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 solutions is so wide that their proper comparison and combination suitable for a particular task might be very complicated and time-consuming. In this work, we present a new generic Subsequence Matching Framework (SMF) that tries to overcome the aforementioned problem 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 a straightforward manner achieving new quality and efficiency. The strictly modular architecture and openness of SMF enables also involvement of efficient solutions from different fields, for instance advanced metric-based indexes.

1 Introduction

A large fraction 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*

S.W. Liddle et al. (Eds.): DEXA 2012, Part II, LNCS 7447, pp. 256–265, 2012.
© Springer-Verlag Berlin Heidelberg 2012

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 fruitful.

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. After a brief overview and analysis of current state of the field (Section 2), we propose a generic subsequence matching framework (SMF) in Section 3. Section 4 contains a detailed example of design and realization of a subsequence matching algorithm with the aid of SMF. The paper concludes in Section 5 by future research directions that cover possible performance boost enabled by a straightforward cooperation of SMF with advanced distance-based indexing and searching techniques [3,4]. Due to space limitations, an extended version of this work is available as a technical report [5].

2 Subsequence Matching Approaches

The field opening paper by Faloutsos et al. [6] 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.

Originally [6], 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 [7]. The Euclidean distance was used for searching since it satisfies the lower bounding lemma on data transformed by DFT.

The *data representation* and the choice of *distance function* are fundamental questions for each specific application. Current approaches regarding these questions were thoroughly overviewed and analyzed [5] with a conclusion that the questions of data representation and distance function can be practically separate from the specific subsequence matching algorithm; it is important to pair the data representation and the distance function wisely in order to satisfy the lower bounding lemma [6].

The work by Faloutsos et al. [6] encouraged many following works. Moon et al. [8] suggested a dual approach for slicing and indexing sequences. 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 [9]. Another significant leap forward was made by the effort of Keogh et al. in their work about exact indexing of Dynamic Time Warping [10]. 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 [11].

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. As a result, the same parts of the process (like DFT or DWT) are implemented repeatedly which leads to the phenomenon of the *implementation bias* [1]. The modern subsequence matching approaches [11,10] 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:

– 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*, SMF defines the functionality of individual module types and their communication interfaces; a subsequence matching algorithm is implemented as a *skeleton* that combines module types in a specific way and these can be instantiated by actual module implementations.

In Section 3.1, we describe the common sub-problems (sub-tasks) that we identified in the field and we define corresponding module types (conceptual level). Further, in Section 3.2, we justify our approach by describing fundamental subsequence algorithms in terms of SMF modules and we present a straightforward implementation of these algorithms within SMF. Section 3.3 is devoted to details about implementation of the framework.

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

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 individual modules in SMF.

Data Representation (Module: Data Transformer). 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) [6,12,13,14] or also by extracting some important characteristics that should improve the effectiveness of the retrieval [15]. 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 1 (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 sequence of real numbers S [6]
PAA	apply the PAA on sequence of real numbers S [14]
Landmarks	extract *landmarks* from sequence S [15]

Windows and Subsequences (Module: Slicer). Majority of the subsequence matching approaches partitions the data and/or query sequences into subsequences of, typically, fixed length (windows) [6,8,9,11]. 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 $S'.len = j - i + 1$.

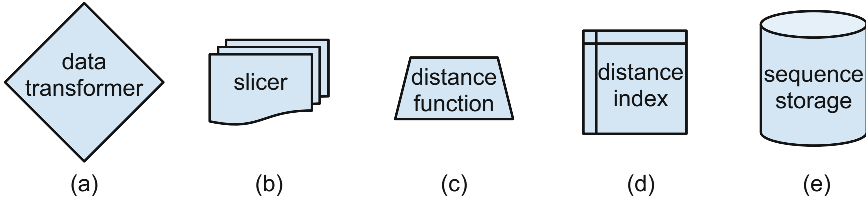

Fig. 1. Types of SMF modules and their notation

The module will be denoted as in Figure 1 (b) and its description and specific examples are as follows:

sequence slicer	**partition S into list of subsequences $S'_1, \ldots, S'_n$**
disjoint slicer	partition S disjointly into subsequences of length w [6]
sliding slicer	use sliding window of size w to partition S [6]

Sequence Distances (Module: Distance Function). There is 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 realization of 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 1 (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 [16]
ERP	calculate Edit distance with Real Penalty on S, T [17]
LB_PAA, LB_Keogh	measures which lower-bound the DTW [10]

Efficient Indexing (Module: Distance Index). 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 1 (d):

distance index	**evaluate efficiently distance-based QBE queries**
R-Tree family	index sequences as n-dimensional spatial data
iSAX tree	use a symbolic representation of the sequences [18,19]
metric indexes	index and search the data according to mutual distances [3]

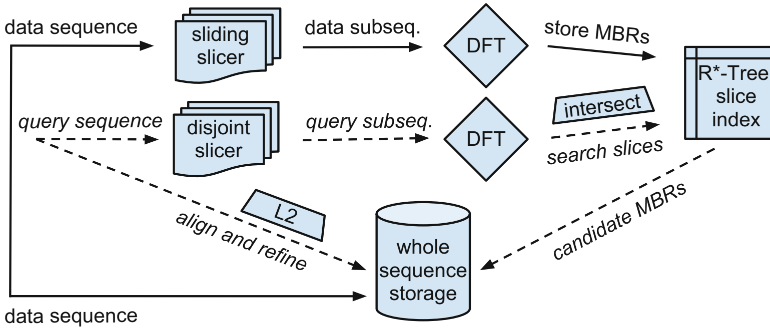

Fig. 2. Schema of the fundamental subsequence matching algorithm [6]

Efficient Aligning (Module: Sequence Storage). 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 [11]. We will call this module *sequence storage* (Figure 1 (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) [11]

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 [6] 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 2. 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'

($Q'.len = w$) 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 [8].

3.3 Implementation

The SMF was not implemented from scratch but with the aid of framework MESSIF [20]. The MESSIF is a collection of Java packages supporting mainly development of metric-based search approaches. SMF uses especially the following MESSIF 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.*

The *sequence* is in SMF handled very generally; it is defined as an interface which requires that each specific sequence type (e.g. a 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 that use the component distance d and, thus, this implementation can be independent of specific sequence type.

4 Example of Subsequence Algorithm with SMF

An algorithm is within SMF implemented as a *skeleton* – module types, their connections via specified interfaces, and all algorithm-specific operations. The algorithm is then configured and instantiated by 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 3 for this skeleton schema. It 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 this algorithm, we have to instantiate these module types with specific modules. Figure 4 shows the key part of a SMF configuration file that starts such algorithm. On the first two lines, the *sliding slicer* module is defined

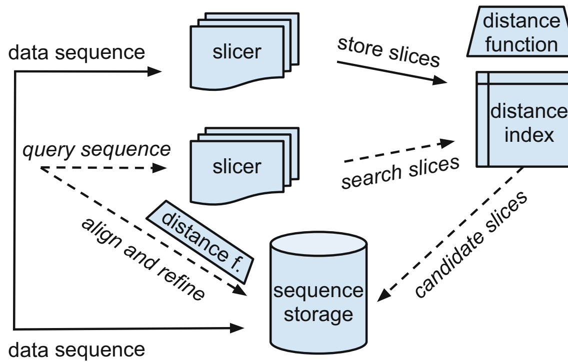

Fig. 3. Skeleton of a simple `VariableQueryAlgorithm` algorithm

```
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.ApproxAlgorithmDistanceIndex(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. 4. SMF configuration file for `VariableQueryAlgorithm` algorithm

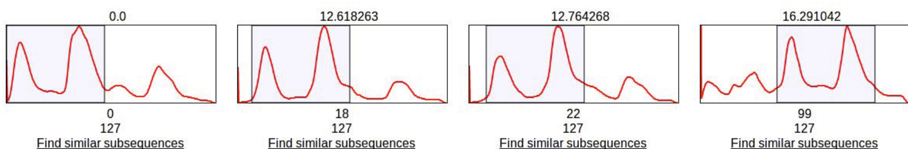

Fig. 5. Demonstration of `VariableQueryAlgorithm`: http://mufin.fi.muni.cz/subseq/

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 *algoritm*, namely a metric index M-Index [4] (we assume that 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 processed directly by the MESSIF framework that enables creation of such algorithm and its efficient management.

This algorithm is demonstrated by a publicly available demo on a set of real number sequences compared by L_2 distance (http://mufin.fi.muni.cz/subseq/). Figure 5 shows screenshot of the GUI of this demo: The user can specify a subsequence (offset and width) of the query sequence and the most similar subsequences are located within the indexes set.

5 Conclusions and Future Work

The sequence data is all around us in various forms and extensive volumes. The research in the area of subsequence matching has been very intensive, resulting in many full or partial solutions in various sub-areas. In this work, we have 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 an example of a design and realization of subsequence algorithm with the aid of SMF. The SMF is available under GPL license at `http://mufin.fi.muni.cz/smf/`.

The architecture of the 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 ACM SIGKDD 2002, pp. 102–111. ACM Press (2002)
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), 721–733 (2011)
5. Novak, D., Volny, P., Zezula, P.: Generic Subsequence Matching Framework: Modularity, Flexibility, Efficiency. Technical report, arXiv:1206.2510v1 (2012)
6. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast Subsequence Matching in Time-Series Databases. ACM SIGMOD Record 23(2), 419–429 (1994)
7. Guttman, A.: R-Trees: A Dynamic Index Structure for Spacial Searching. ACM SIGMOD Record 14(2), 47–57 (1984)
8. 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, p. 263 (2001)
9. 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, p. 382 (2002)
10. Keogh, E., Ratanamahatana, C.A.: Exact indexing of dynamic time warping. Knowledge and Information Systems 7(3), 358–386 (2004)
11. Han, W.S., Lee, J., Moon, Y.S., Jiang, H.: Ranked Subsequence Matching in Time-series Databases. In: Proceedings VLDB 2007, pp. 423–434. ACM (2007)
12. Chan, K.P., Fu, A.W.C.: Efficient Time Series Matching by Wavelets. In: Proceedings ICDE 1999, pp. 126–133 (1999)
13. Korn, F., Jagadish, H.V., Faloutsos, C.: Efficiently supporting ad hoc queries in large datasets of time sequences. ACM SIGMOD Record 26(2), 289–300 (1997)
14. 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), 263–286 (2001)
15. 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 ICDE 2000, pp. 33–42. IEEE Computer Society, Washington, DC (2000)
16. Sakoe, H., Chiba, S.: Dynamic programming algorithm optimization for spoken word recognition. IEEE Transactions on Acoustics Speech and Signal Processing 26(1), 43–49 (1978)
17. Chen, L., Ng, R.: On the Marriage of L_p-norms and Edit Distance. In: Proceedings of VLDB 2004, pp. 792–803 (2004)
18. Shieh, J., Keogh, E.: i SAX. In: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2008, p. 623. ACM Press, New York (2008)
19. 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, pp. 58–67. IEEE (2010)
20. Batko, M., Novak, D., Zezula, P.: MESSIF: Metric Similarity Search Implementation Framework. In: Thanos, C., Borri, F., Candela, L. (eds.) Digital Libraries: R&D. LNCS, vol. 4877, pp. 1–10. Springer, Heidelberg (2007)