String Matching with Gaps for Musical Melodic Recognition

Costas S. Iliopoulos[†] and Masahiro Kurokawa[‡]

[†] Algorithm Design Group, Dept of Computer Science,
King's College London, Strand, England, and
School of Computing, Curtin University of Technology,
Perth, Australia
[‡] Algorithm Design Group, Dept of Computer Science,
King's College London, Strand, England

e-mail: csi@dcs.kcl.ac.uk, kurokawa@dcs.kcl.ac.uk

Abstract. Here, we have designed and implemented algorithms for string matching with gaps for musical melodic recognition on polyphonic music using bit-wise operations. Music analysts are often concerned with finding occurrences of patterns (motifs), or repetitions of the same pattern, possibly with variations, in a score. An important example of flexibility required in score searching arises from the nature of polyphonic music. Within a certain time span each of the simultaneously-performed voices in a musical composition does not, typically, contain the same number of notes. So 'melodic events' occurring in one voice may be separated from their neighbours in a score by intervening events in other voices. Since we cannot generally rely on voice information being present in the score we need to allow for temporal 'gaps' between events in the matched pattern.

Key words: exact string matching, approximate string matching, gaps, pattern recognition, computer-assisted music analysis, bit-wise operation

1 Introduction

This paper focuses on a set of string pattern-matching problems that arise in musical analysis, and especially in musical information retrieval. Music analysts are often concerned with finding occurrences of patterns, or repetitions of the same pattern, possibly with variations, in a score, while computer scientists often have to perform similar tasks on strings (sequences of symbols from an alphabet). Many objects can be viewed as strings: a text file, for instance, is a sequence of characters from the ASCII alphabet; a DNA code is a sequence of characters from the alphabet A,C,G,T (representing the base proteins which constitute DNA). Similarly, a musical score can

[†] Partially supported by a Marie Curie fellowship, Wellcome and Royal Society grants.

be viewed (at one level) as a string: at a very rudimentary level, the alphabet could simply be the set of notes in the chromatic or diatonic notation, or the set of intervals that appear between notes (e.g. pitch may be represented as MIDI numbers and pitch intervals as number of semitones).

Monophonic music (that is, music in which a single note only sounds at any given time) lends itself well to a one-dimensional string matching approach, and efficient matching algorithms for single-line melody-retrieval have been applied with some success. The polyphonic situation (where several voices or instruments may be performing together, and any number of notes may be sounding at any given time) is more complex, however, because of the temporal interaction between non-simultaneous events in different voices. Where full knowledge about the voicing of the music data (in both the search-pattern and the target) is available, matching could be done by successive searches on each voice in turn. In many music-retrieval or analysis applications, especially where the data has been prepared by encoding a printed score in conventional musical notation, this is possible. But in the general case the data is likely to be imperfectly-specified in terms of its voicing, typically depending on how it is obtained: from audio, for example, even given perfect note-extraction, voicing information is likely to be derivable only approximately, if at all. Therefore, we need to allow for temporal gaps between musical events in the matched pattern.

When we consider the approximate version of this problem we do not require a perfect matching but a matching that is good enough to satisfy certain criteria. The problem of finding substrings of a text similar to a given pattern has been extensively studied in recent years because it has a variety of applications including file comparison, spelling correction, information retrieval, searching for similarities among biosequences and computerized music analysis. One of the most common variants of the approximate string matching problem is that of finding substrings that match the pattern with at most k-*differences*. In this case, k defines the approximation extent of the matching (the edit distance with respect to the three edit operations – *mismatch*, *insert*, *delete*). There is another type of approximate matching; δ-approximate matching. It is well known that a musical score can be represented as a string. This can be accomplished by defining the alphabet to be the set of notes in the chromatic or diatonic notation or the set of intervals that appear between notes. These algorithms can be easily used in the analysis of musical works in order to discover similarities between different musical entities that may lead to establishing a "characteristic signature" [CIR98].

In addition, efficient algorithms for computing approximate matching and repetitions of substrings are also used in molecular biology [FLSS93, KMGL88, MJ93] and particularly in DNA sequencing by hybridization, reconstruction of DNA sequences from known DNA fragments, in human organ and bone marrow transplantation as well as the determination of evolutionary trees among distinct species.

Because exact matching may not help us to find occurrences of a particular melody in a musical work due to the transformation of the particular melody throughout the whole musical work we are compelled to use approximate matching that can absorb, to some extent, this transformation and report the occurrences of this melody. The transformation in different occurrences of a particular melody throughout a musical play is translated into errors of different occurrences of a substring with respect to an initial pattern. Quantity δ defines the error margins of such an approximation.

In [CCIMP99], algorithms *Shift-And* and *Shift-Plus* were presented as efficient solutions to find all δ-occurrences of a given pattern in a text. The *Shift-And* algorithm is based on the constant time computation of different states for each symbol in the text by using bitwise techniques. Therefore, the overall complexity is linear to the size of the text. In [IK02], approximate distributed matching problem for polyphonic music is solved in linear time. We must also mention that it is possible to adapt efficient exact pattern matching algorithms to this kind of approximation. For example, in [CILP01] adaptations of the *Tuned-Boyer-Moore* [HS91] and the *Skip-Search* [CLP98] algorithm were presented.

The organization of the paper is as follows. Some definitions are given in section 2. In section 3, δ-occurrence with α-bounded gaps for monophonic music is considered. In section 4 we consider the problem of computing exact matching with α-bounded gaps for polyphonic music. Finally, we give some conclusions and future work in section 5.

2 Definitions

Let Σ be an alphabet. A string is defined as a sequence of zero or more symbols from Σ. The empty string, that is the string with zero symbols, is denoted by ε. The set of all strings over an alphabet Σ is denoted as Σ^*. A string x of length n is represented by the sequence $x_1, x_2, \ldots, x_n$, where $x_i \in \Sigma$ for $1 \leq i \leq n$. We call w a substring of string x if x is of the form uwv for $u, v \in \Sigma^*$. We also say that substring w occurs at position $|u| + 1$ of string x. The starting position of w in x is the position $|u| + 1$ while position $|u| + |w|$ is said to be the end position of w in x. A string w is a prefix of x if x is of the form wu and is a suffix if x is of the form uw.

We define as the concatenation of two strings x and y the string xy. The concatenations of k copies of a string x is denoted by x^k. Note that self-concatenations can result in strings of exponential size. For two strings $x = x_1, x_2, \ldots, x_n$ and $y = y_1, y_2, \ldots, y_m$ such that $x_{n-i+1}, \ldots, x_n = y_1 \ldots y_i$ for some $i \geq 1$, the string $x_1, \ldots, x_n, y_i, \ldots, y_m$ is the superposition of x and y. In this case we say that x and y overlap.

At this point, we are going to give formally the notion of error introduced in approximate string matching. Assume that δ and γ are integers. Two symbols a, b of alphabet Σ are said to be δ-*approximate*, denoted as $a =_\delta b$, if and only if $|a - b| \leq \delta$. We say that two strings x, y are δ-*approximate*, denoted as $x \stackrel{\delta}{=} y$ if and only if $|x| = |y|$ and $x =_\delta y$.

Two strings x, y are said to be γ-*approximate*, denoted as $x =_\gamma y$, if and only if $|x| = |y|$ and $\sum_{i=1}^{|x|} |x_i - y_i| < \gamma$. Furthermore, we say that two strings x, y are (δ, γ)-*approximate* if both conditions are satisfied.

The error in the first case (δ-approximate) is defined locally for each symbol in a string. In the second case (γ-approximate) the error is defined in a more global sense and allows us to distribute the error on the symbols unevenly.

3 δ-occurrence with α-bounded gaps for monophonic music

The problem of computing δ-occurrence with α-bounded gaps is formally defined as follows: given a string $t = t_1, \ldots, t_n$, a pattern $p = p_1, \ldots, p_m$ and integers α, δ, check whether there is a δ-occurrence of p in t with gaps whose sizes are bounded by constant α (Fig. 1).

The basic idea of the algorithm described in [CIPR00] is the computation of continuously increasing prefixes of pattern p in text t so that finally we compute the δ-occurrence of the whole pattern p. That is, the algorithm is an incremental procedure that is based on dynamic programming. The algorithm is shown in Fig. 2.

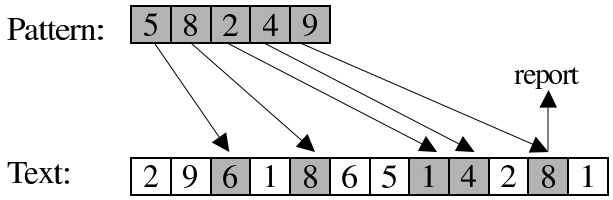


Figure 1: δ-occurrence with α-bounded gaps for $\delta = 1, \alpha = 2$

begin
  $D[0][0] \leftarrow 1$;
  for $i \leftarrow 1$ **to** m **do** $D[i][0] \leftarrow 0$;
  for $j \leftarrow 1$ **to** n **do** $D[0][j] \leftarrow j$;
  for $i \leftarrow 1$ **to** m **do**
    for $j \leftarrow 1$ **to** n **do**
      if $p[i] =_\delta t[j]$ **and** $j - D[i-1][j-1] \leq \alpha + 1$ **and** $D[i-1][j-1] > 0$
        then $D[i][j] \leftarrow j$;
      elseif $p[i] \neq_\delta t[j]$ **and** $j - D[i][j-1] < \alpha + 1$ **then** $D[i][j] \leftarrow D[i][j-1]$;
      else $D[i][j] \leftarrow 0$;
  for $j \leftarrow 0$ **to** n **do**
    if $D[m][j] > 0$ **then** OUTPUT(j);
end

Figure 2: Algorithm for δ-occurrence with α-bounded gaps

This algorithm will be adapted to the problem of finding a singular pattern in a singular text (monophonic music) without any major modifications. Fig. 3 shows 2 bars from Michael Niman's piece and a melody which listeners can easily cognize.

If we set the value $\alpha = 3$ (3 gaps allowed), the algorithm can find this melody in the score, while we have to set a large number of k (at least $k = 12$) to find it using k-difference approximate matching algorithms. The time complexity of this algorithm is $O(nm)$, where n is the number of the musical events in the score, which is equivalent to the number of notes in the score since this is monophonic music, and m is the length of the pattern. The running time is shown in Fig. 4.

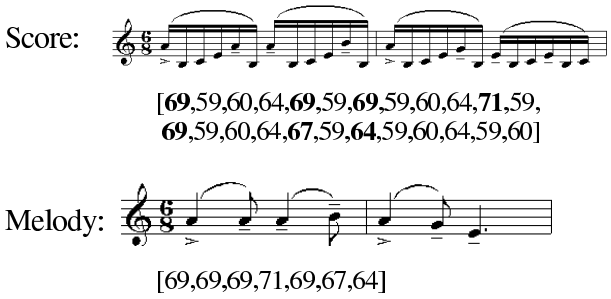


Figure 3: 2 bars from Michael Niman's piece and its melody. If $\alpha \geq 3$, this melody will be found.

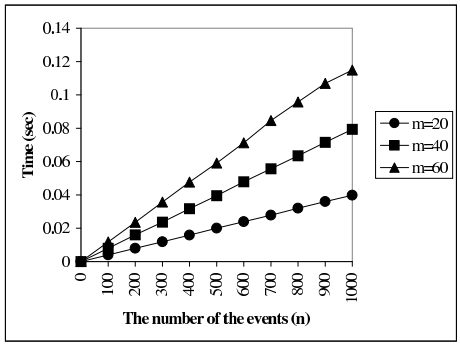


Figure 4: Running time of the algorithm "δ-occurrence with α-bounded gaps for monophonic music". Using a SUN Ultra Enterprise 300MHz running Solaris Unix.

4 Exact matching with α-bounded gaps for polyphonic music

We need to modify the algorithm in order to solve the problem in polyphonic music. Here, we will work on exact matching with α-bounded gaps for polyphonic music, and δ-occurrence will not be considered, as the adjacent pitch does not necessarily mean the most relevant note for a melody. Also, we will suppress the MIDI pitch numbers by dividing by 12 in order to find octave-displaced matches as well. Therefore, 'C' is '1', 'C#' and 'Db' are '2', 'D' is '3', and so on, and the size of alphabet $|\Sigma|$ will be 12.

We are going to use bit arrays and bit-wise operations to deal with several voices at once. Let $Tx[i]$ ($1 \leq i \leq m$, m is the length of a pattern) be a bit array of size $|\Sigma|$ for the position i of the pattern, and $Ty[j]$ ($1 \leq j \leq n$, n is the number of musical events in a plural text) be a bit array of size $|\Sigma|$ for the j-th musical event of the plural text. If $x[i]$ contains a note '8', then the 8th position of $Tx[i]$ will be 0, otherwise 1, where 0 represents 'match' and 1 represents 'mismatch'. Similarly, if $y[j]$ contains notes '3', '4' and '9', then the 3rd and 4th and 9th position of $Ty[j]$ will be 0, otherwise 1. These bit arrays will be used in the searching phase to check whether there is a match or not.

Fig. 5 shows the modified algorithm and the overall time complexity is $O(N+nm)$, where N is the total number of notes in the score, and n is the number of the musical events, and m is the length of the pattern, and Fig. 6 shows its running time. Fig. 7

```
Preprocessing
begin
    for j ← 1 to m do Tx[j] ← 2^σ − 1 − 2^x[j];
    for i ← 1 to n do
      Ty[i] ← 2^σ − 1;
      for each suppressed pitch p in y[i] do Ty[i] ← Ty[i] & (2^σ − 1 − 2^p);
end
Searching
begin
    D[0][0] ← 1;
    for i ← 1 to m do D[i][0] ← 0;
    for j ← 1 to n do D[0][j] ← j;
    for i ← 1 to m do
      for j ← 1 to n do
        if (Tx[i] | Ty[j]) = Tx[i] and j − D[i − 1][j − 1] ≤ α + 1
          and D[i − 1][j − 1] > 0 then D[i][j] ← j;
        elseif (Tx[i] | Ty[j]) ≠ Tx[i] and j − D[i][j − 1] < α + 1
          then D[i][j] ← D[i][j − 1];
        else D[i][j] ← 0;
    for j ← 0 to n do
      if D[m][j] > 0 then OUTPUT(j);
end
```

Figure 5: Modified algorithm for polyphonic music

and Fig. 9 show examples of the preprocessing phase for 1 bar from Mozart's piano sonata and Debussy's Clair de Lune, respectively, and Fig. 8 and Fig. 10 show their searching phases.

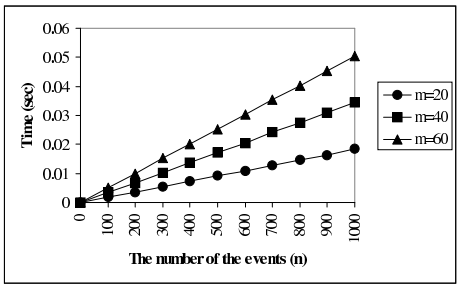


Figure 6: Running time of the modified algorithm for polyphonic music ($N = 4n$). Using a SUN Ultra Enterprise 300MHz running Solaris Unix.

5 Conclusion and further work

Approximate (δ-occurrence) string matching with gaps for monophonic music is solved in $O(nm)$ time, where n is the number of musical events (which is equivalent to the number of notes in a score for monophonic music), and m is the length of a pattern. Exact string matching with gaps for polyphonic music (a plural text and a singular

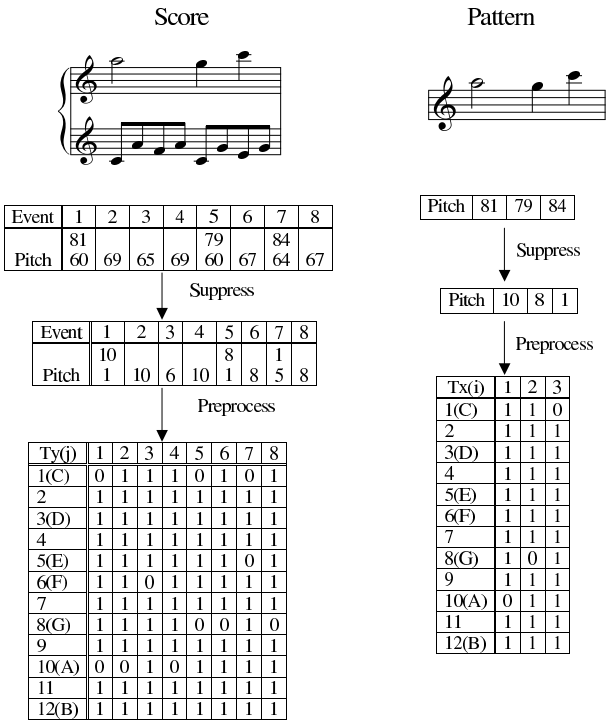


Figure 7: Preprocessing phase for 1 bar from a Mozart's piano sonata and a pattern. $(N = 11, n = 8, m = 3)$

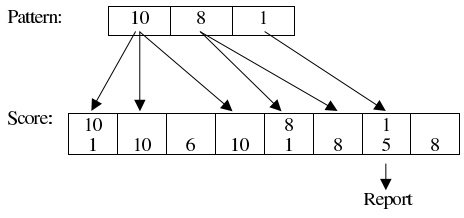
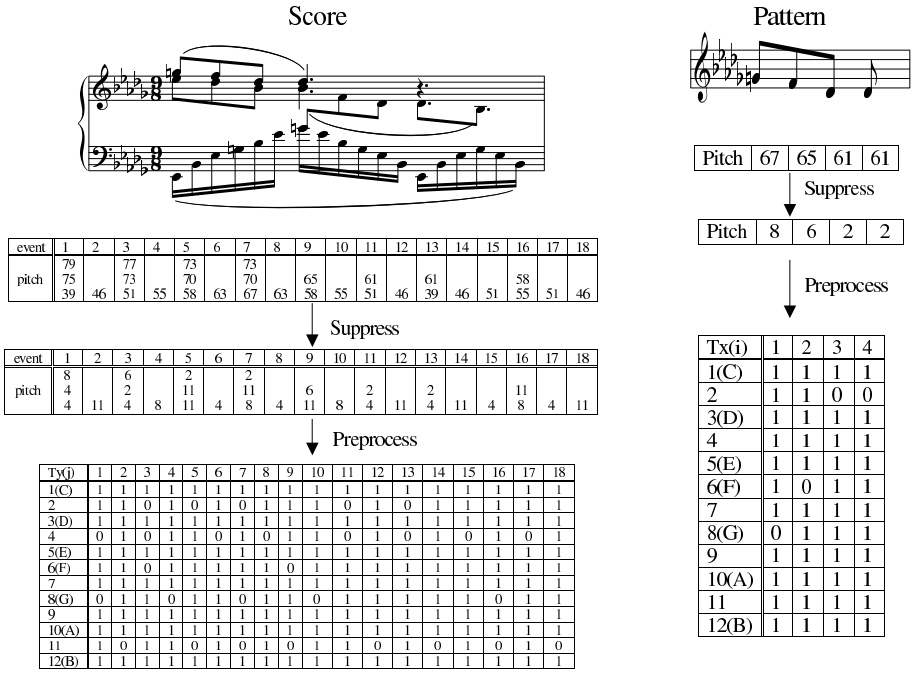


Figure 8: Searching phase using bit-wise operations for 1 bar from the Mozart's piano sonata and the patten. $(\alpha = 3)$



event	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
pitch	79 75 39	46	77 73 51	55	73 70 58	63	73 70 67	63	65 58	55	61 51	46	61 39	46	51	58 55	51	46

event	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
pitch	8 4 4	11	6 2 4	8	2 11 11	4	2 11 8	4	6 11	8	2 4	11	2 4	11	4	11 8	4	11

Ty(j)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1(C)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	0	1	0	1	0	1	1	1	0	1	0	1	1	1	1	1
3(D)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	0	1	0	1	1	0	1	0	1	1	0	1	0	1	0	1	0	1
5(E)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6(F)	1	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8(G)	0	1	1	0	1	1	0	1	1	0	1	1	1	1	1	0	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10(A)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
11	1	0	1	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
12(B)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Pitch	67	65	61	61

Suppress

Pitch	8	6	2	2

Preprocess

Tx(i)	1	2	3	4
1(C)	1	1	1	1
2	1	1	0	0
3(D)	1	1	1	1
4	1	1	1	1
5(E)	1	1	1	1
6(F)	1	0	1	1
7	1	1	1	1
8(G)	0	1	1	1
9	1	1	1	1
10(A)	1	1	1	1
11	1	1	1	1
12(B)	1	1	1	1

Figure 9: Preprocessing phase for 1 bar from Clair de Lune and a pattern. ($N = 30, n = 18, m = 4$)

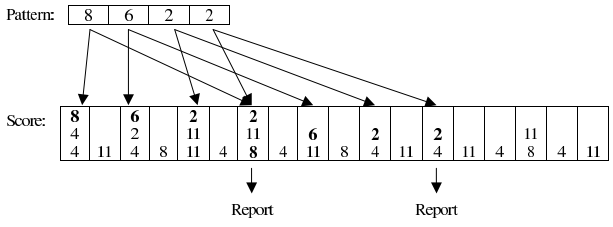


Figure 10: Searching phase using bit-wise operations for 1 bar from Clair de Lune and the patten. ($\alpha = 1$)

pattern) is solved in $O(N + nm)$ time, where N is the total number of notes in a score, and n is the number of musical events ($n \leq N$), and m is the length of a pattern. Using the same technique, exact string matching problem for a plural text and a plural pattern will be solved in $O(N + M + nm)$ time, where M is the total number of notes in the plural pattern. However, we have not solved approximate string matching with gaps for polyphonic music, because "small δ" does not really mean "more relevant" in music. In this particular sense, k-difference algorithms could be more useful, although it is inevitable to have large k and many false matches. We need to design an efficient algorithm for this problem.

References

[CCIMP99] Cambouropoulos, E., Crochemore, M., Iliopoulos, C.S., Mouchard, L., Pinzon, Y.J.: Algorithms for computing approximate repetitions in musical sequences. Proceedings of the 10^{th} Australasian Workshop on Combinatorial Algorithms (AWOCA'99), R. Raman and J. Simpson (editors), Curtin University of Technology, Perth, Western Australia, 129-144.

[CILP01] Crochemore, M., Iliopoulos, C.S., Lecroq, T., Pinzon, Y.J.: Approximate String Matching in Musical Sequences. Proceedings of the Prague Stringology Conference (PSC'01), M. Balik and M. Simanek (editors), Czech Technical University, Collaborative Report DC-2001-06, Prague, Czech Republic, 26-36.

[CIMRTT01] Crochemore, M., Iliopoulos, C.S., Makris, C., Rytter, W., Tsakalidis, A., Tsichlas, K.: Approximate String Matching with Gaps. Nordic Journal of Computing 9, 54-65.

[CIPR00] Crochemore, M., Iliopoulos, C.S., Pinzon, Y.J., Rytter, W.: Finding Motifs with Gaps. Proceedings of the International Symposium on Music Information Retrieval (ISMIR'00), Plymouth, USA, 306-317.

[CIR98] Crawford, T., Iliopoulos, C.S., Raman, R.: String Matching Techniques for Musical Similarity and Melodic Recognition. Computing in Musicology, Vol.11, 73-100.

[CLP98] Charras, C., Lecroq, T., Pehoushek, J.D.: A very fast string matching algorithm for small alphabets and long patterns. Proceedings of the 9^{th} Annual Symposium on Combinatorial Pattern Matching (CPM'98), M. Farach-Colton (editor), number 1448 in Lecture Notes in Computer Science, Piscataway, NJ, Springer-Verlag, Berlin, 55-64.

[FLSS93] Fischetti, V., Landau, G., Schmidt, J., Sellers, P.: Identifying periodic occurrences of a template with applications to protein structure. Information Processing Letters, 45, 11-18.

[HS91] Hume, A., Sunday, D.M.: Fast String Searching. Software-Practice and Experience, 21(11), 1221-1248.

[IK02] Iliopoulos, C.S., Kurokawa, M.: Distributed Matching Problem for Musical Melodic Recognition. Proceedings of the 2002 symposium on AI and Creativity in Arts and Science (AISB'02), A. Cardoso and G. Wiggins (editors), London, 49-56.

[KMGL88] Karlin, S., Morris, M., Ghandour, G., Leung, M.Y.: Efficient Algorithms for molecular sequences analysis. Proc. Natl. Acad. Sci., USA, 85, 841-845.

[LMW02] Lemstrm, K., Meredith, D., Wiggins, G.A.: A geometric approach to computing repeated patterns in polyphonic music. Document submitted to UK Patent office, application number GB 0200203.8.

[MJ93] Milosavljevic, A., Jurka, J.: Discovering simple DNA sequences by the algorithmic significance method. Comput. Appl. Biosci., 9(4), 407-411.