we give a simple illustration of our approach to δ matching. In Section 3 we solve the δ and (δ, γ) problems in $O(\delta n \log m)$ time. In Section 4 we show that Abrahamson's method can be used to calculate γ matches for arbitrary values of γ in $O(n\sqrt{m \log m})$ time. Finally in Section 5 we consider general approximate matching problems and discuss why some are inherently more difficult than others.

2 Our techniques

Our algorithms for δ and (δ, γ) matching build on the ideas in [5]. We construct a function that is zero when there is a match between two symbols and larger than some fixed positive value otherwise. We then show how the function can be computed efficiently using FFTs. The main innovation is in the use of even periodic functions and their discrete cosine expansions to achieve this goal.

As a simple illustration, consider the δ matching problem with $\delta = 1$. First notice that the periodic function $\frac{1}{2} - \frac{1}{2}(-1)^x$ takes the value 0 when x is even and 1 when it is odd. If we define $g(x) = x^2 + \frac{1}{2}(-1)^x - \frac{1}{2}$, it follows that $g(x-y) = 0$ when $|x-y| \leq 1$ and $g(x-y) \geq 4$ otherwise. But we can write

$$g(x-y) = x^2 - 2xy + y^2 + \tfrac{1}{2}(-1)^x(-1)^y - \tfrac{1}{2},$$

since $x+y$ has the same parity as $x-y$. This is the key result. We now define new strings $\epsilon(t)$ and $\epsilon(p)$ where the jth element of $\epsilon(t)$ is 1 if t_j is even and -1 otherwise and similarly for $\epsilon(p)$. It follows that $\sum_{j=1}^{m} g(p_j - t_{i+j-1})$ can be expressed in terms of two inner-products $p \cdot t^{(i)}$ and $\epsilon(p) \cdot \epsilon(t)^{(i)}$, both of which can be calculated in $O(n \log m)$ time, plus other terms that are calculable in $O(n+m)$ time. The δ matching problem is then solved since, provided that the FFT is carried out to sufficient precision, we can identify indices where $\sum_{j=1}^{m} g(p_j - t_{i+j-1}) = 0$. In this case and the ones to follow, we need only enough precision to be able to distinguish 0 from any number greater than or equal to 1. The relative error of the Cooley-Tukey FFT method, for example, is $\epsilon \log n$ where ϵ is the machine floating-point precision (the smallest positive number such that $1 + \epsilon$ is distinguishable from unity in the floating point representation employed). Therefore, for any realistic size of input, non integer values resulting from the FFT calculation can simply be rounded to the nearest whole number without fear of mistake. See [12] for a more in depth discussion of FFT accuracy under different measures of error.

3 Algorithms for δ and (δ, γ) matching

We start by generalising the arguments that were used in Section 2 to tackle the δ matching problem for the special case $\delta = 1$. The idea is the same, namely to modify the squared difference by subtracting a periodic function

thereby obtaining a function that is zero over a range of differences. We start with a few definitions and basic results about vector spaces.

A real-valued function $f(x)$ defined on $\mathbb{Z}$, the set of integers, is even and periodic with period 2δ, if

$$f(x) = f(-x) \text{ and } f(x) = f(x + k2\delta) \text{ for all } x, k \in \mathbb{Z}$$

Using standard properties of the discrete cosine transform [8] a convenient basis for the space of these functions is the set of functions

$$h_k(x) = r(k)\cos(xk\pi/\delta),\ k = 0, \ldots, \delta,$$

where $r(k) = 1/\sqrt{2\delta}$ if $k \mod \delta = 0$ and $r(k) = 1/\sqrt{\delta}$ otherwise. These functions are orthonormal in the sense that $\sum_{x=1-\delta}^{\delta} h_j(x)h_k(x) = 0$ when $j \neq k$ and $\sum_{x=1-\delta}^{\delta} h_k^2(x) = 1$. Consequently, any even function $f(x)$ with period 2δ can be written as

$$f(x) = \sum_{k=0}^{\delta} \alpha_k h_k(x),$$

where the coefficients are given by

$$\alpha_k = \sum_{x=1-\delta}^{\delta} f(x)h_k(x).$$

We will be interested in two special cases of such functions, $f^{[1]}$ and $f^{[2]}$, where

$$f^{[1]}(x) = |x| \text{ and } f^{[2]}(x) = x^2 \text{ for } |x| \leq \delta.$$

Both $f^{[1]}(x)$ and $f^{[2]}(x)$ are even and periodic and so defined over the whole of $\mathbb{Z}$. We denote the coefficients of these functions by $\{\alpha_k^{[1]}\}$ and $\{\alpha_k^{[2]}\}$, respectively.

To tackle the δ matching problem, consider the function

$$g(x) = x^2 - f^{[2]}(x).$$

By construction, this is an even function with the property that $g(x) = 0$ for $|x| \leq \delta$ and $g(x) \geq 1$, otherwise. Figure 1 shows the functions x^2 and $f^{[2]}$ and the result after subtraction. The important point is that

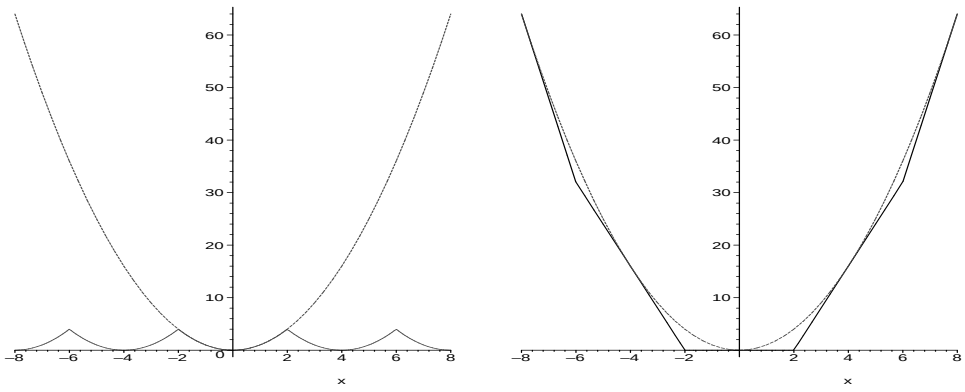

Figure 1: The functions x^2, $f^{[2]}(x)$ and $g(x) = x^2 - f^{[2]}(x)$: $\delta = 2$

$$\begin{aligned} g(x-y) = x^2 + y^2 - 2xy - \alpha_0^{[2]} r(0) \\ - \sum_{k=1}^{\delta} \alpha_k^{[2]} r(k) c_k(x) c_k(y) \\ - \sum_{k=1}^{\delta-1} \alpha_k^{[2]} r(k) s_k(x) s_k(y) \end{aligned}$$

where

$$\begin{aligned} c_k(x) &= \cos(xk\pi/\delta) \\ s_k(x) &= \sin(xk\pi/\delta), \end{aligned}$$

and we have used the fact that $s_0(x) = s_\delta(x) = 0$ and $c_0(x) = 1$. In other words, $g(x-y)$ involves a total of 2δ product terms, so the dissimilarity measure based on g can be expressed in terms of 2δ inner-products.

To perform δ matching we compute $\Sigma_{j=1}^{m} g(p_j - t_{i+j-1})$ for $1 \le i \le n-m+1$. Any value that is 0 indicates a match. We can also set any non-zero values to 1 to indicate a mismatch. The main steps of the algorithm are therefore:

1. Calculate the $\alpha_k^{[2]}$ coefficients using the equation $\alpha_k^{[2]} = \sum_{x=1-\delta}^{\delta} x^2 h_k(x)$
2. Compute $A = \sum_{j=1}^{m} p_j^2$ and $B_i = \sum_{j=1}^{m} t_{i+j-1}^2$ for $1 \le i \le n-m+1$
3. Compute $C_i = p \cdot t^{(i)}$ for $1 \le i \le n-m+1$
4. For each $k \in \{1, \ldots, \delta\}$, construct a pair of arrays p' and t' such that $p'_j = \alpha_k^{[2]} r(k) c_k(p_j)$ and $t'_j = c_k(t_j)$ and compute $D_i(k) = p' \cdot t'^{(i)}$ for $1 \le i \le n-m+1$

5. For each $k \in \{1, \ldots, \delta - 1\}$, construct a pair of arrays p' and t' such that $p'_j = \alpha_k^{[2]} r(k) s_k(p_j)$ and $t'_j = s_k(t_j)$ and compute $E_i(k) = p' \cdot t'^{(i)}$ for $1 \leq i \leq n - m + 1$
6. $\sum_{j=1}^{m} g(p_j - t_{i+j-1}) = A + B_i - 2C_i - \alpha_0 r(0) m - \sum_{k=1}^{\delta} D_i(k) - \sum_{k=1}^{\delta-1} E_i(k)$

The $\alpha_k^{[2]}$ need only be computed once for the particular value of δ that is of interest since they do not depend on the input pattern or text. Each coefficient takes requires $O(\delta)$ multiplications and additions to compute and so the total time to compute all $\alpha_k^{[2]}$ for a given δ is $O(\delta^2)$. The arrays A and B can be calculated simply in linear time. C can be calculated with a single inner-product calculation in $O(n \log m)$ time using FFTs. The arrays D and E require δ and $\delta - 1$ inner-product calculations respectively taking a total of $O(\delta n \log m)$ time to compute. It follows that δ matching can be carried out with 2δ inner product calculations in $O(\delta n \log m)$ time.

(δ, γ) matching

For (δ, γ) matching, we make use of the set of locations $\mathcal{I}_\delta$ where the δ matches occur, i.e.

$$\mathcal{I}_\delta = \big\{ i : \max_{j=1,\ldots,m} \big| p_j - t_{i+j-1} \big| \leq \delta \big\}.$$

Now consider the even periodic function $f^{[1]}(x)$. Recall that this is the same as $|x|$ when $|x| \leq \delta$. We define the periodic total difference between p and $t^{(i)}$ to be

$$\Delta_i = \sum_{j=1}^{m} f^{[1]}\big(p_j - t_{i+j-1}\big),$$

for $i = 1, \ldots, n - m + 1$.

Since $f^{[1]}(x - y)$ can expressed as a linear combination of $2\delta - 1$ product terms, i.e.

$$\begin{aligned} f^{[1]}(x - y) = {} & \alpha_0^{[1]} r(0) \\ & + \sum_{k=1}^{\delta} \alpha_k^{[1]} r(k) c_k(x) c_k(y) \\ & + \sum_{k=1}^{\delta-1} \alpha_k^{[1]} r(x) s_k(y) s_k(y), \end{aligned}$$

it follows that all the periodic differences can be calculated in $(2\delta-1)O(n \log m)$ time using FFTs. In particular, we can identify

$$\mathcal{J}_\gamma^* = \{i : \Delta_i \leq \gamma\},$$

the set of locations where the periodic total difference is no larger than γ.

The (δ, γ) matching problem is now solved. Locations that are in the intersection $\mathcal{I}_\delta \cap \mathcal{J}_\gamma^*$ will meet both the δ matching and γ matching criteria since total differences are correctly calculated for all locations in $\mathcal{I}_\delta$. The final algorithm requires $4\delta - 1$ inner-product calculations and takes $O(\delta n \log m)$ time overall.

4 Total-difference algorithm

A naive algorithm for the total difference problem involves $O(n)$ successive calculations each of length $O(m)$ and therefore runs in $O(nm)$ time. For the canonical case, $n = 2m$, this implies that the running time is quadratic in m. In [6] the question of whether the problem can be solved in subquadratic time is raised. We will give a subquadratic algorithm, running in $O(m\sqrt{m \log m})$ time for the canonical problem and hence by the usual argument in time $O(n\sqrt{m \log m})$ for text of length n. A solution to the total difference problem immediately gives a solution to the γ matching problem,

The essence of the method is firstly to solve an incomplete version of the problem using FFTs and then tidy up loose ends afterwards by straightforward calculations.

We start by observing that the difference between two numbers x and y can be calculated as the sum of four products

$$|x - y| = xI_xJ_y - I_xyJ_y + J_xyI_y - xJ_xI_y,$$

where $I_a = 1$ if $a > \theta$ and $I_a = 0$ otherwise and $J_a = 1 - I_a$, provided x and y are on opposite sides of some arbitrary value θ. When this is not the case the right hand side is 0. This observation forms the basis of the algorithm. In other words contributions to M_i from pairs of values (p_j, t_{i+j-1}) on opposite sides of specific thresholds are accumulated first. The remaining terms are collected in the second stage of the algorithm.

The algorithm

Assume $n = 2m$.

1) Partition
Sort the values in the set $\{p_1, p_2, \ldots, p_m\}$ in increasing order and let A be the associated array of indices of the sorted p-values. Now partition A from left to right into b successive arrays each of length m/b. Denote the kth of these arrays by A_k and let θ_k be its largest element. For notational convenience define $\theta_0 = -\infty$.

Next consider the set $\{t_1, t_2, \ldots, t_{2m}\}$. By sorting this set in increasing order, obtain a corresponding collection of arrays $\{B_k : k = 1, 2, \ldots, b\}$, where B_b contains all the indices j such that $t_j > \theta_{b-1}$ and B_k contains all the indices j such that $\theta_{k-1} < t_j \leq \theta_k$ for $k < b$. The end result is that we can associate each entry t_j in the text with m/b entries in the pattern, since j must belong to one of the arrays B_k and there are m/b members of of the array A_k.

2) FFT
For $k = 1, \ldots, b$ and x a real number define two functions:

$$I_k(x) = \begin{cases} 1 & \text{if } x > \theta_k \\ 0 & \text{otherwise,} \end{cases} \quad \text{and} \quad J_k(x) = \begin{cases} 1 & \text{if } \theta_{k-1} < x \leq \theta_k \\ 0 & \text{otherwise.} \end{cases}$$

For each k create four new text strings $\epsilon_{k1}(t), \epsilon_{k2}(t), \epsilon_{k3}(t)$ and $\epsilon_{k4}(t)$ with jth elements that are respectively $t_j I_k(t_j), I_k(t_j), J_k(t_j)$ and $t_j J_k(t_j)$. Correspondingly, create four new pattern strings $\nu_{k1}(p), \nu_{k2}(p), \nu_{k3}(p)$ and $\nu_{k4}(p)$ with jth elements that are respectively $J_k(p_j), -p_j J_k(p_j), p_j I_k(p_j)$ and $-I_k(p_j)$. With this construction

$$\sum_{j=1}^{m} |p_j - t_{i+j-1}| = \sum_{k=1}^{b} \sum_{\ell=1}^{4} \nu_{k\ell}(p) \cdot \epsilon_{k\ell}(t)^{(i)} + \quad \text{remainder,}$$

where the remainder contains terms from pairs of values (p_j, t_{i+j-1}) where $p_j \in A_k$ and $t_{i+j-1} \in B_k$ for some k.

3) The remainder
Now deal with the omitted cases, i.e. pairs of text and pattern elements that lie in associated arrays (A_k, B_k) for some k. Working through each of the arrays B_k, for each element of the text the number of associated elements in the pattern is m/b. The total number of such pairs considered is then $2m \times m/b$. For these cases, the contributions to the total differences can be calculated in the straightforward manner in $O(m^2/b)$ time.

4) The Running time
The running time for 1) is $O(m \log m)$. The running time for 2) is $4bO(m \log m)$, since $4b$ FFTs have to be calculated. The running time from 3) is $O(m^2/b)$, so the total time is $O(m^2/b + 4bm \log m)$. Taking $b = \sqrt{m/\log m}$, the running time is then $O(m\sqrt{m \log m})$ for the case $n = 2m$ and hence $O(n\sqrt{m \log m})$ in general.

5 Discussion

We have given new faster solutions to approximate matching problems on strings of integers. In particular, we have presented an algorithm for computing L_1 distances in $O(n\sqrt{m \log m})$ time (the total difference problem) and an $O(\delta n \log m)$ for δ and (δ, γ) matching. A number of important questions remain unresolved. For example, is it possible to calculate all L_∞ distances in $o(nm)$ time? Can Abrahamson's method be extended to the general L_p class of vector norms?

Other important problems for numeric string data arise when the values in the pattern can be *shifted* up or down by some constant when looking for a match. In the musical setting, for example, this corresponds to looking for a musical pattern in a database while allowing the key to change. For the L_2 norm we define the shift-matching problem as follows.

Problem 5 (L_2 shift matching) *Determine $\mathcal{L}_\delta$ where*

$$\mathcal{L}_\delta = \{i : \min_{l_i \in \mathbb{Z}} \{\sum_{j=1}^{m} (p_j - t_{i+j-1} + l_i)^2\} \leq \delta\}.$$

Optimal values for l_i that minimises the sum of squares difference at each i can be found by expanding the squared term and differentiating with respect to l_i. Therefore $l_i = \text{avg}(t^{(i)}) - \text{avg}(p)$ minimises the sum at any given position, where the average over a string is defined as the mean value of its characters. This can clearly be calculated in linear time for $1 \leq i \leq n-m+1$. The l_i values can be used to calculate $\mathcal{L}_\delta$ in $O(n \log m)$ time using the FFT methods described earlier. It is an open question if shift-matching using the L_1 or L_∞ norms can be solved in $o(nm)$ time. An interesting subproblem that arises is how to compute the median of the values in each substring $t^{(i)}$ in $o(nm)$ time for $1 \leq i \leq n - m + 1$.

References

[1] K. Abrahamson. Generalized string matching. *SIAM journal on Computing*, 16(6):1039–1051, 1987.

[2] A. Amir. Private communication. 2004.

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

[4] R. Clifford, T. Crawford, C. Iliopoulos, and D. Meredith. String matching techniques for music analysis. In *String Algorithmics*, NATO book series. KCL Press, 2004.

[5] R. Cole and R. Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 592–601, 2002.

[6] R. Cole, R. Hariharan, and P. Indyk. Fast algorithms for subset matching and tree pattern matching. Preprint.

[7] R. Cole, C. Iliopoulos, T. Lecroq, W. Plandowski, and W. Rytter. On special families of morphisms related to δ-matching and don't care symbols. *Information Processing Letters*, 85(5):227–233, 2003.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[9] T. Crawford, C. S. Iliopoulos, and R. Raman. String-matching techniques for musical similarity and melodic recognition. In *Melodic Similarity: Concepts, Procedures, and Applications*, volume 11, pages 73–100. MIT-Press, 1998.

[10] M. Fischer and M. Paterson. String matching and other products. In R. Karp, editor, *Proceedings of the 7th SIAM-AMS Complexity of Computation*, pages 113–125, 1974.

[11] D. Gusfield. *Algorithms on strings, trees and sequences. Computer Science and Computational Biology*. Cambridge University Press, 1997.

[12] J. C. Schatzman. Accuracy of the discrete fourier transform and the fast fourier transform. *SIAM Journal of Scientific Computing*, 17(5):1150–1166, 1996.