where DR takes great relevance. In software repositories, a frequent task for developers is to
find where an object is mentioned in their source code files; for instance with user function
calls. Hence, pieces of source code are treated as documents and a DR framework is built
as a part of the development environment. In music collections, we also find various tasks
related to Music Information Retrieval (MIR). In MIDI sequence analysis\(^1\), one of the most
relevant concerns is to locate the occurrences of a theme in a piece of music. The theme can
be a melody or a sequence of notes called a musical pattern. The music is simply a song file in
MIDI format, a set of symbolically encoded notes that form the musical sequences represent-
ing each document, which is obtained from a digital-to-symbolic conversion of audio data.
Bioinformatics is another research area where DR solutions are often sought \([8]\). Advances
in DNA sequencing have produced databases of thousands of human genomes, which implies
additional problems related to data storage and how to retrieve pieces of sequences from
it. The challenge is again to build small representations for these big biological sequences
and to offer methods to carry out efficient searches on them. These sequences must be well-
compressed in data structures that allow us to filter biological documents without the need
to decompress the whole representation. In DNA sequencing, a popular problem is to list all
the genes where a DNA marker appears, where the sequence is composed only of base pairs
from the set \{A, C, T, G\}. Another frequent task, related with protein sequences, is to find
all the proteins where an amino acid sequence appears frequently.

The Inverted Index \([5]\) is the most widely used data structure to solve DR problems when
the texts can be split into words. It is very similar to a book index, where for a set of
pre-determined words, we store for each word a list of all the documents that contain it. In
order to answer a DR query, where queries are sets of words, the inverted index finds the lists
of documents where each query word appears. After that, it must solve operations for sets
such as union, intersection or differences between the retrieved lists. The type of operation
depends on the problem to solve, and other variables are included to build the final answer,
such as scores for each document or weighting documents according to query word frequen-
cies. However, this approach is not easily applicable to human languages such as Chinese,
Korean, Thai, and other Asian languages, because these texts have no delimiters to mark
word boundaries. The same problem happens with agglutinating languages as Hungarian,
Turkish or Finnish, where sentences are concatenated into words. Another example is the
biological sequence analysis on DNA sequences, where as we mentioned the alphabet is a set
composed of only four characters without any delimiter. There are also many applications
where inverted indexes cannot be applied because the concept of word does not exist: source
and binary codes in software repositories, MIDI files, or any other multimedia database.
Consequently, the indexes of general string collections must be more general than inverted
indexes.

In this given context, an elementary and closely related problem (widely studied in text
indexing \([88]\)) is Pattern Matching. It aims to locate all the positions where a given arbitrary
string, called the search pattern, occurs in a text given beforehand. The Suffix Tree (ST)
\([116]\) is the most popular data structure used to solve this problem in optimal time and
linear space. For a given text \(T_{1..n}\) and a search pattern \(p_{1..m}\) that matches \(\text{occ}\) times in \(T\),

\(^1\)MIDI is an acronym for the Musical Instrument Digital Interface, and has taken on multiple meanings as
the data in a Standard MIDI File (SMF). That standard describes the format designed to work with MIDI
hardware devices \([110]\).