

Generic Subsequence Matching Framework: Modularity, Flexibility, Efficiency

David Novak, Petr Volny, and Pavel Zezula

Masaryk University, Brno, Czech Republic
[david.novak | xvolny1 | zezula]@fi.muni.cz

Abstract. Subsequence matching has appeared to be an ideal approach for solving many problems related to the fields of data mining and similarity retrieval. It has been shown that almost any data class (audio, image, biometrics, signals) is or can be represented by some kind of time series or string of symbols, which can be seen as an input for various subsequence matching approaches. The variety of data types, specific tasks and their partial or full solutions is so wide that the choice, implementation and parametrization of a suitable solution for a given task might be complicated and time-consuming; a possibly fruitful combination of fragments from different research areas may not be obvious nor easy to realize. The leading authors of this field also mention the implementation bias that makes difficult a proper comparison of competing approaches. Therefore we present a new generic Subsequence Matching Framework (SMF) that tries to overcome the aforementioned problems by a uniform frame that simplifies and speeds up the design, development and evaluation of subsequence matching related systems. We identify several relatively separate subtasks solved differently over the literature and SMF enables to combine them in straightforward manner achieving new quality and efficiency. This framework can be used in many application domains and its components can be reused effectively. Its strictly modular architecture and openness enables also involvement of efficient solutions from different fields, for instance efficient metric-based indexes. This is an extended version of a paper published on DEXA 2012.

1 Introduction

Majority of the data being produced in current digital era is in the form of *time series* or can be transformed into sequences of numbers. This concept is very natural and ubiquitous: audio signals, various biometric data, image features, economic data, etc. are often viewed as time series and need to be also organized and searched in this way.

One of the key research issues drawing a lot of attention during the last two decades is the *subsequence matching problem*, which can be basically formulated as follows: Given a query sequence, find the best-matching subsequence from the sequences in the database. Depending on the specific data and application, this general problem has many variants – query sequences of fixed or variable

size, data-specific definition of sequence matching, requirement of dynamic time warping, etc. Therefore, the effort in this research area resulted in many approaches and techniques – both, very general and those focusing on a specific fragment of this complex problem.

The leading authors in this field identified two main problems that limit the comparability and cooperation potential of various approaches: the *data bias* (algorithms are often evaluated on heterogeneous datasets) and the *implementation bias* (the implementation of the specific technique can strongly influence experiment results) [1]. The effort to overcome the data bias is expressed by founding a common set of data collections [2] which is publicly available and that should be used by any consequent research in this area. However, the implementation bias lingers, which also obstructs a straightforward combination of compatible approaches whose interconnection could be very efficient.

Analysis of this situation brought us to conclusion that there is a need for a unified environment for developing, prototyping, testing, and combination of subsequence matching approaches. In this paper we propose such generic subsequence matching framework (SMF), namely:

- we overview and decompose the state-of-the-art approaches and techniques for subsequence matching and we identify several common sub-problems that these approaches deal with in various ways (Section 2);
- we propose general architecture of SMF that should fulfill our targets and we describe our implementation of the framework (Section 3); power of SMF is demonstrated by elegant realization of several variants of fundamental subsequence matching algorithms;
- we describe real applications with different requirements for subsequence matching that can be simply implemented with the aid of our framework (Section 4).

The paper concludes in Section 5 by future directions that cover possible performance boost enabled by a straightforward cooperation of our framework with advanced distance-based indexing and searching technologies [3,4].

2 Time Series Processing Overview

The field opening paper by Faloutsos et al. [5] introduced a subsequence matching application model that has been used ever since only with smaller modifications. The model can be summarized in the following four steps that should be adopted by a subsequence matching application:

- slicing** of the time series sequences (both data and query) into shorter subsequences (of a fixed length),
- transforming** each subsequence into lower dimension,
- indexing** the subsequences in a multi-dimensional index structure,
- searching** in the index with a distance measure that obeys the lower bounding lemma on the transformed data.

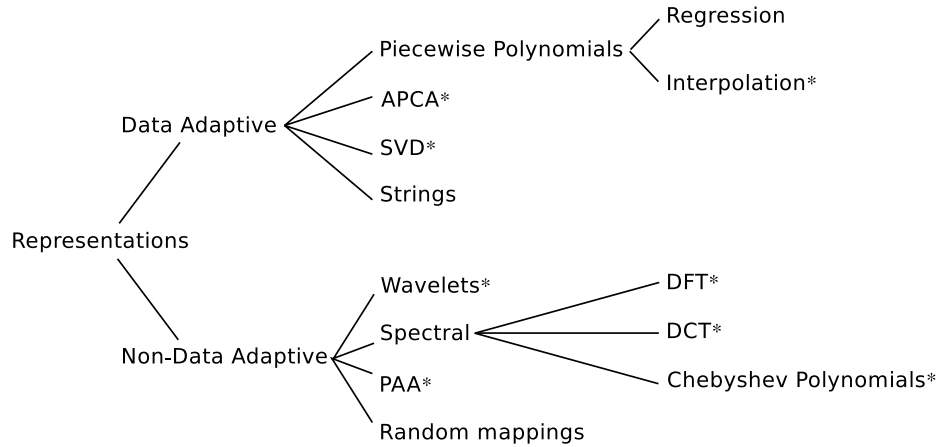


Fig. 1. Basic hierarchy of representations. Those with asterisk after the name allows lower bounding.

In the original work [5], this approach was demonstrated on a subsequence matching algorithm that used the *sliding window* approach to slice the indexed data and *disjoint window* for the query. The Discrete Fourier Transformation (DFT) was used for dimensionality reduction and the data was indexed using the minimum bounding rectangles in R-tree [6]. The Euclidean distance was used for searching since it satisfies the lower bounding lemma on data transformed by DFT.

Many works followed this model contributing to it by using other time series representations, more sophisticated distance functions and corresponding lower bounding mechanisms. In the rest of this section, we provide a brief overview of the key approaches in this field.

2.1 Data Representation

The way in which the data is represented may be crucial for both efficiency and effectiveness of the whole system. Naturally, one can work directly with the original data, but this *raw data* is typically very large and it would require great computing infrastructure to process such data fast enough. Therefore, a lot of effort has been aimed at finding the best possible approximation of the time series that would reduce the dimensionality of the data on one hand and preserve the main features of the data on the other.

Figure 1 depicts the most significant approaches used for reducing time series data representation: *Discrete Fourier Transformation* (DFT) [5], *Discrete Cosine Transform* (DCT) [7], *Chebyshev Polynomials* (CHEB) [8], *Discrete Wavelet Transform* (DWT) [9], *Piecewise Aggregate Approximation* (PAA) [10], *Single Value Decomposition* (SVD) [7], *Adaptive Piecewise Constant Approximation* (APCA) [11]. On the higher level, we distinguish two groups of approaches: *data*

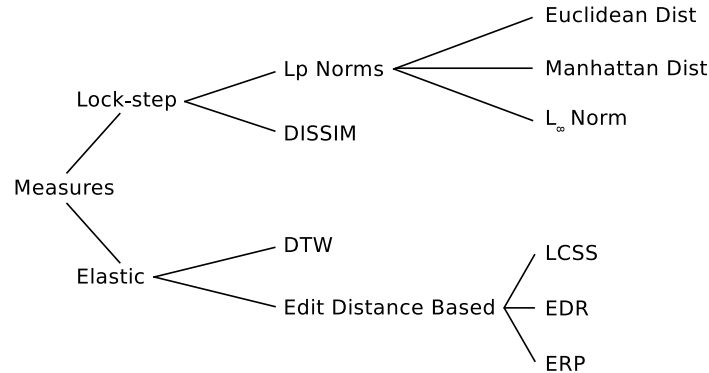


Fig. 2. Basic hierarchy of distance functions.

adaptive and *non-data adaptive*. An important feature for any representation is the ability to be indexed because proper indexing can dramatically improve performance of the whole retrieval process. Members of the latter group sometimes struggle with indexing [12]. Another desirable property of the data representation is the possibility to perform *lower bounding*, which can again improve the performance of the subsequence matching process.

2.2 Distance computation

The notion of similarity between two sequences is typically expressed by a *distance measure* (distance function). The most frequently used functions are depicted in Figure 2. The lock-step measures, like Euclidean distance, are usually relatively cheap to compute but they lack the robustness against even the basic data transformations. On the other hand, more sophisticated dynamic programming methods like Dynamic Time Warping (DTW) allow shifts on the time axis and serve better to applications like speech recognition or query by humming. It is also important to pair the data representation and the distance function wisely in order to satisfy the lower bounding lemma [5]. Distance functions can also differ by their input domain. Some functions are defined on continuous domains (real numbers) and some work on strings (sequences of symbols from a finite alphabet). The most common distance measures are: *Euclidean Distance* (ED) [5], *Dynamic Time Warping* (DTW) [13,14,15], *Edit Distance with Real Penalty* (ERP) [16], *Edit Distance on Real Sequence* (EDR) [17], *Longest Common Subsequence* (LCSS) [18].

2.3 Subsequence Matching Approaches

In order to build a subsequence matching application, number of questions has to be answered: What kind of data will be used and what is its dimensionality? What is the volume of the data? Do we need the warping ability? Are approximate answers acceptable? etc. On the basis of the answers, one can make proper

design decisions like what representation and distance measure to use, which storage index to employ, etc.

The above mentioned work by Faloutsos et al. [5] encouraged many new approaches that followed his work. Moon et al. [19] suggested dual approach for slicing and indexing sequences to the one used in [5]. This *DualMatch* uses the sliding windows for queries and disjoint windows for data sequences to reduce the number of windows that are indexed. DualMatch was followed by the generalization of windows creation method called GeneralMatch [20]. Another significant leap forward was made by the effort of Keogh et al. in their work about exact indexing of Dynamic Time Warping [15]. They introduced a similarity measure that is relatively easy to compute and it lower-bounds the expensive DTW function. This approach was further enhanced by improving I/O part of the subsequence matching process using Deferred Group Subsequence Retrieval introduced in [21]. Moreover, many new distance functions and data representations were introduced as we outlined in the previous sections.

If we focus on the performance side of the system, we have to employ enhancements like indexing, lower bounding, window size optimization, reducing I/O operations or approximate queries. Lots of approaches for building subsequence matching applications often use the very same techniques for solving common sub-tasks included in the whole retrieval process and changes only some parts with some novel approach. This leads to implementing the same parts of the process like DFT or DWT repeatedly which leads to the phenomenon of the *implementation bias* [1]. The modern subsequence matching approaches [21,15] employ many smaller tasks in the retrieval process that solve sub-problems like optimizing I/O operations. Implementations of routines that solve such sub-problems should be reusable and employable in similar approaches. This led us to think about the whole subsequence matching process as a chain of sub-tasks, each solving a small part of the problem. We have observed that many of the published approaches fit into this model and their novelty is often only in reordering, changing or adding new subtask implementation into the chain.

3 Subsequence Matching Framework

In this section, we describe the general Subsequence Matching Framework (SMF) that is currently available under GPL license at <http://mufin.fi.muni.cz/smf/>. The framework can be perceived on the following two levels that should, naturally, coincide:

- on the *conceptual level*, the framework is composed of mutually cooperating modules, each of which solves a specific sub-task, and these modules are cooperating within specific subsequence matching algorithms;
- on the *implementation level*, the framework defines the functionality of individual module types and their communication interfaces; a subsequence matching algorithm is then implemented as a *skeleton* that combines modules in a specific way and this skeleton can be filled with actual module implementations.

Table 1. Notation used throughout this paper.

Symbol	Definition
$S[k]$	the k -th value of the sequence S
$S[i : j]$	subsequence of S from $S[i]$ to $S[j]$, inclusive
$S.len$	the length of sequence S
$S.id$	the unique identifier of sequence S
$S'.pid$	if S' is subsequence of S then $S'.pid = S.id$
$S'.offset$	if $S' = S[i : j]$ then $S'.offset = i$ and $S'.len = j - i + 1$
$D(Q, S)$	distance between two sequences Q and S

In Section 3.1, we describe the common sub-problems (sub-tasks) that we identified in the field and we define corresponding types of modules (conceptual level). Further, in Section 3.2, we justify our approach by describing fundamental sub-sequence algorithms in terms of our modules and we present a straightforward implementation of these algorithms within SMF. Section 3.3 is devoted to details about implementation of the framework including an example of a configuration file by which one can create a brand new algorithm only by exchanging specific modules in a text configuration file.

The key term in the whole framework is, naturally, a *sequence*. As we want to keep the framework as general as possible, we do not lay practically any restrictions on the components of the sequence – it can be integers, real numbers, vectors of numbers, or any more sophisticated structures. The sequence similarity functions are defined relatively independently of specific sequence type (see Section 3.3). In the following, we will use the notation summarized in Table 1.

3.1 Common Sub-problems: Modules in SMF

Studying the field of subsequence matching, we identified several common sub-problems addressed by a number of approaches in some sense. Specifically, we can see the following sub-tasks that correspond to isolated modules in our framework.

Data Representation (Data Transformer Module) The raw data sequences entering an application are often transformed into other representation which can be motivated either by simple dimensionality reduction (DFT, DWT, SVD, PAA) [5,9,7,10] or also by extracting some important characteristics that should improve the effectiveness of the retrieval [22] (see Section 2.1 for details). In either case, the general task can be defined simply as follows: *Transform given sequence S into another sequence S'* . We will use the symbol in Figure 3 (a) for this *data transformer* module. The following table summarizes information about this module and gives a few examples of specific approaches implementing this functionality.

data transformer	transform sequence S into sequence S'
DFT	apply the DFT on a sequence of real numbers S [5]
PAA	apply the PAA on a sequence of real numbers S [10]
Landmarks	extract <i>landmarks</i> from a sequence S [22]

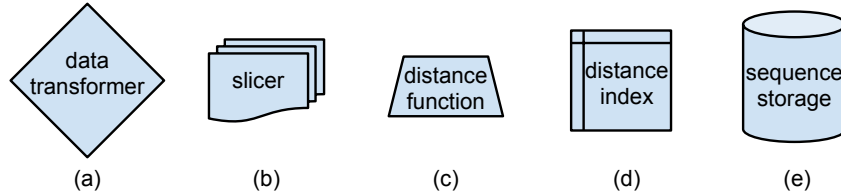


Fig. 3. Types of SMF modules and their notation.

Windows and Subwindows (Slicer Module) Majority of the subsequence matching approaches partitions the data and/or query sequences into subsequences of, typically, fixed length (windows) [5,19,20,21]. Again, this task can be isolated, well defined, and the implementation can be reused in many variants of subsequence matching algorithms. Partitioning a sequence S , each resulting subsequence $S' = S[i : j]$ has $S'.pid = S.id$, $S'.offset = i$, and $S'.len = j - i + 1$. The module will be denoted as in Figure 3 (b) and its description and specific examples are as follows:

sequence slicer	partition S into list of subsequences S'_1, \dots, S'_n
disjoint slicer	partition S disjointly into subsequences of length w [5]
sliding slicer	use sliding window of size w to partition S [5]

Distance Functions (Distance Module) As analyzed in Section 2.2, there emerged a high number of specific distance functions D that can be evaluated between two sequences S and T . The intention of SMF is to partially separate the distance functions from the data and to use the specific distance function as a parameter of the algorithm (see Section 3.3 for details on implementation of the this independence). Of course, it is the matter of configuration to use appropriate function for respective data type, e.g. to preserve the lower bounding property. The distance functions symbol is in Figure 3 (c) and it can be summarized as follows:

distance function	evaluate dissimilarity of sequences S, T
L_p metrics	evaluate distance L_p on equally long number sequences
DTW	use DTW on any pair of number sequences S, T [13]
ERP	calculate Edit distance with Real Penalty on S, T [16]
LB.PAA, LB_Keogh	measures that lower bound the DTW [15]

Efficient Indexing (Distance Index Module) An efficient subsequence-matching algorithm typically employs an index to efficiently evaluate distance-based queries on the stored (sub-)sequences using the query-by-example paradigm (QBE). Again, we see the choice of the specific index as a relatively separate component of the whole algorithm and thus as an exchangeable module. Also, we see a space for improvement in boosting the efficiency of this component in future. We denote this module as in Figure 3 (d):

distance index	evaluate efficiently distance-based QBE queries
R-Tree family	index sequences as n -dimensional spatial data
<i>i</i> SAX tree	use a symbolic representation of the sequences [23,24]
metric indexes	index and search the data according to mutual distances [3]

Efficient Aligning (Sequence Storage Module) The approaches that use sequence slicing typically also need to store the original whole sequences. The slice index (for window size w) returns a set of candidate subsequences S' , $S'.len = w$ each matching some query subsequence Q' such that $Q'.len = w$. If the query sequence Q is actually longer than w , the subsequent task is to align Q to corresponding subsequence $S[i : (i + Q.len - 1)]$ where $i = S'.offset - Q'.offset$ and $S.id = S'.pid$. To do this aligning for each S' in the candidate set may be very demanding. For smaller datasets, this can be done in memory with no special treatment, but more advanced approaches are profitable on disk [21]. We will call this module *sequence storage* (Figure 3 (e)) and it is specified as follows:

sequence storage	store sequences S and return $S[i : j]$ for given $S.id$
hash map	basic hash map evaluating queries one by one
deferred retrieval	deferred group sequence retrieval (I/O efficient) [21]

3.2 Subsequence Matching Strategies in SMF

Staying at the conceptual level, let us have a look at the whole subsequence matching algorithms and their composition from individual modules introduced above. As an example, we take again the fundamental algorithm [5] for general subsequence matching of queries Q , $Q.len \geq w$ for an established window size w . The schema of a slight modification of this algorithm is in Figure 4. The solid lines correspond to data insertion and the dash lines (with italic labels) correspond to the query processing.

A data sequence S is first partitioned by the sliding window approach (*sliding slicer* module) into slices $S' = S[i : (i + w - 1)]$, these are transformed by Discrete Fourier Transformation (*data transformer* module DFT), and the Minimum Bounding Rectangles (MBR) of these transformed slices are stored in an R*-tree storage (*distance index* module); the original sequences S is also stored (*whole sequence storage* module). Processing a subsequence query, the query sequence Q is partitioned using the *disjoint slicer* module, each slice $Q' = Q[i : (i + w - 1)]$ is transformed by DFT and it is searched within the slice index (using L_2 distance or a simple binary function *intersect*). For each of the returned candidate subsequences S' , a query-corresponding alignment $S[i : (i + Q.len - 1)]$ is retrieved from the *whole storage* (see above for details) and the candidate set is refined using L_2 distance $D(Q, S[i : (i + Q.len - 1)])$.

Preserving the skeleton of an algorithm (module types and their cooperation), one can substitute individual modules with other compatible modules obtaining a different processing efficiency or even a fundamentally different algorithm. For instance, swapping the sliding and disjoint slicer modules practically results in the DualMatch approach [19]. Exchanging the R*-tree index for a metric-based

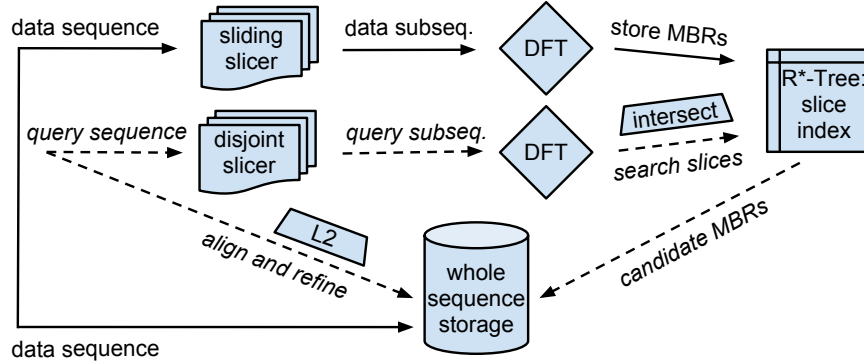


Fig. 4. Schema of the fundamental subsequence matching algorithm [5].

index like the M-Index [4] could possibly improve the efficiency for larger dimensionality of the stored slices (especially if the M-Index could use approximate evaluation strategy). In general, a fast and straightforward alternation of modules is very helpful when seeking the best solution for a particular task and data collection and tuning the performance of this solution.

3.3 Implementation of SMF

The SMF was not implemented from scratch but with the aid of framework MESSIF [25]. The MESSIF is a collection of Java packages supporting mainly development of metric-based search approaches. From MESSIF, the SMF uses especially the following functionality:

- encapsulation of the concept of data objects and distances,
- implementation of the queries and query evaluation process,
- distance based indexes (building, querying),
- configuration and management of the algorithm via text *config files*.

Data Independence The *sequence* is in SMF handled very generally; it is defined as an interface which requires that each specific sequence type (e.g. a simple float sequence) must, among other, specify the distance between two sequence components $d(S[i], S'[j])$. For number sequences, this distances could be, naturally, absolute value of differences $d(S[i], S'[j]) = |S[i] - S'[j]|$, but one can imagine complex sequence components, for instance vectors where d could be an L_p metric. Implementation of a sequence distance $D(S, S')$ (for instance, DTW) then treats S and S' only as general sequences with component distance d and, thus, this implementation can be independent of specific sequence type.

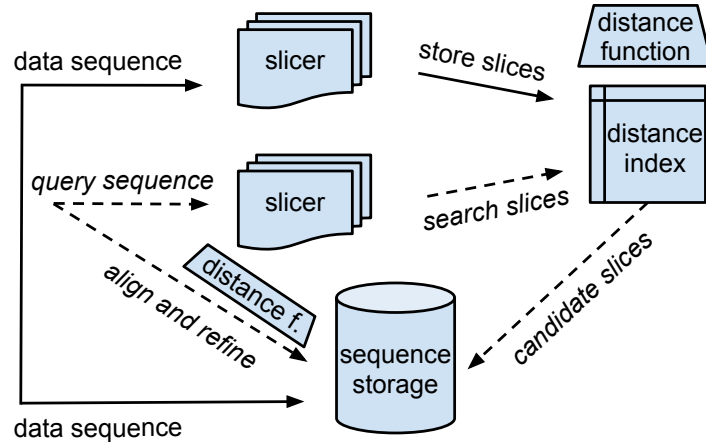


Fig. 5. Skeleton of `VariableQueryAlgorithm`, a simple subsequence matching algorithm with variable length query.

Module Implementation The SMF module types specified in Section 3.1 are typically implemented as Java interfaces and the specific modules as Java classes. The interface specifies prototypes of the methods (inputs and outputs) that the specific module must provide.

Algorithm Implementation Realizing a specific algorithm, one must implement its skeleton – module types, their connections, and all algorithm-specific operations. This skeleton is compiled within the SMF package together with all the interfaces and module implementations. Then, the algorithm is instantiated only by means of a text configuration file – module types required by the algorithm skeleton are filled by specific modules.

Let us describe this principle on an example of a simple algorithm for general subsequence matching with variable query length – see Figure 5 for the schema of the skeleton. This schema uses two *slicer* modules, one *distance index* with a *distance function*, and a *sequence storage* for the whole sequences (again, with a *distance function*).

In order to run a specific algorithm, we have to instantiate these module types by specific modules. Figure 6 shows a part of a SMF configuration file that starts such algorithm. The syntax of these files is taken from the MESSIF framework and it is quite intuitive. On the first two lines, the *sliding slicer* module is created by an action called `namedInstanceAdd` which creates an instance `slidingSlicer` of class `smf.modules.slicer.SlidingSlicer` (with parameter w); the *disjoint slicer* is created accordingly. Then, the instance of *distance index* is created; it is a self-standing *algorithm*, namely a metric index M-Index [4] (we assume that

```

slidingSlicer = namedInstanceAdd
slidingSlicer.param.1 = smf.modules.slicer.SlidingSlicer(<w>)

disjointSlicer = namedInstanceAdd
disjointSlicer.param.1 = smf.modules.slicer.DisjointSlicer(<w>)

index = namedInstanceAdd
index.param.1 = smf.modules.index
                    .ApproximateAlgorithmDistanceIndex(mIndex)

seqStorage = namedInstanceAdd
seqStorage.param.1 = smf.modules.seqstorage.MemorySequenceStorage()

startSearchAlg = algorithmStart
startSearchAlg.param.1 = smf.algorithms.VariableQueryAlgorithm
startSearchAlg.param.2 = smf.sequence.impl.SequenceFloatL2
startSearchAlg.param.3 = seqStorage
startSearchAlg.param.4 = index
startSearchAlg.param.5 = slidingSlicer
startSearchAlg.param.6 = disjointSlicer
startSearchAlg.param.7 = <w>

```

Fig. 6. Example of a SMF configuration file

the instance `mIndex` has been already created). In this example, the *sequence storage* is instantiated as a simple memory storage (`seqStorage`).

Finally, the actual `VariableQueryAlgorithm` is started passing the created module instances as parameters to the skeleton. The `param.2` of this action specifies that this particular algorithm instance requires sequences of floating point numbers and will compare them by Euclidean distance. Such SMF configuration files are supported directly by the MESSIF framework that enables an efficient management of the running algorithm.

4 Use Cases

One of the motivations for building the SMF framework were several applications that all need subsequence matching but are relatively heterogeneous. Let us demonstrate three such applications that differ in data type, distance function, query specification, and requirements for indexing efficiency; all of them can be straightforwardly implemented using SMF.

4.1 General Subsequence Matching System

The first demonstration is a general subsequence matching system for queries with variable length. The demo is built on a small collection of simple time-series data and it uses the algorithm described in Section 3.3 (Figures 5 and 6). Figure 7

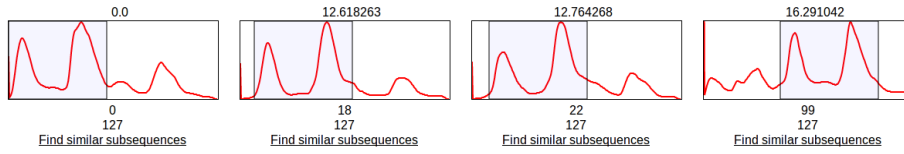


Fig. 7. General subsequence matching system with variable query length.

shows screenshot of the GUI of this demo: The user can specify a subsequence (offset and width) of the query sequence and after clicking “Find similar subsequences”, the most similar ones (according to Euclidean distance) are located (their distances are above the answer series). The demo is publicly available at <http://mufin.fi.muni.cz/subseq/>. The characteristics of this simple demo can be summarized as follows:

data type	series of real numbers
similarity measure	Euclidean distance
query type	subsequence query with variable length (longer than w)
requirements	no additional requirements

4.2 Gait Recognition

The biometrics form a wide application area that often manages data in the form of sequences. Among others, recognition of people according to their *gait* characteristics is currently a on the increase. There are several fundamental approaches to this problem and one of them works directly with 3D coordinates of anatomical landmarks of human body. The feasibility of this approach is growing as the hardware that can extract such 3D positions is more and more mature and available [26]. The left image in Figure 8 sketches the principle. The trajectories are further processed and the development of mutual distances between various anatomical landmarks is studied [27] – see time-series in the right part of Figure 8. Currently, the SMF is used for research in this area:

data type	series of real numbers or of number vectors
similarity measure	various L_p metrics, DTW, special measures
query type	subsequence query with fixed or variable length
requirements	data and similarity flexibility, high performance in future

4.3 Unsupervised Spoken Term Detection

The Automatic Speech Recognition (ASR) is an extremely important area for human-computer interaction and the fundamental problem in this field is Spoken Term Detection (STD). Besides well-developed complex approaches based on

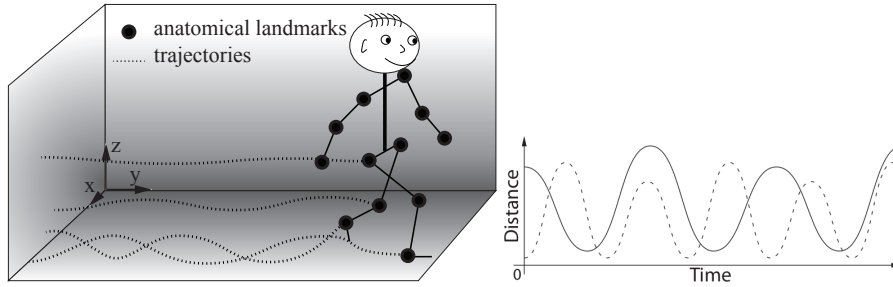


Fig. 8. Gait recognition based on 3D coordinates of anatomical landmarks.

specific language models, the pattern recognition based on *unsupervised methods*, that focus on situations when no linguistic corpus is available, represent quite recent stream of research [28]. One of the studied approaches uses *posteriorgram templates* [29] extracted using a phonetic recognizer. The DTW measure is often used with this data [29], but we would like to evaluate other options like ERD and ERP comparing the results. Again, the SMF is an ideal platform for this research and future applications:

data type	series of probabilities or of probability vectors
similarity measure	DTW, ERD, ERP; special distances between components
query type	subsequence query with variable length
requirements	data and similarity flexibility, high performance in future

5 Conclusions and Future Work

The data in the form of sequences are all around us in various forms and extensive volumes. The research in the area of subsequence matching has been very intensive resulting in many partial or full solutions in various sub-areas of the field. In this work, we identified several sub-tasks that circulate over the field and are tackled within various subsequence matching approaches and algorithms.

We present a generic subsequence matching framework (SMF) that brings the option of choosing freely among the existing partial solutions and combining them in order to achieve ideal solutions for heterogeneous requirements of different applications. Also, this framework overcomes the often mentioned implementation bias present in the field and it enables a straightforward utilization of techniques from different areas, for instance advanced metric indexes. We describe SMF on conceptual and implementation levels and present several examples and demonstration applications from diverse fields. The SMF is available under GPL license at <http://mufin.fi.muni.cz/smf/>.

The architecture of the SMF framework is strictly modular and thus one of natural directions of future development is implementation of other modules. Also, we will develop SMF according to requirements emerging from continuous research streams that utilize SMF. Finally and most importantly, we would

like to contribute to the efficiency of the subsequence matching systems by involvement of advanced metric indexes. We believe in a positive impact of such cooperation of these two research fields that were so far evolving relatively separately.

Acknowledgments

This work was supported by national research projects GACR 103/10/0886, and GACR P202/10/P220.

References

1. Keogh, E., Kasetty, S.: On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM Press (2002) 102–111
2. Keogh, E., Zhu, Q., Hu, B., Hay, Y., Xi, X., Wei, L., Ratanamahatana, C.A.: The UCR Time Series Classification/Clustering Homepage (2011)
3. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Springer (2006)
4. Novak, D., Batko, M., Zezula, P.: Metric Index: An Efficient and Scalable Solution for Precise and Approximate Similarity Search. *Information Systems* **36**(4) (June 2011) 721–733
5. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast Subsequence Matching in Time-Series Databases. *ACM SIGMOD Record* **23**(2) (1994) 419–429
6. Guttman, A.: R-trees. *ACM SIGMOD Record* **14**(2) (June 1984) 47–57
7. Korn, F., Jagadish, H.V., Faloutsos, C.: Efficiently supporting ad hoc queries in large datasets of time sequences. *ACM SIGMOD Record* **26**(2) (June 1997) 289–300
8. Cai, Y., Ng, R.: Indexing spatio-temporal trajectories with Chebyshev polynomials. *SIGMOD '04*. ACM Press, New York, USA (2004)
9. Chan, K.P., Fu, A.W.c.: Efficient Time Series Matching by Wavelets. In: *ICDE '99 Proceedings of the 15th International Conference on Data Engineering*. (1999) 126–133
10. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems* **3**(3) (2001) 263–286
11. Chakrabarti, K., Keogh, E., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems (TODS)* **27**(2) (2002) 188
12. Yi, B.K., Faloutsos, C.: Fast Time Sequence Indexing for Arbitrary Lp Norms. In: *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB 2000*. (September 2000) 385–394
13. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics Speech and Signal Processing* **26**(1) (1978) 43–49

14. Kim, S., Park, S., Chu, W.: An index-based approach for similarity search supporting time warping in large sequence databases. In: Proceedings of the 17th International Conference on Data Engineering, ICDE '01, IEEE Comput. Soc (2001) 607–614
15. Keogh, E., Ratanamahatana, C.A.: Exact indexing of dynamic time warping. *Knowledge and Information Systems* **7**(3) (May 2004) 358–386
16. Chen, L., Ng, R.: On the Marriage of L_p-norms and Edit Distance. In: Proceedings of the Thirtieth international conference on Very large data bases, VLDB '04. (2004) 792–803
17. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: Proceedings of the 2005 ACM SIGMOD international conference on Management of data - SIGMOD '05. SIGMOD '05, New York, USA, ACM Press (2005) 491
18. Vlachos, M., Gunopoulos, D., Kollios, G.: Discovering Similar Multidimensional Trajectories. In: Proceedings of the 18th International Conference on Data Engineering, Washington, DC, USA, IEEE Computer Society (2002) 673–684
19. Moon, Y.S., Whang, K.Y., Loh, W.K.: Duality-Based Subsequence Matching in Time-Series Databases. In: Proceedings of the 17th International Conference on Data Engineering. (2001) 263
20. Moon, Y.S., Whang, K.Y., Han, W.S.: General Match: A Subsequence Matching Method in Time-series Databases Based on Generalized Windows. In: International Conference on Management of Data. (2002) 382
21. Han, W.S., Lee, J., Moon, Y.S., Jiang, H.: Ranked Subsequence Matching in Time-series Databases. In: Proceedings of the 33rd international conference on Very large data bases, VLDB '07, ACM (2007) 423–434
22. Perng, C.s., Wang, H., Zhang, S.R., Parker, D.S.: Landmarks: A New Model for Similarity-based Pattern Querying in Time Series Databases. In: Proceedings of the 16th International Conference on Data Engineering, ICDE '00, Washington, DC, USA, IEEE Computer Society (2000) 33–42
23. Shieh, J., Keogh, E.: i SAX. In: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '08, New York, New York, USA, ACM Press (August 2008) 623
24. Camerra, A., Palpanas, T., Shieh, J., Keogh, E.: iSAX 2.0: Indexing and Mining One Billion Time Series. In: 2010 IEEE International Conference on Data Mining, IEEE (December 2010) 58–67
25. Batko, M., Novak, D., Zezula, P.: MESSIF: Metric Similarity Search Implementation Framework. In Thanos, C., Borri, F., Candela, L., eds.: *Digital Libraries: Research and Development*. Volume LNCS 4877. Springer (2007) 1–10
26. Bhanu, B., Han, J.: *Human Recognition at a Distance in Video*. Advances in Computer Vision and Pattern Recognition. Springer (2010)
27. Valcik, J., Sedmidubsky, J., Balazia, M., Zezula, P.: Identifying Walk Cycles for Human Recognition. In: Pacific Asia Workshop on Intelligence and Security Informatics (PAISI 2012), LNCS (2012) 9
28. Volny, P., Novak, D., Zezula, P.: Employing Subsequence Matching in Audio Data Processing. Technical report, FI, Masaryk University, Brno (2011)
29. Hazen, T.J., Shen, W., White, C.: Query-by-example Spoken Term Detection using Phonetic Posteriorgram Templates. In: 2009 IEEE Workshop on Automatic Speech Recognition Understanding, IEEE Comput. Soc. Press (2009) 421–426