Variable-length similarity search for very large data series : subsequence matching, motif and discord detection

Michele Linardi

UNIVERSITÉ PARIS DESCARTES

École doctorale EDITE de Paris ED130

Laboratoire/équipe de recherche: LIPADE

Variable-Length Similarity Search for Very Large Data Series

Subsequence Matching, Motif and Discord Detection

Par Michele LINARDI

Thèse de doctorat de Informatique

Dirigée par Themis PALPANAS

Présentée et soutenue publiquement le 21 août 2019

Devant un jury composé de :

Themis PALPANAS	[Directeur de thèse - Professeur] - Université Paris Descartes
Johann GAMPER	[Rapporteur - Professeur] - Université de Bolzano (Italie)
Panagiotis PAPAPETROU	[Rapporteur - Professeur] - Université de Stockholm (Suède)
Elisa FROMONT	[Examinateur - Professeur] - Université Rennes 1
Ioana ILEANA	[Examinateur - Maître de conférences] - Université Paris Descartes

Titre: Recherche de similarité de longueur variable pour l'analyse de grands séries temporelles: Appariement de séquences, Recherche de Motifs et Anomalies

Résumé (français) :

Les séries de données ou série chronologique (suite de valeurs numériques représentant l'évolution d'une quantité) sont devenues l'un des types de données les plus importants et les plus populaires, omniprésents dans presque tous les domaines scientifiques. Au cours des deux dernières décennies, mais de manière encore plus évidente au cours de cette dernière période, l'intérêt porté à ce type de données s'accroît rapidement. La raison en est principalement due aux récents progrès des technologies de détection, de mise en réseau, de traitement de données et de stockage, qui ont considérablement aidé le processus de génération et de collecte de grandes quantités de séries de données.

La recherche de similarité de séries de données est devenue une opération fondamentale au cœur de plusieurs algorithmes d'analyse et applications liées aux collections de séries de données. De nombreuses solutions à différents problèmes d'exploration de données, telles que le regroupement (clustering), la mise en correspondance des sous-séquences (subsequence matching), l'imputation des valeurs manquantes (imputation of missing values), la découverte de motifs (motif discovery) et la détection d'anomalies (discord discovery) sont basés sur l'utilisation de la recherche de similarité.

À cet égard, toutes les solutions sur mesure pour les problèmes susmentionnés nécessitent la connaissance préalable de la longueur de la série, sur laquelle une recherche de similarité est effectuée. Dans ce scénario, l'utilisateur doit connaître la longueur des résultats attendus, ce qui est souvent une hypothèse irréaliste. Cet aspect est donc très important. Dans plusieurs cas, la longueur est un paramètre critique qui influence sensiblement la qualité du résultat final.

En détail, nous avons noté que les index de séries de données permettent d'effectuer une recherche de similarité rapide. Néanmoins, tous les index existants ne peuvent répondre qu'aux requêtes d'une seule longueur (fixées au moment de la construction de l'index), ce qui constitue une limite sévère. Dans cette thèse, nous proposons d'abord *ULISSE*, la première index de série de données conçue pour répondre aux requêtes de recherche de similarité de *longueur variable*. Notre con-

tribution est double. Premièrement, nous introduisons une nouvelle technique de représentation, qui résume efficacement et succinctement plusieurs séquences de différentes longueurs. Sur la base de l'index proposé, nous décrivons des algorithmes efficaces pour la recherche de similarité approximative et exacte, combinant des visites d'index sur disque et des analyses séquentielles en mémoire. Notre approche prend en charge les séquences non normalisées et normalisées, et peut être utilisée sans modification avec la distance Euclidienne et le déformation temporelle dynamique (DTW), pour répondre aux requêtes de type: *k-NN* et ϵ-range. Nous évaluons notre approche de manière expérimentale en utilisant plusieurs jeux de données synthétiques et réels. Les résultats montrent que *ULISSE* s'est révélé de nombreuses fois plus efficace en terme de coût d'espace et de temps, par rapport aux approches concurrentes.

Par la suite, nous introduisons un nouveau *framework*, qui fournit un algorithme de recherche exacte de motifs (séquences fréquentes) et d'anomalies, qui trouve efficacement tous les motifs et les anomalies de tailles différentes. L'évaluation expérimentale que nous avons effectuée sur plusieurs ensembles de données réelles montre que nos approches sont jusqu'à des ordres de grandeur plus rapides que les alternatives. Nous démontrons en outre que nous pouvons supprimer la contrainte irréaliste d'effectuer des analyses en utilisant une longueur prédéfinie, ce qui conduit à des résultats plus intuitifs et exploitables, qui auraient autrement été manqués.

Mots-clés (français): Série de données, Recherche de Similarité, Longueur Variable, Lower-bounding, Subsequence Matching, Récherche de Motifs, Récherche d'Anomalies

Abstract:

Data series (ordered sequences of real valued points, a.k.a. time series) has become one of the most important and popular data-type, which is present in almost all scientific fields. For the last two decades, but more evidently in this last period the interest in this data-type is growing at a fast pace. The reason behind this is mainly due to the recent advances in sensing, networking, data processing and storage technologies, which have significantly assisted the process of generating and collecting large amounts of data series.

Data series *similarity search* has emerged as a fundamental operation at the core of several analysis tasks and applications related to data series collections. Many solutions to different data mining problems, such as Clustering, Subsequence Matching, Imputation of Missing Values, Motif Discovery, and Anomaly detection work by means of *similarity search*.

Data series indexes have been proposed for fast similarity search. Nevertheless all existing indexes can only answer queries of a single length (fixed at index construction time), which is a severe limitation. In this regard, all solutions for the aforementioned problems require the prior knowledge of the series length, on which *similarity search* is performed. Consequently, the user must know the length of the expected results, which is often an unrealistic assumption. This aspect is thus of paramount importance. In several cases, the length is a critical parameter that heavily influences the quality of the final outcome.

In this thesis, we propose scalable solutions that enable variable-length analysis of very large data series collections. We propose *ULISSE*, the first data series index structure designed for answering similarity search queries of *variable length*. Our contribution is two-fold. First, we introduce a novel representation technique, which effectively and succinctly summarizes multiple sequences of different length. Based on the proposed index, we describe efficient algorithms for approximate and exact similarity search, combining disk based index visits and in-memory sequential scans. Our approach supports non Z-normalized and Z-normalized sequences, and can be used with no changes with both Euclidean Distance and Dynamic Time Warping, for answering both *k-NN* and ϵ-range queries. We experimentally evaluate our approach using several synthetic and real datasets. The results show that *ULISSE* is several times, and up to orders of magnitude more efficient in terms of both space and time cost, when compared to competing approaches.

Subsequently, we introduce a new framework, which provides an exact and scalable

motif and discord discovery algorithm that efficiently finds all motifs and discords in a given range of lengths. The experimental evaluation we conducted over several diverse real datasets show that our approaches are up to orders of magnitude faster than the alternatives. We moreover demonstrate that we can remove the unrealistic constraint of performing analytics using a predefined length, leading to more intuitive and actionable results, which would have otherwise been missed.

Keywords : Data Series, Similarity Search, Variable Length, Lower-bounding, Subsequence Matching, Motif Discovery, Discord Discovery

Vita brevis, ars longa, occasio praeceps, experimentum periculosum, iudicium difficile (Hipócrates)

Life is short, and art long, opportunity fleeting, experimentations perilous, and judgment difficult. (Hippocrates)

Art long, vitalité brève, occasion précipitée, expérimentation périlleuse, jugement difficile (Hippocrate)

Acknowledgment

The last drop of this delicious Japanese whiskey, which my friend Federico offered me is gone. The retailers have already pulled the shutters down, and the bars are full of people talking. Tonight Marine is not here with me, but soon she will be back home. Many other things are far away, and perhaps too far.

Once again, another Parisian night is going to start; I can feel it. I can only hear a few cars on the street, and my living room is quiet. My cigarette is still smoking in the ashtray, and in a while it will be off. Now, my thoughts are running fast: *Verba volant, Scripta manent.*

During my entire life, many wise people have been telling me that achieving important goals requires a lot of "good" work and "luck" as well. At this point, looking back a bit and considering my personal story, I can surely say that back in 2011, I have been lucky the day I met Themis. He has been not only a great academic supervisor, a mentor full of wisdom, *savoir-faire*, and a big dose of *savoir-être*. Themis has been somebody beyond that. He is the person who taught me a lot through his actions and his suggestions, both in my academic career and more generally in my life. Many scientists might agree that good work is the result of excellence, diligence, and perseverance. I found all of these principles in the guidance of Themis.

I cannot see anybody helping me better than Themis, when I was not able to set my milestones, in the jungle of my discoveries. I cannot see anybody better than Themis, pushing me that right when motivation was low. I cannot see anybody else like Themis, explaining me with endless patience an ocean of details that make the difference. I cannot justify better a few regrets of mine today. If I had listened to you more, I could have done doubtlessly more. I am sure that this will help me to improve in the future. Each moment we spent working together was full of insights and great take away messages. It was an honor to be, and I will always be your student. Thank you Themi!

Going at the other side of the ocean, and deep in the south-west, I cannot forget to mention Eamonn Keogh and his former P.h.D. student Yan Zhu. I spent three great months at U.C. Riverside in Keogh's lab. There, I had the possibility to work in a magic environment: the home of the time series mining research. *Thank you guys!*

Among the faculty members of Paris Descartes, I cannot forget to cite and thank (a lot) Pavlos Moraitis, and Salima Benbernou. First of all, for their warm welcome on my first day, which helped me to feel very fast a member of the *LIPADE family*. Second, for their essential support and help during these last years.

Last but not least (yet among faculties), I would like to thank my advisors and colleagues at I.U.T. Paris Descartes: The director Xavier Sense, Mourad Ouziri, Pascal Poullard, Veronique Heiwy, Hassine Moungla, Jerome Fessy, and Denis Poitrenaud. Thank you for your help, support, time, and trust in me. I enjoyed this last year I spent with you as a teaching assistant. It was a very enriching and challenging experience!

I would also like to thank a lot the reviewers of my dissertation: Johann Gamper and Panagiotis Papapetrou, who accepted to review this manuscript respecting a very tight (prohibitive) deadline for the submission of the final report.

I guess that these last four years were the most emotional ones of my life. For some unexplainable reasons, they coincide with the period, where I was a Ph.D. student, and for the first time quite far from home. Since the beginning of this journey, I rapidly understood that a Ph.D. student is not only a Ph.D. student when he/she is in the corner of a big and dusty open space of the lab. A Ph.D. student is a Ph.D. student 24 hours per day, and a person who needs to think a lot. In this sense, critical reading and thinking are the keys to acquiring knowledge and looking for improvements. The more we do it, the better we can hope to perform. Given these premises, it is clear that the environment where I lived and the people who surrounded me have played a fundamental role for me. For this reason, I want to lovely thank everyone supported me, while I was working at my research. Hence, I would like to mention all my colleagues at Paris Descartes: Fedrico, Paul, Botao, Sabiha, Wissam, Anna, Cherifa, Heloise, Mohamed, and Khodor. *It has been a pleasure to share with you guys the same roof, door, open-space, and interesting discussions at the lab.*

The second half of the day for a Ph.D. student starts when he/she goes back home. This last year was the most important and full of work for me and I could not be luckier than I was once back home, where I had a lot of wonderful time with Marine. Her support, help, suggestions, love, and patient have been among the most beautiful gifts I could ever ask. *I love you ma cocotte!*

I would finally reserve a special thank to my family and friends that have been far, while I was doing my Ph.D. I am immensely grateful for your support, encourage-

ment, love, and constant presence. Even though we cannot see each other as we would, there is always a space in my thoughts for you. *I never dared to tell you that I miss you all!*

Paris, August 2019

Michele Linardi

Remerciements

Je viens de terminer mon verre de ce délicieux whisky japonais que mon pote Federico m'a offert. Les commerçants ont déjà baissé leur stores et les cafés sont pleins de gens qui bavardent. Ce soir, Marine n'est pas là avec moi, mais elle sera bientôt de retour. Beaucoup d'autres choses sont loin et peut-être trop loin. Une autre nuit parisienne commence; Je peux le sentir. Je n'entends que quelques voitures dans la rue et mon salon est calme. Ma cigarette fume encore dans le cendrier et dans un moment elle sera éteinte. Maintenant, mes pensées vont vites: *Verba volant, Scripta manent.*

Tout au long de ma vie, de nombreuses personnes m'ont dit que pour atteindre des objectifs importants, il faut beaucoup de "bon" travail et de la "chance" aussi. En regardant un peu en arrière, je peux sûrement dire qu'en 2011, j'ai eu de la chance le jour où j'ai rencontré Themis. Il a été un excellent encadrant plein de sagesse, de *savoir-faire*, et une bonne dose de *savoir-être*. Il m'a également beaucoup appris tant dans ma carrière universitaire que plus généralement dans ma vie. De nombreux scientifiques conviendront peut-être qu'un travail de qualité est le résultat de l'excellence, de la diligence et de la persévérance. J'ai trouvé tous ces principes dans les conseils de Thémis.

Personne ne m'a aidé mieux que Themis dans la jungle de mes découvertes. Personne ne m'a poussé comme il l'a fait lorsque ma motivation était faible. Personne comme Themis ne m'a expliqué avec une patience infinie, un océan de détails qui font vraiment la différence. Je ne peux pas mieux justifier quelques-uns de mes regrets aujourd'hui. Si je t'avais écouté en peux plus, j'aurais sans doute pu faire plus. Je suis sûr que cela m'aidera à m'améliorer à l'avenir. Chaque moment de travail meme les plus difficiles ont été vraiment formateurs. C'était un plaisir d'être ton étudiant et je le serai toujours. Merci Themi!

En allant de l'autre côté de l'océan dans le profond sud-ouest, je ne peux pas oublier de mentionner Eamonn Keogh et son ancienne doctorante Yan Zhu. J'ai passé trois mois formidables à l'Université de Riverside chez le labo de Eamonn Keogh. Ici, j'ai eu la possibilité de travailler dans un environnement magique: le temple de la recherche sur l'exploration de séries temporelles. *Merci les gars!*

Parmi les professeurs de Paris Descartes, je ne peux pas oublier de citer et remercier (beaucoup) Pavlos Moraitis et Salima Benbernou. Tout d'abord, pour leur accueil chaleureux à mon arrivée dans le labo, ce qui m'a permis de me sentir très vite membre de la famille *LIPADE*. Deuxièmement, pour leur soutien essentiel et leur aide durant ces dernières années.

Je voudrais aussi remercier mes collègues et encadrants de l'IUT Paris Descartes: le directeur Xavier Sense, Mourad Ouziri, Pascal Poullard, Véronique Heiwy, Hassine Moungla, Jérôme Fessy et Denis Poitrenaud. Merci pour votre aide, votre soutien, votre temps et votre confiance en moi. J'ai énormément apprécié cette année passée avec vous en tant qu'assistant d'enseignement. C'etait une expérience très enrichissante et stimulante!

Je tiens à remercier également les rapporteurs de ma thèse: Johann Gamper et Panagiotis Papapetrou, qui ont accepté de réviser ce manuscrit en respectant un délai très serré pour la présentation du rapport final.

Je suis sûr que ces quatre dernières années ont été les plus émotionnelles prenantes de ma vie. Pour plusieurs raisons inexplicables, elles coïncident avec la période où j'étais doctorant et pour la première fois assez loin de chez moi. Depuis le début de ce voyage, j'ai tout de suite compris qu'un doctorant n'est pas seulement un doctorant quand il/elle est dans son bureau à la fac. Un doctorant reste un doctorant 24 heures sur 24 car il/elle est une personne qui a besoin de réfléchir énormément. En ce sens, la lecture critique et la réflexion sont les clés pour acquérir des connaissances et rechercher des améliorations. Plus nous le faisons, mieux nous pouvons espérer de réussir dans notre intention. Il est donc clair que l'environnement dans lequel j'ai vécu et mon entourage ont été fondamentaux pour moi. Pour cette raison, je voudrais mentionner et remercier d'abord mes collègues de Paris Descartes: Fedrico, Paul, Botao, Sabiha, Wissam, Anna, Cherifa, Héloïse, Mohamed et Khodor. *C'était un plaisir de partager avec vous beaucoup de temps et des discussions très intéressantes au labo.*

La seconde moitié de la journée pour un doctorant commence lorsqu'il rentre chez lui. Cette dernière année a été la plus importante et la plus riche en travail pour

moi. Cependant je ne pouvais pas être plus chanceux que je ne l'étais une fois à la maison, où j'ai passé beaucoup de temps merveilleux avec Marine. Son soutien, son aide, ses suggestions, son amour et sa patience ont été parmi les plus beaux cadeaux que j'ai jamais pu demander. *Je t'aime ma cocotte!*

Je voudrais enfin réserver un remerciement particulier à ma famille et à mes amis qui ont été loin pendant ces dernières années. Je vous suis extrêmement reconnaissant pour votre soutien, vos encouragements, votre amour et votre présence constante. Même si nous ne pouvons pas nous voir comme nous le voudrions, il y a toujours une place pour vous dans mes pensées . *Je n'ai jamais osé vous le dire mais vous me manquez énormément !*

Paris, Août 2019

Michele Linardi

Contents

List of Figures

Chapter 1

Introduction

Data series (i.e., ordered sequences of points) are one of the most common data types, present in almost every scientific and social domain (such as meteorology, astronomy, chemistry, medicine, neuroscience, finance, agriculture, entomology, sociology, smart cities, marketing, operation health monitoring, human action recognition and others) [39, 79, 87, 32, 72]. If the dimension that imposes the ordering of the sequences is time then we talk about time series. Though, a series can also be defined over other measures (e.g., angle in radial profiles in astronomy, mass in mass spectroscopy in physics, position in genome sequences in biology, etc.). In the rest of this these, we use the terms *data series*, *time series*, and *sequence* interchangeably.

Once the data series have been collected, the domain experts face the arduous tasks of processing and analyzing them [116] in order to gain insights, e.g., by identifying similar patterns, and performing classification, or clustering. A core operation that is part of all these analysis tasks is **similarity search**, which has attracted lots of attention because of its importance [44, 3, 90, 8, 97, 113, 114, 102, 75, 48, 29, 28, 111, 63]. Nevertheless, all existing and efficient (mostly index-based) similarity search techniques are restricted in that they only support queries of a fixed length, and they require that this length is chosen at index construction. The same observation holds for techniques proposed to discover motifs [52, 21] and discords (i.e., anomalous subsequences) [21, 106]: they all assume a fixed sequence length, which has to be predefined.

Evidently, this is a constraint that penalizes the flexibility needed by analysts, who often times need to analyze patterns of slightly different lengths (within a given

data series collection) [35, 37, 20, 56, 57]. To that extent, we can report several examples (from real user studies), which benefit of multi-length search:

- In the *SENTINEL-2* mission data, oceanographers are interested in searching for similar coral bleaching patterns[1] of different lengths;
- At Airbus[2] engineers need to perform similarity search queries for patterns of variable length when studying aircraft takeoffs and landings [71];
- In neuroscience, analysts need to search in Electroencephalogram (EEG) recordings for Cyclic Alternating Patterns (CAP) of different lengths (duration), in order to get insights about brain activity during sleep [81].
- Entomologists want to study insects that feed by ingesting plant fluids, and cause devastating damage to agriculture worldwide [70]. This feeding processes can be recorded and analyzed in order to find repeated patterns that permit to understands and ultimately controlling the pests. It turns out that several interesting behaviors in these data occur along different time windows. Here, extracting variable length patterns becomes an essential operation.

In this thesis, we focus on three core problems that are based on similarity search, i.e., sequence matching, motif and discord discovery, and for which we remove the constraint of having to operate with a pre-determined sequence length. Our work is the first that proposes efficient and effective solutions for the above three problems when considering sequences of variable-length.

In the following, we provide an overview of these three problems, and the corresponding challenges.

1.1 Sequence Similarity Search (or Sequence Matching)

A sequence similarity search query is the operation that takes as input a data series Q (query) and a parameter $k \in \mathbb{N}$ finding the k most similar[3] series to Q

[1] `http://www.esa.int/Our_Activities/Observing_the_Earth`

[2] `http://www.airbus.com/`

[3] We provide a formal description of similarity measures in Chapter 3

in a data series collection C. This operation in very large data series collections is notoriously challenging [98, 112, 73, 74, 15], due to the high dimensionality (length) of the data series.

The vast majority of sequence matching solutions (including the state-of-the-art) relies on data summarization and indexing, which permit to perform fast and scalable similarity search [22, 77, 45, 3, 90, 36, 98, 16, 14, 113, 114, 103, 49].

Despite the effectiveness and benefits of the proposed indexing techniques, which have enabled and powered many applications over the years, they are restricted in different ways: either they only support queries of a fixed size, or they do not offer a scalable solution. The solutions working for a fixed length, require that this length is chosen at index construction time (it should be the same as the length of the series in the index). Given these premises, it is clear that a straightforward solution for answering sequence matching queries would be to use one of the available indexing techniques. However, in order to support (exact) results for variable-length sequence matching, we would need to:

- create several distinct indexes, one for each possible query length;
- for each one of these indexes, index all overlapping subsequences (using a sliding window).

We illustrate this fact in Figure 1.1, where we depict two queries of different lengths (ℓ_1 and ℓ_2).

Given a data series (from a collection C), we denote as D (shown in black), we draw in red the subsequences that we need to compare to each query in order to compute the exact answer. Using an indexing technique implies inserting all the subsequences in the index: since we want to answer queries of two different lengths, we are obliged to use two distinct indexes.

Nevertheless, this solution is prohibitively expensive, in both space and time. Space complexity is increased, since we need to index a large number of subsequences for each one of the supported query lengths: given a data series collection $C = D^1, ..., D^{|C|}$ and a query length range $[\ell_{min}, \ell_{max}]$, the number of subsequences we would normally have to examine (and index) is:

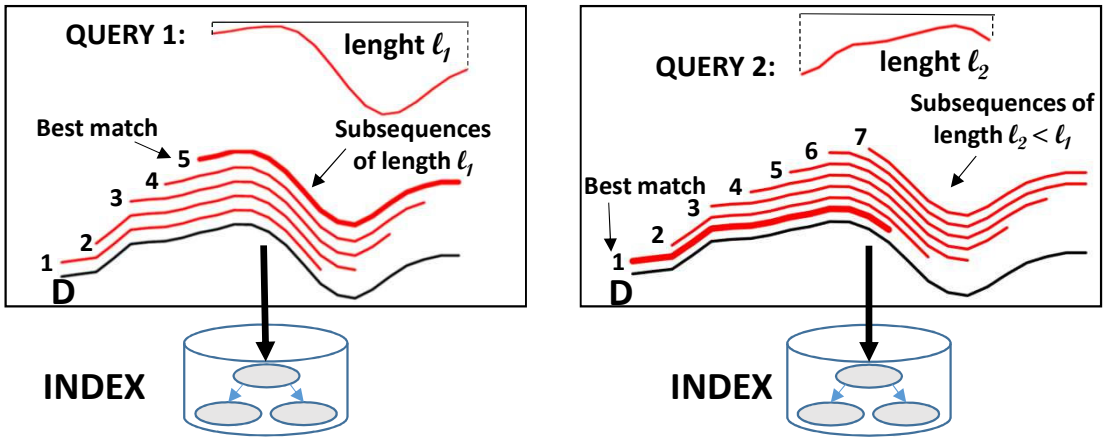


Figure 1.1: Indexing for supporting queries of 2 different lengths.

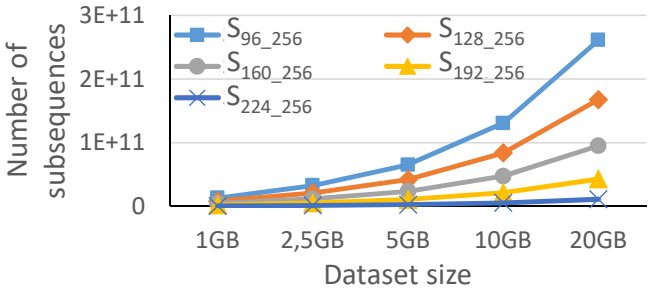


Figure 1.2: Search space evolution of variable length sequence matching. Each dataset contains series of length 256

$$S_{\ell_{min},\ell_{max}} = \sum_{\ell=1}^{(\ell_{max}-\ell_{min})+1} \sum_{i=1}^{|C|} (|D^i| - (\ell - 1)). \tag{1.1}$$

Figure 1.2 shows how quickly this number explodes as the dataset size and the query length range increase: considering the largest query length range (S_{96-256}) in the 20GB dataset, we end up with a collection of subsequences (that need to be indexed) more than 2 orders of magnitude larger than the original dataset. Computational time is significantly increased as well, since we have to construct different indexes for each query length we wish to support.

In the current literature, a technique based on multi-resolution indexes [38, 36] has been proposed in order to mitigate this explosion in size, by creating a smaller number of distinct indexes and performing more post-processing. Nonetheless,

this solution works exclusively for *non* Z-normalized series[4] (which means that it cannot return results with similar trends, but different absolute values), and thus, renders the solution useless for a wide spectrum of applications. Besides, it only mitigates the problem, since it still leads to a space explosion (albeit, at a lower rate), and therefore, it is not scalable, either.

We note that the technique discussed above (despite its limitations) is indeed the current state of the art, and no other technique has been proposed since, even though during the same period of time we have witnessed lots of activity and a steady stream of proposals on the *single-length* similarity search problem (e.g., [45, 3, 90, 7, 98, 114, 114, 103, 49]). This attests to the challenging nature of the first problem we are tackling in this thesis.

To tame the search space explosion, and to propose a new effective solution we follow a key idea: a data structure that indexes data series of length ℓ, already contains all the information necessary for reasoning about any subsequence of length $\ell' < \ell$ of these series. Therefore, the problem of enabling a data series index to answer queries of variable-length, becomes a problem of how to reorganize this information that already exists in the index. To this effect, we want to propose a new summarization technique that is able to represent contiguous and overlapping subsequences, leading to succinct, yet powerful summaries. It has to combine the representation of several subsequences within a single summary, and enable fast (approximate and exact) answers for variable-length sequence matching queries.

1.2 Data Series Motif

Over the last decade, data series motif discovery has emerged as one of the most used primitive for data series mining. Informally, we can describe motifs, as the most significant patterns that occur in a data series, i.e., subsequences that repeat themselves approximatively in the same manner. We report in Figure 1.3 a snippet of an Electrocardiogram recording (ECG), which records the electrical activity of the heart. This data series contains a series of repeated patterns highlighted in red, which represents heartbeat waveforms generated by ventricular contractions. These subsequences naturally represent a group of motifs over these data.

[4]Z-normalization transforms a series so that it has a mean value of zero, and a standard deviation of one. This allows the search to be effective, irrespective of shifting (i.e., offset translation) and scaling [43].

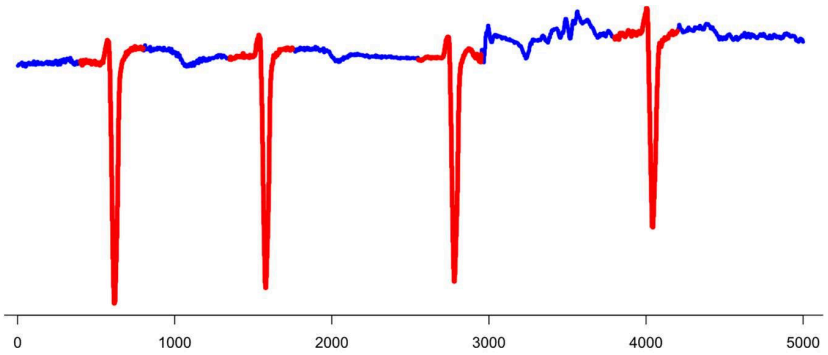


Figure 1.3: Electrocardiogram recording (ECG), with highlighted heartbeat waveforms motifs (in red)

Motif discovery has many applications to a wide variety of domains [100, 104], including classification, clustering, and rule discovery. More recently, there has been substantial progress on the scalability of motif discovery, and now massive datasets can be routinely searched on conventional hardware [100].

Another critical improvement in motif discovery, is the reduction in the number of parameters requiring specification. The first motif discovery algorithm, PROJECTION [12], required the user to set seven parameters, and it still only produces answers that are approximately correct. Researchers and practitioners have *"chipped"* away at this over the years [69, 84], and the current state-of-the-art algorithms only require the user to set a single parameter, which is the desired length of the motifs. Surprisingly, the ease with which we can now perform motif discovery has revealed that even this single burden on the user's experience or intuition can be *arduous* along the analysis task pipeline. The issue of being restricted to specify length as an input parameter, has been noted in domains that use motif discovery, such as cardiology [92] and speech therapy [95].

On the other hand, we can still consider the case, in which the user has good knowledge of the data domain. Also here, searching with one single motif length can be penalizing, especially when the data can contain motifs of various lengths.

To that extent, we show an example in Figure 1.4, where we report the 10-second and 12-second motifs discovered in the Electrical Penetration Graph (EPG) of an insect called Asian citrus psyllid. The first motif denotes the insect's highly technical probing skill as it searches for a rich leaf vein (stylet passage), whereas the second motif is just a simple repetitive "sucking" behavior (xylem ingestion). This example shows the utility of variable length motif discovery. An entomologist using

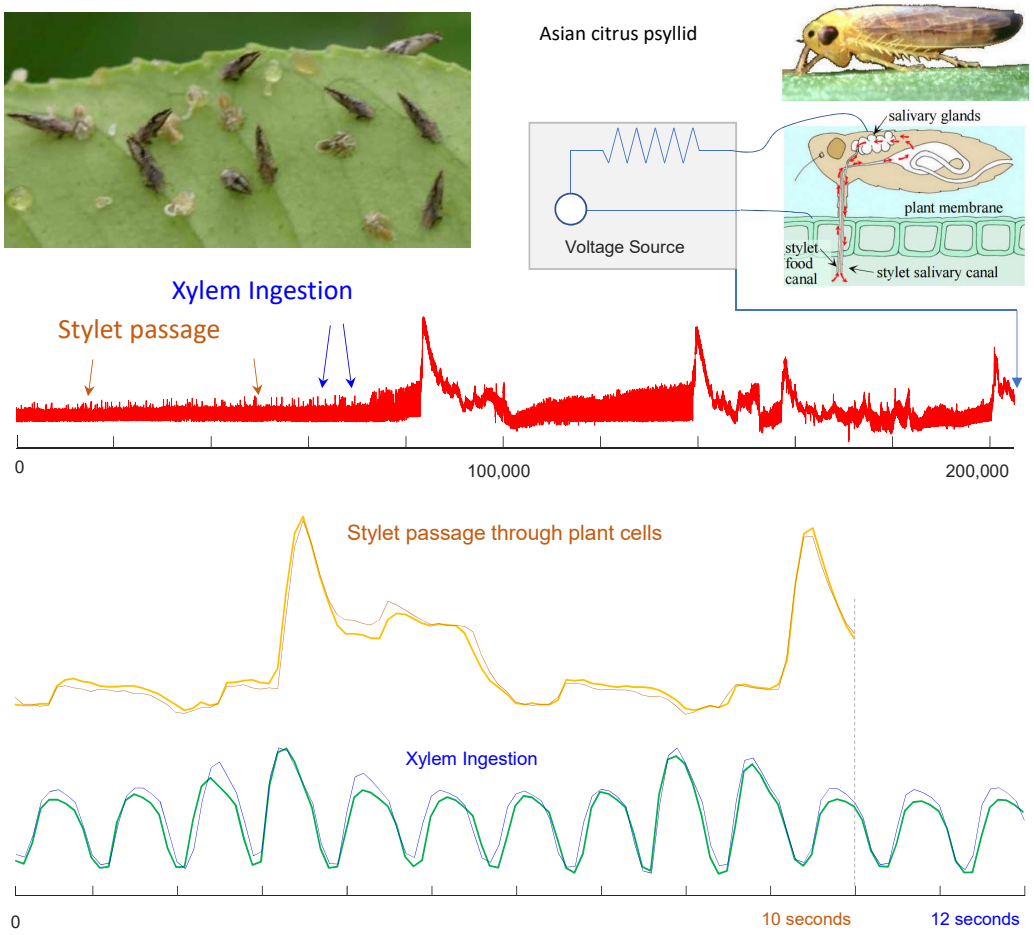


Figure 1.4: Demonstration of semantically different motifs, of slightly different lengths, extracted from a single dataset.

classic motif search, for instance at the length of 12 seconds, might have plausibly believed that this insect only engaged in xylem ingestion during this time period, and not realized the insect had found it necessary to reposition itself at least twice. The two motif pairs are radically different, reflecting two different types of insect activities. In order to capture all useful activity information within the data, a fast search of motifs over all lengths is necessary.

The obvious variable-length motif search is to make the state-of-the-art algorithm search over all lengths in a given range and rank the various length motifs discovered. We noticed that this strategy poses two challenges:

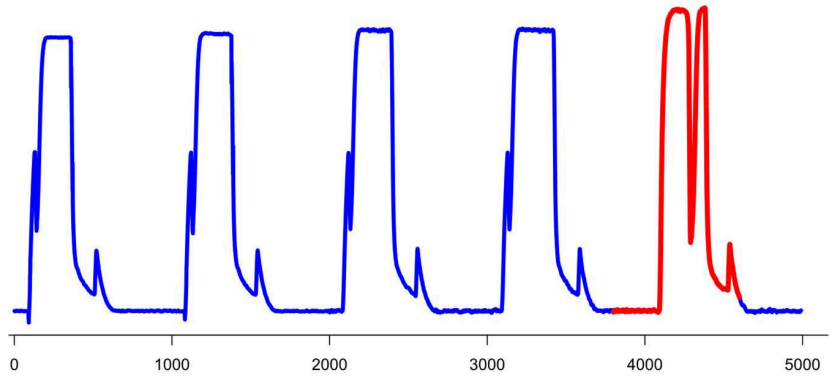


Figure 1.5: NASA Shuttle Valve data series. The discord, which represents a failure is highlighted in red.

- As in the case of the sequence matching task, the problem of searching over a much larger solution space in an efficient way is crucial for motif discovery as well.
- Once we enumerate motifs of several different lengths, we need to dispose of a strategy that permits to rank them.

1.3 Data Series Discord

Symmetrically to data series motif, another popular and well studied data series primitive, the discord [107, 40, 109, 86, 60], has been proposed to discover subsequences that represent outliers. Hence, with discords, we want to represent rare and abnormal patterns that occur in a Data Series datasets.

We depict an example in Figure 1.5, where a NASA Shuttle Valve data series is reported. Specifically, these data are recorded, while conducting cyclic conditions test in laboratory [5]. In the picture, the series contains measurements of the solenoid, which exhibit a cyclic phase. We know (from experts annotations) that the last cycle reports a failure (highlighted in red). As we note, the shape of this pattern clearly deviates from the previous cyclic patterns. In this case this subsequence is identified as a discord.

In the literature, the discord discovery solutions that have been proposed are not as effective and scalable as practice requires. The reasons are twofold:

[5] https://cs.fit.edu/~pkc/nasa/data/

- First, they only support fixed-length discord discovery. This rigidity with the subsequence length restricts the search space, and consequently, also the produced solutions and the effectiveness of the algorithm.
- Second, the existing techniques provide poor support for enumerating multiple discords, namely, for the identification of multiple anomalous subsequences. These works have considered only cases with up to *3* anomalous subsequences.

1.4 Contributions

Here, we provide the outline of our main contributions.

[**Variable-Length Similarity Search**] We first study the variable-length sequence matching query. We focus on efficiency improvement of similarity search, which is the operation at the core of the solution to this problem. In that regard, we propose $ULISSE$ (ULtra compact Index for variable-length Similarity SEarch in data series), which is the first single-index solution that supports fast answering of variable-length similarity search queries for both non Z-normalized and Z-normalized data series collections. *ULISSE* produces exact (i.e., correct) results, and is based on the following key idea: a data structure that indexes data series of length ℓ, already contains all the information necessary for reasoning about any subsequence of length $\ell' < \ell$ of these series. Therefore, the problem of enabling a data series index to answer queries of variable-length, becomes a problem of how to reorganize this information that already exists in the index. To this effect, *ULISSE* proposes a new summarization technique that is able to represent contiguous and overlapping subsequences, leading to succinct, yet powerful summaries: it combines the representation of several subsequences within a single summary, and enables fast (approximate and exact) similarity search for variable-length queries.

The contributions of this part of the thesis can be summarized as follows:

- We introduce the problem of Variable-Length Subsequences Indexing, which calls for a single index that can inherently answer queries of different lengths.
- We provide a new data series summarization technique, able to represent several contiguous series of different lengths.

- The technique we propose produces succinct, discretized envelopes for the summarized series, and can be applied to both non Z-normalized and Z-normalized data series.
- Based on this summarization technique, we develop an indexing algorithm, which organizes the series and their discretized summaries in a hierarchical tree structure, namely, the *ULISSE* index.
- We propose efficient exact and approximate K-NN algorithms, suitable for the *ULISSE* index, which can compute the similarity using either Euclidean Distance or Dynamic Time Warping measure.
- We perform an experimental evaluation with several synthetic and real datasets. The results demonstrate the effectiveness and scalability of ULISSE to dataset sizes that competing approaches cannot handle.
- Finally, we describe a prototype system we developed to support similarity search queries of *variable length*. It employs the *ULISSE* index in order to allow users to interactively run and explore the results of approximate and exact subsequence similarity search in both non Z-normalized and Z-normalized large data series collections.

[**Variable-Length Motif and Discord Discovery**] Furthermore, we consider the motif and discord discovery problems in conjunction. Our work wants to improve the efficiency of motif and discord search, we thus propose a solution that significantly extend the state-of-the-art algorithms.

In fact, the actual solution for fixed length motif and discord discovery [21] requires the user to define the length of the desired motif or discord. This mining operation is supported by computation of the *Matrix profile*, which is a meta data series storing the z-normalized Euclidean distance between each subsequence and its nearest neighbor. The Matrix profile does not only derive motifs and discords, but also ranks the other subsequences, giving a convenient and graphical representation of their occurrences and proximity. Unfortunately, this technique comes with an important shortcoming: it does not provide an effective solution for trying several different motif/discord lengths. Therefore, the analyst is forced to run the algorithm using all possible lengths in a range of interest, and rank the various motifs discovered, picking eventually the patterns that contain the desired insight.

To that extent, we firstly define the problems of variable-length motif and discord discovery, which significantly extend the usability of the motif and discord discovery operations, respectively. This premises allow us to build and propose a new

data series motif and discord framework. The contributions of this part of the thesis can be summarized as follows:

- A Variable Length Motif Discovery algorithm (VALMOD), which takes as input a data series T, and finds the subsequence pairs with the smallest Euclidean distance of each length in the (user-defined) range $[\ell_{min}, \ell_{max}]$. VALMOD is based on a novel lower bounding technique, which is specifically designed for the motif discovery problem.
- Furthermore, we extend VALMOD to the discord discovery problem. We propose a new exact variable-length discord discovery, which aims at finding the subsequence pairs with the largest Euclidean distances of each length in the (user-defined) range $[\ell_{min}, \ell_{max}]$.
- We evaluate our techniques using five diverse real datasets, and demonstrate the scalability of our approach. The results show that our solution is up to 20x faster than the state-of-the-art techniques.
- Furthermore, we present real case studies with datasets from entomology, seismology, and traffic data analysis, which demonstrate the usefulness of our approach in real world user studies.
- Finally, we present a motif discovery prototype system, which implements the scalable motif discovery algorithm (VALMOD), and uses a newly proposed meta-data structure that helps the user to select the most promising pattern length. We demonstrate how the proposed system efficiently finds all motifs in a given range of lengths, and outputs a length-invariant ranking of motifs.

1.5 Thesis Outline and Publications

In this thesis, it is important to note that we present the contributions that we can find in a collection of articles (both accepted for publication and under peer review). These papers are first-authored by the thesis writer. The manuscript is thus organized in chapters as follows:

In **Chapter 2**, we propose the revision of the state-of-the-art methods for the problems we treat in this thesis.

In **Chapter 3**, we introduce all the details and reports our scalable solution for variable-length sequence matching query. This work is published in:

- Scalable, Variable-Length Similarity Search in Data Series: The ULISSE Approach. (Michele Linardi, Themis Palpanas) **PVLDB 11(13), 2018**
- ULISSE: ULtra Compact Index for Variable-Length Similarity Search in Data Series (Michele Linardi, Themis Palpanas) **ICDE, 2018**

Moreover, one further article is under review:

- Scalable Data Series Subsequence Matching with ULISSE (Michele Linardi, Themis Palpanas) 2019

In **Chapter 4**, we propose the demonstration of VALS (Scalable VAriable-Length Similarity Search Suite), which is a system that permits to utilize and compare the state-of-the arts solutions for subsequence matching in data series. This work is under review:

- VALS with ULISSE: Variable-Length Similarity Search in Large Data Series Collections (Michele Linardi, Themis Palpanas), 2019

In **Chapter 5**, we propose our solution to data series motif and discord discovery. This work is published in

- Matrix Profile X: VALMOD - Scalable Discovery of Variable-Length Motifs in Data Series (Michele Linardi, Yan Zhu, Themis Palpanas, Eamonn Keogh) **SIGMOD Conference, 2018**

Morever, the following article is under review:

- Matrix Profile Goes MAD: Variable-Length Motif And Discord Discovery in Data Series (Michele Linardi, Yan Zhu, Themis Palpanas, Eamonn Keogh), 2019

In **Chapter 6**, we describe the demonstration of the motif discovery system we implemented. This work is published in:

- VALMOD: A Suite for Easy and Exact Detection of Variable Length Motifs in Data Series (Michele Linardi, Yan Zhu, Themis Palpanas, Eamonn Keogh) **SIGMOD Conference, 2018**

In **Chapter 7**, we conclude and we propose promising research directions for future work.

Chapter 2

Related work

2.1 Data series Indexes and Summarization

Several different kind of methods tackle the sequence matching problem (a.k.a similarity search). In this regard, one of the most considered techniques turns out to be data series indexing.

The literature includes several approaches, which are all based on the same principle: they first reduce the dimensionality of the data series by applying a summarization technique, which provides a compact form of the data (e.g., an index) that permits to prune the raw data space at search time.

Beyond the low representation error, a crucial and desirable property of a summarization is the lower bounding condition. Given two (real valued) data series, namely X, Y, the lower bound condition is formulated as:

$$D_{reduced}(Summ(X), Summ(Y)) \leq D_{raw}(X, Y) \tag{2.1}$$

We denote, with $D_{reduced}$ the distance between the summaries of X and Y (given by the function $Summ()$). On the other hand, D_{raw} is the distance computed in the real space of the series values. In general this latter distance is a metric. In this work we consider the case of Euclidean distance lower bounding [22, 77, 90, 6, 3, 98], which obeys to the triangular inequality condition[90]. Fur-

thermore, we consider also the case, where D_{raw} is not a metric, e.g., Dynamic Time Warping [46]. The Equation 2.1 serves the pruning strategy that is used in the proposed solutions to perform efficient subsequence matching.

The summarization techniques proposed in previous works are divided in two groups. The first one includes those that perform a spectral decomposition of the series. The most recent and popular methods are:

- Discrete Fourier Transform (DFT) [2], which consists into the extraction of the first c DFT coefficients (frequencies) of a data series.
- Singular Value Decomposition (SVD) [101] compresses data series based on the SVD theorem, which states that a real valued matrix can be decomposed in a spectral form. Only the k first components are used to represent the matrix, following the Principal Component Analysis (PCA).
- Discrete Wavelet Transform (DWT) [47, 76] was introduced to overcome the limitation of previous approaches, such as the Fourier transform. Specifically, DWT can use an infinite family of basis functions as opposed to DFT, which utilizes only the exponential function. Moreover, DWT transformation is performed in linear time, whereas the fast Fourier transformation has an additional logarithmic factor.

We note that, all the aforementioned dimensionality reduction technique have laid the foundation for several state-of-the-art similarity search systems, at least for a decade. More recently a suite of techniques based on piecewise approximation have been considered and evaluated:

- Piecewise Flat Approximation (PFA) [66].
- Piecewise Linear Approximation (PLA) [11].
- Adaptive Piecewise Linear Approximation (APLA) [10].
- Piecewise Aggregate Approximation (PAA) [41].
- Adaptive Piecewise Constant Approximation (APCA) [10].

All these works show that simple piecewise-based approximation outperform previous spectral decomposition based techniques by being easy to compute and index, and they moreover satisfy the lower bound condition.

Several index structure have been adapted, or specifically conceived to perform similarity search such as:

- **R*-tree** [5], which is a height-balanced spatial access method that partitions the data space into a hierarchy of nested overlapping rectangles.
- **M-tree** [13], which is a multidimensional, metric-space access method that uses hyper-spheres to divide the data entries according to their relative distances.
- **SFA trie** [85], which first summarizes the series using DFT, and it organizes them in a trie structure.
- **DSTree** [99], the DSTree is a binary tree that is built upon a data series summarization, which extends the APCA capability providing dynamic segmentation of the data series.
- **iSAX** (indexable Symbolic ApproXimation) [89, 8, 16, 113, 48] is a symbolic data series summarization built upon PAA. The summarization are stored in a hierarchical binary tree structure.

We revise here the Piecewise Aggregate Approximation (PAA) and the SAX approximation, which are the building block of the iSAX, the state-of-the-art indexing technique for similarity search in data series, and which we also use in our work.

2.1.1 Piecewise Aggregate Approximation

The Piecewise Aggregate Approximation (PAA) of a data series D, $PAA(D) = \{p_1, ..., p_w\}$, represents D in a w-dimensional space by means of w real-valued segments of length s, where the value of each segment is the mean of the corresponding values of D. We denote the first k dimensions of $PAA(D)$, $(k \leq w)$, as $PAA(D)_{1,..,k}$.

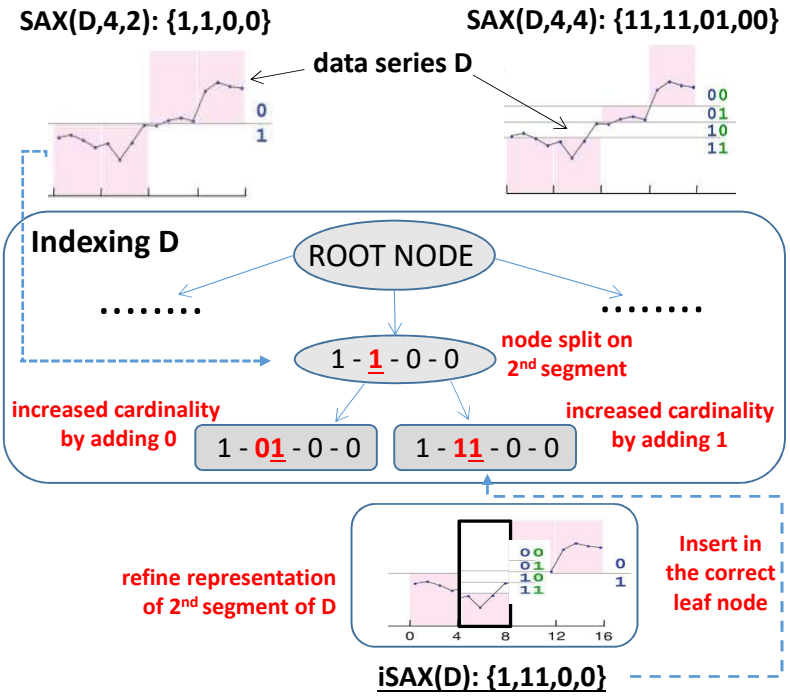


Figure 2.1: Indexing of series D (and an inner node split).

2.1.2 Symbolic Representation

We introduce here the $iSAX$ representation of a data series D, which stands for *indexable Symbolic Approximation*. The symbolic approximation is denoted by $SAX(D, w, |alphabet|)$, which is the representation of $PAA(D)$ by w discrete coefficients, drawn from an alphabet of cardinality $|alphabet|$ [90].

The main idea of the $iSAX$ representation (see Figure 2.1, top), is that the real-values space may be segmented by $|alphabet| - 1$ breakpoints in $|alphabet|$ regions that are labeled by distinct symbols: binary values (e.g., with $|alphabet| = 4$ the available labels are $\{00, 01, 10, 11\}$). $iSAX$ assigns symbols to the PAA coefficients, depending in which region they are located.

The iSAX data series index is a tree data structure [90, 16], which hierarchically organizes the $iSAX$ representations of a data series collection. It is composed by

three types of nodes (refer to Figure 2.1):

i The root node points to n children nodes (in the worst case $n = 2^w$, when the series in the collection cover all possible iSAX representations).

ii Each inner node contains the iSAX representation of all the series below it.

iii Each leaf node contains both the iSAX representation *and* the raw data of all the series inside it (in order to be able to prune false positives and produce exact, correct answers).

When the number of series in a leaf node becomes greater than the maximum leaf capacity, the leaf splits: it becomes an inner node and creates two new leaves, by increasing the cardinality of one of the segments of its iSAX representation. The two refined $iSAX$ representations (new bit set to *0* and *1*) are assigned to the two new leaves.

The similarity search algorithm, which is built upon the $iSAX$ index have state-of-the-art performance in solving the similarity search problem. It provides *ultra fast* approximate search, since the structure of the index permits to visit first the most promising node (the one with the same $iSAX$ representation of the query). Based on the same principle, the index permits to perform exact search. To that extent, the search algorithm visits in order the leaf nodes, which contain the most similar representation to the query. In the case of $K - NN$ query answering, since the $iSAX$ representation respects the lower bounding condition (both for Euclidean and Dynamic Time Warping distance), the search is over when the *best-so-far* distance is smaller than the actual lower bounding distance (computed by Equation (2.1)). The pruned candidates (those that are not considered) are guaranteed to contain no false negatives.

In general, we note that the $iSAX$ technique, but also all the approaches mentioned above share a common limitation: they can only work for a fixed, predetermined data series length, which has to be decided before the index creation.

Specifically, we know that Equation (2.1) holds *iff* the summarized series are of the same length. Hence, in this setting, the content of the query must be of fixed (predefined) length.

This limitation has been already studied. In the next part we present the details of the available solutions that we have been proposed so far. In the next part, we

denote the query-by-content problem, where the query can be of arbitrary length as Variable Length Similarity Search.

2.1.3 Indexing Techniques for Variable Length Similarity Search

At first, we note that Variable Length Similarity Search has been proposed only in the ϵ-range search variant, where the search outcome contains all the subsequence that have distance smaller or equal to ϵ. This means that *to the best of our knowledge* no indexing techniques for exact $K - NN$ search of variable length is available in the literature.

Faloustos et.al [22] proposed an indexing technique for variable length similarity search query called I-adaptive index, which is the first (seminal) approach that treats this problem.

In details, the I-adaptive index is built extracting the subsequences of a fixed length, which are grouped in MBRs (Minimum Bounding Rectangles) that form the building blocks of a R-tree. The authors presented two search methods, Prefix Search and Multiple Search, which work for arbitrary length queries. The first uses an index search using a fixed prefix of the query sequence. On the other hand, Multiple Search splits the query sequence in non-overlapping subsequences of fixed length and performs queries for each of these subsequences.

In a later work, Kahveci and Singh [38], proposed MRI (Multi Resolution Index), which is the first technique based on the construction of multiple indexes for variable length similarity search query. In this work, the authors clearly shown the limitation of Prefix Search and Multiple Search. The main disadvantage of these two approaches turns out to be the poor exploitation of the whole query sequence, since in the first, only the prefix subpart is used to perform a range query. If the prefix length sensibly differs from the entire query sequence length, the search space we need to consider can explode exponentially. The Multiple Search instead, segments the query in equal length parts, and perform a separate search with each segment, refining the search range at the end of each query. In practice, each query has a similar probability to prune the search space, with no real benefit deriving from the range refinement. To that extent, storing subsequences at different resolution (building indexes for different series lengths) provides an effective improvement of Multiple Search, since a greater part of a single query is used to

answer the query, considering furthermore multiple windows at different length. This provides more elasticity over the query length variability, improving search efficiency.

Kadiyala and Shiri in their work [36] have redesigned the MRI construction, exploiting the overlapping of the subsequences at different resolutions. This avoids to consider unnecessary subsequences, at the index building stage, drastically decreasing the indexing size and construction time. This new indexing technique, called Compact Multi Resolution Index (CMRI), has a space requirement, which is 99% smaller the one of MRI. The authors moreover, redefined the Multiple Search in order to access the disk only at the end of the multiple queries performed, optimizing also the range search proposed in MRI.

As a matter of fact, CMRI is the state-of-the-art Multi Resolution indexing technique for answering similarity search query of variable length. This solution is shown to be a strong and incremental contribution over the previous works.

More recently, Wu et al. [18] have proposed the KV-Match index, which supports ϵ-range similarity search queries of variable length, using both Z-normalized Euclidean and DTW distances. The idea of this technique is similar to the CMRI one, since many indexes are built for different subsequence window lengths, which are considered at query time using multiple query segments. We note that for Z-normalized sequences, this method provides exact answers only for *constrained* ϵ-range search. To this effect, two new parameters that constrain the mean and the standard deviation of a valid result are considered at query answering time.

2.1.4 Sequential Scan techniques for Similarity Search

Recent works have shown that similarity search based on sequential scans can be performed efficiently [78, 1]. These techniques aim to prune the search space exploiting the overlapping of the query candidates (subsequences). To that extent, two different methods have been proposed:

- UCR Suite (Rakthanmanon et al. [78]), which is an optimized serial scan algorithm for subsequence similarity search on a single long series. UCR Suite applies the following optimizations: (a) early-abandoning consists into stopping (abandon) the distance computations as soon as the partially calculated distance is greater than the best-so-far distance in $K-NN$ search; (b)

query points-reordering, which consists into sorting in descending order the points of the query according their absolute values. Specifically, the authors note that, when the data series are Z-normalized (mean of each series equal to *zero* and standard deviation equal to *one*), the points that are the farthest away from zero are likely to contribute the most to the final distance quantity. This heuristic allows to abandon earlier the distance computations (c) lower bound of DTW distance, here the author propose a multi-step lower bounding computation, which takes linear time, and permits to prune DTW distance calculations.

- MASS algorithm [68], which performs distance calculation in Frequency domain, namely on the Fourier transformation of the data series. This permits to compute the distance between a data series query Q and all the subsequences of length $|Q|$ in a series D in $O(|D|log(D))$ time. The complexity of the proposed algorithm does not depend to the length of Q.

The sequential scan techniques are mostly beneficial when the dataset consists of a single, very long data series, and queries are looking for potential matches in small subsequences of this long data series. Such approaches, in general, do not provide a large benefit when the dataset is composed of a large number of small data series, namely when the candidates do not overlap (have points in common).

2.1.5 Summary

In Figure 2.2, we report a summary of the current state of the art solutions for answering similarity search query in data series collections. For each line in the table, we report the type of index and the summarization technique applied to reduce the data series dimensionality, along with the kind of search (approximate and exact), the considered distance measures, the support of Z-Normalized query, and the possibility to issue queries of variable-length. We note that no indexing solution supports Z-Normalized similarity search queries of variable length. To that extent, UCR Suite represents a complete solution, which is based on sequential raw data scan and distances computation pruning. In this work, we want to propose a solution that can also draw the benefit from data series summarization and indexing.

Index	Summarization	Type of Similarity Search query supported		Distance Measure	Z-normalization	Support Query of variable length
		APPROXIMATE	EXACT			
R*-tree [22,47,76]	DWT, DFT, PAA	YES	YES	Euclidean, DTW	NO	YES
M-tree [13]	-	NO	YES	Metric	YES	NO
SFA-trie [85]	Symbolic DFT	YES	YES	Euclidean	YES	NO
DS-tree [99]	APCA	YES	YES	Euclidean	YES	NO
iSAX index [8,16,48,89,113]	iSAX	YES	YES	Euclidean	YES	NO
CMRI (R*-tree) [36]	APCA	YES	YES	Euclidean	NO	YES
UCR Suites (No Index) [78]	-	NO	YES	Euclidean, DTW	YES	YES
OUR INDEX [Chapter 3]	iSAX	YES	YES	Euclidean, DTW	YES	YES

Figure 2.2: Summary of current state of the art solutions for answering similarity search query in data series collections.

2.2 Motif and Discord Discovery

While research on data series similarity measures and data series query-by-content date back to the early 1990s [73], *data series motifs* and *data series discords* were both introduced just fifteen and twelve years ago, respectively [12, 23]. Following their definition, there was an explosion of interest in their use for diverse applications. There exist analogies between *data series motifs* and sequence *motifs* (in DNA), which have been exploited. For example, discriminative motifs in bioinformatics [91] inspired discriminative data series motifs (i.e., data series shapelets) [70]. Likewise, the work of Grabocka et al. [30] on generating idealized motifs, is similar to the idea of consensus sequence (or canonical sequence) in molecular biology. The literature on the general data series motif and discord search has been recently studied and referenced in several different recent studies [21, 109]. In the next parts we present the relevant techniques, reporting their characteristics and their *modus-operandi* as well.

2.2.1 Motif Discovery Techniques

The QUICK MOTIF [51] and STOMP [21] algorithms represent the state of the art for fixed-length motif discovery. QUICK MOTIF works building a summarized representation of the data using Piecewise Aggregate Approximation (PAA), and

arranges these summaries in Minimum Bounding Rectangles (MBRs) in a Hilbert R-Tree index. The algorithm then prunes the search space based on the MBRs. On the other hand, STOMP is based on the computation of the *matrix profile*, in order to discover the best matches for each subsequence. The smallest of these matches is called the motif pair. In general, we observe that both the above approaches solve a restricted version of the problem: they discover motif sets of cardinality two (i.e., motif pairs) of a fixed, predefined length.

The main idea of our work is to remove these limitations proposing a general and efficient solution, which can evaluate more candidates of various lengths. To that extent, we note that there are only three studies that deal with issues of variable length motifs, and attempt to address them [62, 25, 110, 24]. While these studies are pioneers in demonstrating the *utility* of variable length motifs, they cannot serve as practical solutions to the task at hand for two reasons: (i) they are all approximate, while we need to produce exact results; and (ii) they require setting many parameters (most of which are unintuitive). Approximate algorithms can be very useful in many contexts, if the amount of error can be bounded, or at least known. In certain cases, such as when analyzing seismological data, the threat of litigation, or even criminal proceedings [9], would make any analyst reluctant to use an approximate algorithm.

The other work to explicitly consider variable length motifs is MOEN [67]. Its operation is based on the distance computation of subsequences of increasing length, and a corresponding pruning strategy based on upper and lower bounds of the distance computed for the smaller length subsequences. Unlike the algorithms discussed above, MOEN is exact and requires few parameters. However, it has been tuned for producing only a single motif pair for each length in the range.

2.2.2 Discord Discovery Techniques

Exact discord discovery is a problem that has attracted lots of attention. The approaches that have been proposed in the literature can be divided in the following two different categories. First, the *index-based* solutions, i.e., Haar wavelets [23, 6] and SAX [40, 42, 86], where series are first discretized and then inserted in an index structure that supports fast similarity search. Second, the *sequential scan* solutions [109, 58, 23, 59, 105, 21], which consider the direct subsequence pairwise distance computations, and the corresponding search space optimization.

Indexing techniques are based on the discretization of the real valued data series, with several user defined parameters required for this operation. In general, selecting and tuning these parameters is not trivial, and the choices made may influence the behavior of the discord discovery algorithm, since it is strictly dependent on the quality of the data representation. In this regard, the most recent work in this category, GrammarViz [86], proposes a method of *Top-k 1st* discord search based on grammar compression of data series represented by discrete SAX coefficients. These representations are then inserted in a hierarchical structure, which permits to prune unpromising candidates subsequences. The intuition is that rare patterns are assigned to representations that have high *Kolmogorov complexity*. This means that a rare SAX string is not compressible, due to the lack of repeated terms.

The state of the art for the sequential scan methods is represented by STOMP, since computing the matrix profile permits to discover, in the same fashion as motifs, the *Top-k 1st* discords. Surprisingly, there exists just one work that addresses the problem of m^{th} discord discovery [105]. The authors of this work, proposed the Disk Aware discords Discovery algorithm (DAD), which is based on a smart sequential scan performed on disk resident data. This algorithm is divided in two parts. The first is discord candidate selection, where it identifies the sequences, whose nearest neighbor distance is less than a predefined range. The second part, which is called refinement, is applied in order to find the exact discords among the candidates. Despite the good performance that this algorithm exhibits in finding the first discord, when m is greater than one, it becomes hard to estimate an effective range. In turn, this leads to scalability problems, due to the explosion of the number of distances to compute.

In summary, while there exists a large and growing body of work on the motif and discord discovery problems, the idea, which motivates this work is to offer the first scalable, parameter-light, *exact* variable-length algorithm in the literature for solving both these problems.

Chapter 3

Scalable Data Series Subsequence Matching

Data series similarity search is an important operation and at the core of several analysis tasks and applications related to data series collections. Despite the fact that data series indexes enable fast similarity search, all existing indexes can only answer queries of a single length (fixed at index construction time), which is a severe limitation. In this chapter, we propose *ULISSE*, the first data series index structure designed for answering similarity search queries of *variable length*. Our contribution is two-fold. First, we introduce a novel representation technique, which effectively and succinctly summarizes multiple sequences of different length. Based on the proposed index, we describe efficient algorithms for approximate and exact similarity search, combining disk based index visits and in-memory sequential scans. Our approach supports non Z-normalized and Z-normalized sequences, and can be used with no changes with both Euclidean Distance and Dynamic Time Warping, for answering both *k-NN* and ϵ-range queries. We experimentally evaluate our approach using several synthetic and real datasets. The results show that *ULISSE* is several times, and up to orders of magnitude more efficient in terms of both space and time cost, when compared to competing approaches.

3.1 Chapter Organization

The rest of this chapter is organized as follows. In Section 3.2 we introduce the notation, and we formulate the problem. In Section 3.3, we describe the *ULISSE* summarization techniques, and in Sections 3.4 and 3.5 we explain our indexing and query answering algorithms. Section 3.6 describes the experimental evaluation, and we conclude in Section 3.7.

3.2 Preliminaries and Problem Formulation

Let a data series $D = d_1,...,d_{|D|}$ be a sequence of numbers $d_i \in \mathbb{R}$, where $i \in \mathbb{N}$ represents the position in D. We denote the length, or size of the data series D with $|D|$. The subsequence $D_{o,\ell}=d_o,...,d_{o+\ell-1}$ of length ℓ, is a contiguous subset of ℓ points of D starting at offset o, where $1 \leq o \leq |D|$ and $1 \leq \ell \leq |D| - o + 1$. A subsequence is itself a data series. A data series collection, C, is a set of data series.

We say that a data series D is Z-normalized, denoted D^n, when its mean μ is 0 and its standard deviation σ is 1. The normalized version of $D = d_1, ..., d_{|D|}$ is computed as follows: $D^n = \{\frac{d_1-\mu}{\sigma}, ..., \frac{d_{|D|}-\mu}{\sigma}\}$. Z-normalization is an essential operation in several applications, because it allows similarity search irrespective of shifting and scaling [43, 78].

Euclidean Distance. Given two data series $D = d_1, ..., d_{|D|}$ and $D' = d'_1, ..., d'_{|D'|}$ of the same length (i.e., $|D| = |D'|$), we can calculate their Euclidean Distance as follows: $ED(D, D') = \sqrt{\sum_i^{|D|} d(d_i, d'_i)}$, where the distance function d is applied to two real values, namely A and B, as follows: $d(A, B) = (A - B)^2$.

Dynamic Time Warping. The Euclidean distance is a lock-step measure, which is computed by summing up the distances between pairs of points that have the same positions in their respective series. Dynamic Time Warping (DTW) [50] represents a more elastic measure, allowing for small mis-alignments of the matched points on the x-axis.

Given two data series d and d', the DTW distance is computed by considering the differences between pairs of points $(d(d_i, d'_j))$, where the indexes i, j might be

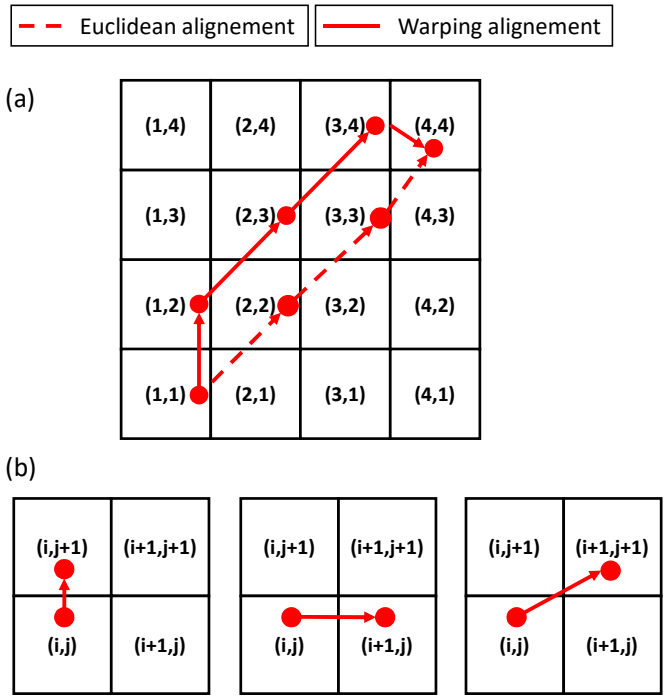


Figure 3.1: *(a)* Euclidean and Warping alignment in a squared matrix. *(b)* Valid index steps in a warping alignment.

different. In this manner, a particular alignment of d and d' is performed before to compute the distance. We define a sequence alignment as a vector of index pairs $A \in \mathbb{R}^{\ell \times 2}$, where $(i, j) \in A \iff 1 \leq i, j \leq \ell$, and ℓ is the length of the two series. The alignment of the Euclidean Distance is a special case, where the indexes are equal to their position in A. In the case of two series of length ℓ, the space of the possible alignments spans the paths that join two cells in a squared matrix composed by ℓ^2 cells. In Figure 3.1(a), we depict a Euclidean distance alignment of two series of length *4*, which exactly crosses the diagonal of the matrix, joining the cells *(1,1)* and *(4,4)*. On the other hand, in the same figure we report another possible alignment that we call warping alignment, which deviates from the diagonal. We use the terms *warping path* and *warping alignment* interchangeably.

In order to restrict the allowed paths, we can apply the following local constraints on the index pairs:

- We require that the first and last pairs of A correspond to the first and last

pairs of points in d and d', respectively. If $|d| = |d'| = \ell$, we have $A[1] = (1, 1)$ and $A[\ell] = (\ell, \ell)$. Furthermore, for any $(a, b), (c, d) \in A \iff (a \neq c) \vee (b \neq d)$. This latter, avoids to consider the same index pair twice in a single path.

- Given $k \in \mathbb{N}$ $(1 < k \leq \ell)$, we require that $A[k][0] - A[k-1][0] \leq 1$ and $A[k][1] - A[k-1][1] \leq 1$ always holds. This restricts each index to move by at most 1 unit to its next alignment position.

- Moreover, we always require that $A[k][0] - A[k-1][0] \geq 0$ and $A[k][1] - A[k-1][1] \geq 0$. This guarantees a monotonic movement of the path, towards the last index pair. In Figure 3.1(b), we depict the three possible steps that each index pair can perform in a valid alignment.

These constraints permit to bound the length of an alignment between two series of length ℓ, between ℓ and $2 \times \ell - 1$. Typically, warping paths are also subject to global constraints. We can thus set their maximum deviation from the matrix diagonal. In that regard, Sakoe and Chiba [83] and Itakura [34] proposed different warping path constraints, which restrict the matrix positions that a valid path can visit. The *Sakoe-Chiba band* [83] constraint allows each index of a warping path to be at most r points far from the diagonal (Euclidean Distance alignment). On the other hand, the *Itakura-parallelogram* [34] constraint allows to choose different r values depending on the index position i. In general, r is called the *warping window*.

Given a valid warping path, A^*, that satisfies the previously introduced constraints, we can formally define the DTW distance between two series d and d' of the same length ℓ, as:

$$DTW(d, d') = \underset{A^*}{\operatorname{argmin}}\left(\sqrt{\sum_{i}^{|A^*|} d(d_{A^*[i][0]}, d'_{A^*[i][1]})}\right).$$

We note that computing the DTW distance corresponds to finding the valid alignment that minimizes the sum of the distances.

In Figure 3.2, we consider two series (d and d'), which are extracted from two offsets that are *5* points away, in the same long sequence. In this manner, the prefix of d is equal to the suffix of d', which starts at position *6*. In the plots, the values of d span the right vertical axis, whereas those of d' the left one. If we compute the Euclidean distance, as depicted in Figure 3.2(a), the fixed alignment of points does not capture the similarity of the two series. On the other hand, when computing the DTW distance, the warping path aligns the two similar parts, as

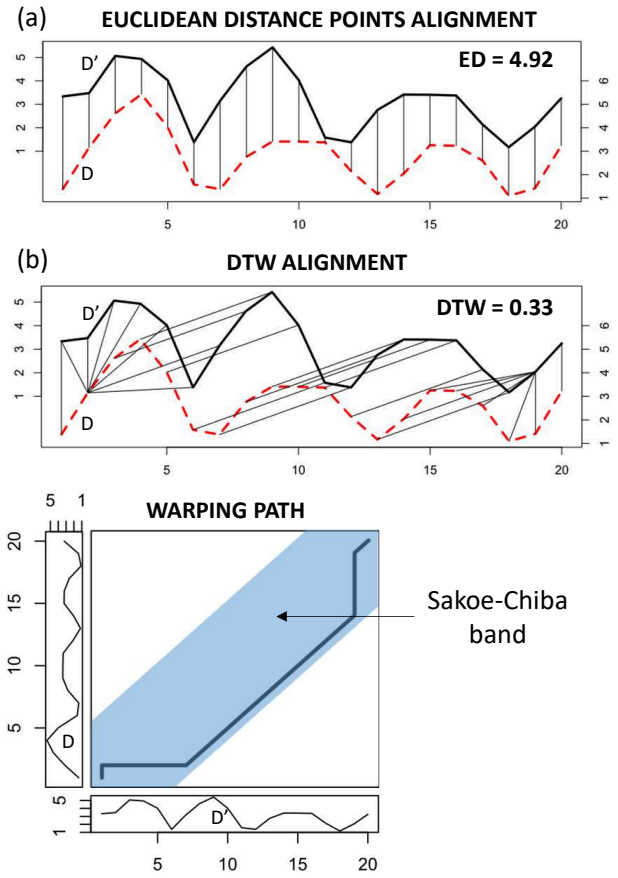


Figure 3.2: *(a)* Euclidean distance alignment between the data series D and D'. *(b)* DTW Alignment between D and D'.

reported in Figure 3.2(b). At the bottom of the figure, we also report the warping path, which is constrained by a *Sakoe-Chiba band*.

Problem Definition. The problem we wish to solve in this work is the following:

Problem 1 (Variable-Length Subsequences Indexing) *Given a data series collection C, and a series length range $[\ell_{min}, \ell_{max}]$, we want to build an index that supports exact similarity search, under the Euclidean and Dynamic Time Warping (DTW) measures, for queries of any length within the range $[\ell_{min}, \ell_{max}]$.*

In our case similarity search is formally defined as follows:

Definition 1 (Similarity search) *Given a data series collection* $C = \{D^1, ..., D^C\}$*, a series length range* $[\ell_{min}, \ell_{max}]$*, a query data series* Q*, where* $\ell_{min} \leq |Q| \leq \ell_{max}$*, and* $k \in \mathbb{N}$*, we want to find the set* $R = \{D^i_{o,\ell} | D^i \in C \wedge \ell = |Q| \wedge (\ell + o - 1) \leq |D^i|\}$*, where* $|R| = k$*. We require that* $\forall D^i_{o,\ell} \in R \nexists D^{i'}_{o',\ell'}$ *s.t.* $dist(D^{i'}_{o',\ell'}, Q) < dist(D^i_{o,\ell}, Q)$*, where* $\ell' = |Q|$*,* $(\ell' + o' - 1) \leq |D^{i'}|$ *and* $D^{i'} \in C$*. We informally call* R*, the* k *nearest neighbors set of* Q*.*

In this work, we perform Similarity Search using either Euclidean Distance (ED) or Dynamic Time Warping (DTW), as the *dist* function.

3.3 The ULISSE framework

The key idea of the *ULISSE* approach is the succinct summarization of *sets* of series, namely, overlapping subsequences. In this section, we present this summarization method.

3.3.1 Representing Multiple Subsequences

When we consider contiguous and overlapping subsequences of different lengths within the range $[\ell_{min}, \ell_{max}]$ (Figure 3.3.a), we expect the outcome as a bunch of similar series, whose differences are affected by the misalignment and the different number of points. We conduct a simple experiment in Figure 3.3.b, where we zero-align all the series shown in Figure 3.3.a; we call those *master series*.

Definition 2 (Master Series) *Given a data series* D*, and a subsequence length range* $[\ell_{min}, \ell_{max}]$*, the master series are subsequences of the form* $D_{i,min(|D|-i+1,\ell_{max})}$*, for each* i *such that* $1 \leq i \leq |D| - (\ell_{min} - 1)$*, where* $1 \leq \ell_{min} \leq \ell_{max} \leq |D|$*.*

We observe that the following property holds for the master series.

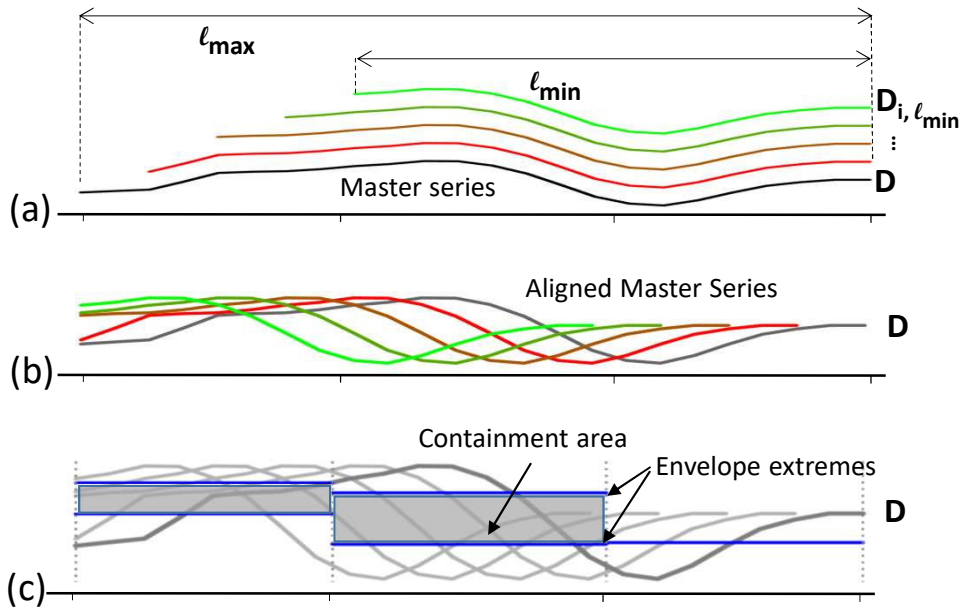


Figure 3.3: *a)* master series of D in the length interval ℓ_{min}, ℓ_{max}. *b)* Zero-aligned master series. *c)* Envelope built over the master series.

Lemma 1 *For any master series of the form $D_{i,\ell'}$, we have that $PAA(D_{i,\ell'})_{1,..,k} = PAA(D_{i,\ell''})_{1,..,k}$ holds for each ℓ'' such that $\ell'' \geq \ell_{min}$, $\ell'' \leq \ell' \leq \ell_{max}$ and $\ell', \ell'' \% k = 0$.*

Proof 1 *It trivially follows from the fact that, each non master series is always entirely overlapped by a master series. Since the subsequences are not subject to any scale normalization, their prefix coincides to the prefix of the equi-offset master series.* ■

Intuitively, the above lemma says that by computing only the PAA of the master series in D, we are able to represent the PAA prefix of any subsequence of D.

When we zero-align the PAA summaries of the master series, we compute the minimum and maximum PAA values (over all the subsequences) for each segment: this forms what we call an *Envelope* (refer to Figure 3.3.c). (When the length of a master series is not a multiple of the PAA segment length, we compute the PAA coefficients of the longest prefix, which is multiple of a segment.) We call *containment area* the space in between the segments that define the Envelope.

3.3.2 PAA Envelope for Non-Z-Normalized Subseqeunces

We denote by L and U the PAA coefficients, which delimit the lower and upper parts, respectively, of a containment area (see Figure 3.3.c). Furthermore, we introduce a parameter γ, which corresponds to the number of master series we represent by the Envelope. This allows to tune the number of subsequences of length in the range $[\ell_{min}, \ell_{max}]$, that a single Envelope represents, influencing both the tightness of a containment area and the size of the Index (number of computed Envelopes). We will show the effect of the relative tradeoff i.e., Tightness/Index size in the Experimental evaluation. Given a, the point from where we start to consider the subsequences in D, and s, the chosen length of the PAA segment, we refer to an Envelope using the following signature:

$$paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]} = [L, U] \tag{3.1}$$

3.3.3 PAA Envelope for Z-Normalized Subsequences

So far we have considered that each subsequence in the input series D is not subject of any scale normalization, i.e., is not Z-normalized. We introduce here a negative result, concerning the *unsuitability* of a generic $paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}$ to describe subsequences that are Z-normalized.

Intuitively, we argue that the PAA coefficients of a single master series $D_{i,a}$, generate a containment area, which may not embed the coefficients of the Z-normalized subsequence in the form $D'_{i,a'}$, for $a' < a$. This happens, because Z-normalization causes the subsequences of different lengths to change their shape, and even shift on the y-axis. Figure 3.4 depicts such an example.

We can now formalize this negative result.

Lemma 2 *A $paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}$ is not guaranteed to contain all the PAA coefficients of the Z-normalized subsequences of lengths $[\ell_{min}, \ell_{max}]$, of D.*

Proof 2 *To prove the correctness of the lemma, it suffices to pick such a case where a subsequence of D, namely $D_{a,\ell'}$, with $\ell_{min} \leq \ell' \leq \ell_{max}$, is not encoded by $paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}$. Formally, we should consider the case where*

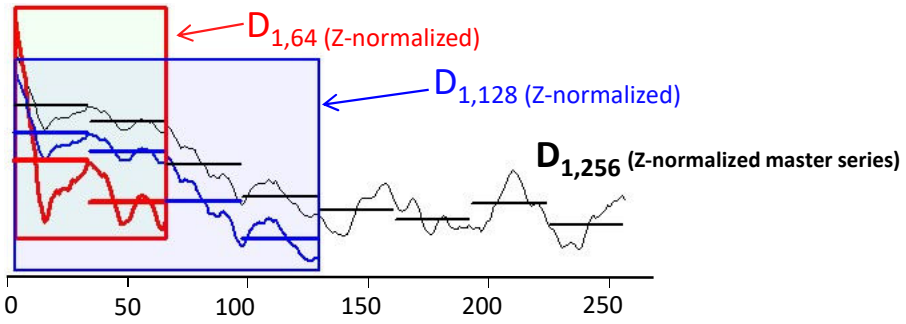


Figure 3.4: Master series $D_{1,256}$ with marked PAA coefficients.

$\exists k$ such that $PAA(D_{i,\ell'})_k > U_k$ or $PAA(D_{i,\ell'})_k < L_k$. We may pick a Z-normalized series D choosing $\ell_{max} = |D| = \ell_{min} + 1$ and $\gamma = 0$. The resulting $paaENV_{[D,\ell_{min}=\ell_{max}-1,\ell_{max}=|D|,i=1,\gamma=0,s]}$ obtains equal bounds, namely $L = U$. Let consider the z-normalized subsequence $D_{1,\ell_{min}}$. Its PAA coefficients must be in the envelope. This implies that, $PAA(D_{1,\ell_{min}})_1 = L_1 = U_1$ must hold. If s is the PAA segment length, in the case of Z-normalization, $PAA(D_{1,\ell_{min}})_1 = (((\sum_{i=1}^{s} d_i) - (\mu_{D_{1,\ell_{min}}} \times s))/\sigma_{D_{1,\ell_{min}}})/s$ and $U_1 = (((\sum_{i=1}^{s} d_i) - (\mu_D \times s))/\sigma_D)/s$. Therefore, the following equation: $(\mu_{D_{1,\ell_{min}}} \times s)/\sigma_{D_{1,\ell_{min}}} = (\mu_D \times s)/\sigma_D$ holds, which is equivalent to $\mu_{D_{1,\ell_{min}}}/\sigma_{D_{1,\ell_{min}}} = \mu_D/\sigma_D$. At this point we may have that $\mu_D = \mu_{D_{1,\ell_{min}}}$, when $D_{\ell_{max},1} = \mu_{D_{1,\ell_{min}}}$. This clearly leads to have a smaller dispersion on D than $D_{1,\ell_{min}}$ and thus $\sigma_D < \sigma_{D_{1,\ell_{min}}} \implies PAA(D_{1,\ell_{min}})_1 \neq L_1 \neq U_1$.
■

If we want to build an Envelope, containing all the Z-normalized sequences, we need to take into account the shifted coefficients of the Z-normalized subsequences, which are not master series. Hence, each PAA segment coefficient (in a master series) will be represented by the set of values resulting from the Z-normalizations of all the subsequences of length in $[\ell_{min}, \ell_{max}]$ that are not master series and contain that segment.

Given a generic master series $D_{i,\ell} = \{d_i, ..d_{i+\ell-1}\}$, and s the length of the segment, its k^{th} PAA coefficient set is computed by: $PAA^*(D_{i,\ell})_k = \{(\frac{(\sum_{p=s(k-1)+1}^{s(k-1)+s} d_p) - (\mu_{D_{i,\ell'}} \times s)}{\sigma_{D_{i,\ell'}}})/s | \ell_{min} \leq \ell' \leq \ell_{max}, \ell' \geq (s(k-1) + s - (i-1))\}$.

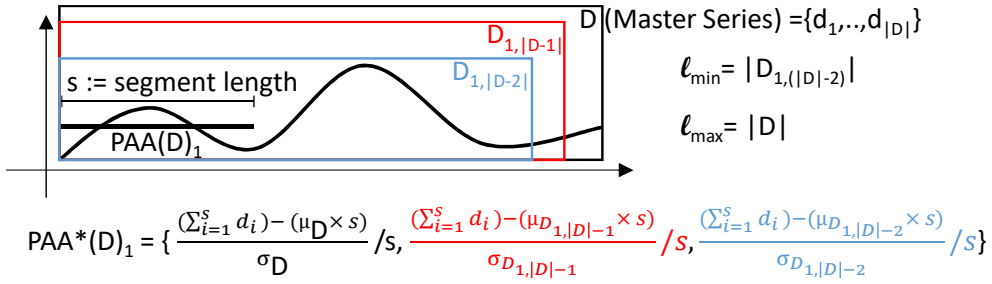


Figure 3.5: $PAA^*(D)_1$ computation. Three PAA coefficients are computed with the different normalizations.

In Figure 3.5, we depict an example of PAA^* computation for the first segment of the master series D.

We can then follow the same procedure as before (in the case of non Z-normalized sequences), computing the minimum and maximum PAA coefficients for each segment given by the above formula, in order to get the Envelope for the Z-normalized sequences (which we also denote with $paaENV$).

3.3.4 Indexing the Envelopes

Here, we define the procedure used to index the Envelopes. In that regard, we aim to adapt the $iSAX$ indexing mechanism (depicted in Figure 2.1).

Given a $paaENV$, we can translate its PAA extremes into the relative iSAX representation: $uENV_{paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}} = [iSAX(L), iSAX(U)]$, where $iSAX(L)$ ($iSAX(U)$) is the vector of the minimum (maximum) PAA coefficients of all the segments corresponding to the subsequences of D.

The *ULISSE* Envelope, $uENV$, represents the principal building block of the *ULISSE* index. Note that, we might remove for brevity the subscript containing the parameters from the $uENV$ notation, when they are explicit.

In Figure 3.6, we show a small example of envelope building, given an input series D. The picture shows the PAA coefficients computation of the master series. They are calculated by using a sliding window starting at point $a = 1$, which stops after γ steps. Note that the Envelope generates a containment area, which embeds

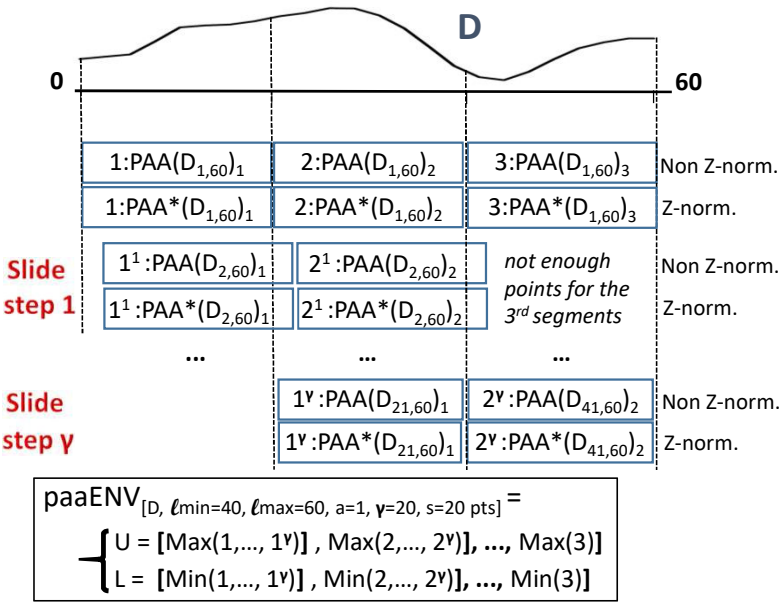


Figure 3.6: $uENV$ building, with input: data series D of length 60, PAA segment size $= 20$, $\gamma = 20$, $\ell_{min} = 40$ and $\ell_{max} = 60$.

all the subsequences of D of all lengths in the range $[\ell_{min}, \ell_{max}]$.

3.4 Indexing Algorithm

3.4.1 Indexing Non-Z-Normalized Subsequences

We are now ready to introduce the algorithms for building an $uENV$. Algorithm 1 describes the procedure for non-Z-normalized subsequences. As we noticed, maintaining the running sum of the last s points, i.e., the length of a PAA segment (refer to Line 7), allows us to compute all the PAA values of the expected envelope in $O(w(\ell_{max} + \gamma))$ time in the worst case, where $\ell_{max} + \gamma$ is the points window we need to take into account for processing each master series, and w is the number of PAA segments in the maximum subsequence length ℓ_{max}. Since w, is usually a very small number (ranging between 8-16), it essentially plays the role of a constant factor. In order to consider not more than γ steps for each segment position, we store how many times we use it, to update the final envelope in the vector, in

Algorithm 1: *uENV computation*

Input: **float**[] D, **int** s, **int** ℓ_{min}, **int** ℓ_{max}, **int** γ, **int** a
Output: **uENV**[$\mathbf{iSAX_{min}}$, $\mathbf{iSAX_{max}}$]

```
1  int w ← ⌊ℓ_max/s⌋ ;
2  int segUpdateList[S] ← {0,...,0};
3  float U[w] ← {−∞, ..., −∞}, L[w] ← {∞, ..., ∞};
4  if |D| − (i − 1) ≥ ℓ_min then
5  |   float paaRSum ← 0;
   |   // iterate the master series.
6  |   for i ← a to min(|D|,a + ℓ_max + γ) do
   |   |   // running sum of paa segment
7  |   |   paaRSum ← paaRSum + D[i];
8  |   |   if (j-a) > s then
9  |   |   |   paaRSum ← paaRSum - D[i-s];
10 |   |   for z ← 1 to min(⌊[i-(a-1)] / s⌋,w) do
11 |   |   |   if segUpdatedList[z] ≤ γ then
12 |   |   |   |   segUpdateList[z] ++;
13 |   |   |   |   float paa ← (paaRSum / s);
14 |   |   |   |   L[z] ← min(paa, L[z]);
15 |   |   |   |   U[z] ← max(paa, U[z]);
16 |   uENV ← [iSAX(L),iSAX(U)];
17 else
18 |   uENV ← ∅;
```

Line 2.

3.4.2 Indexing Z-Normalized Subsequences

In Algorithm 2, we show the procedure that computes an indexable Envelope for Z-normalized sequences, which we denote as $uENV_{norm}$. This routine iterates over the points of the overlapping subsequences of variable length (*First loop* in Line 7), and performs the computation in two parts. The first operation consists of computing the sum of each PAA segment we keep in the vector $PAAs$ defined in Line 2. When we encounter a new point, we update the sum of all the segments that contain that point (Lines 8- 11). The second part, starting in Line 16 (*Second loop*), performs the segment normalizations, which depend on the statistics (mean and std.deviation) of all the subsequences of different length (master and non-master series), in which they appear. During this step, we keep the sum and the squared

Algorithm 2: *$uENV_{norm}$ computation*

Input: **float**[] D, **int** s, **int** ℓ_{min}, **int** ℓ_{max}, **int** γ, **int** a
Output: $\mathbf{uENV_{norm}}$[$\mathbf{iSAX_{min}}$, $\mathbf{iSAX_{max}}$]

```
1  int w ← ⌊ℓ_max/s⌋ ;
   // sum of PAA segments values
2  float PAAs [ℓ_max + γ − (s − 1)] ← {0,...,0};
3  float U[w] ← {−∞, ..., −∞}, L[w] ← {∞, ..., ∞};
4  if |D| − (a − 1) ≥ ℓ_min then
5      int nSeg← 1;
6      float accSum, accSqSum ← 0;
       // First loop:  Iterate the points.
7      for i ← a to min(|D|,(a+ℓ_max+γ)) do
           // update sum of PAA segments values
8          if i − a > s then
9              nSeg++;
10             PAAs[nSeg] ← PAAs[nSeg-1] - D[i-s];
11         PAAs[nSeg] += D[i];
           // keep sum and squared sum.
12         accSum += D[i], accSqSum += (D[i])^2;
           // the window contains enough points.
13         if i-(a-1) ≥ ℓ_min then
14             acSAc ← accSum, acSqSAc ← accSqSum;
15             int nMse ← min(γ+1,(i-(a-1)-ℓ_min) + 1);
               // Second loop:  Normalizations of PAA coefficients.
16             for j ← 1 to nMse do
17                 int wSubSeq ← i-(a-1)-(j-1) ;
18                 if wSubSeq ≤ ℓ_max then
19                     float μ ←acSAc/wSubSeq;
20                     float σ ← sqrt((acSqSAc/wSubSeq − μ^2));
21                     int nSeg ← ⌊wSubSeq÷s⌋;
22                     for z ← 1 to nSeg do
23                         float a ← PAAs[j+[(z-1)×s]];
24                         float b ← s×μ;
25                         float paaNorm ← ((a−b)/σ)/s;
26                         L[z] ← min(paaNorm, L[z]);
27                         U[z] ← max(paaNorm, U[z]);
28                 acSAc -= D[j], acSqSAc -= (D[j])^2;
29     uENV_norm ← [iSAX(L),iSAX(U)];
30 else
31     uENV_norm ← ∅;
```

sum of the window, which permits us to compute the mean and the standard deviation in constant time (Lines 19,20). We then compute the Z-normalizations of all the PAA coefficients in Line 25, by using Equation 3.3.3.

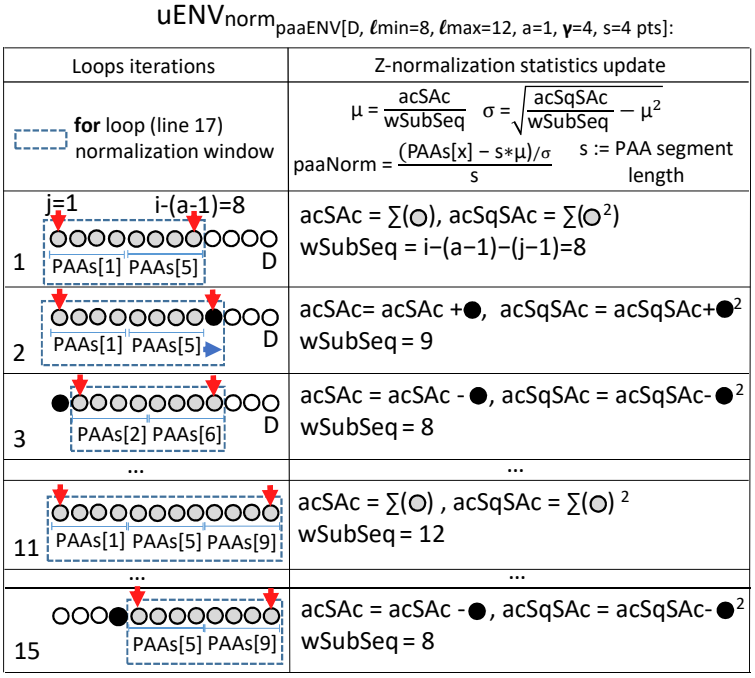


Figure 3.7: Running example of Algorithm 2. *Left column*) Points iteration. *Right column*) Statistics update at each step.

In Figure 3.7, we show an example that illustrates the operation of the algorithm. In *1*, the *First loop* has iterated over *8* points (marked with the dashed square). Since they form a subsequence of length ℓ_{min}, the *Second Loop* starts to compute the Z-normalized PAA coefficients of the two segments, computing the mean and the standard deviation using the sum ($acSAc$) and squared sum ($acSqAc$) of the points considered by the *First loop* (gray circles). The second step takes place after that the *First Loop* has considered the 9^{th} point (black circle) of the series. Here, the *Second Loop* updates the sum and the squared sum, with the new point, calculating then the corresponding new Z-normalized PAA coefficients. At step *3*, the algorithm considers the second subsequence of length ℓ_{min}, which is contained in the nine points window. The *Second Loop* considers in order all the overlapping subsequences, with different prefixes and length. This permits to update the statistics (and all possible normalizations) in constant time. The algorithm terminates, when all the points are considered by the *First loop*, and the *Second Loop* either encounters a subsequence of length ℓmin (as depicted in the step *15*), or performs at most γ iterations, since all the subsequences starting at position $a + \gamma + 1$ or later (if any) will be represented by other Envelopes.

Algorithm 3: *ULISSE index computation*

Input: **Collection** C, **int** s, **int** ℓ_{min}, **int** ℓ_{max}, **int** γ, **bool** $bNorm$
Output: ***ULISSE*** **index** I

```
1  foreach D in C do
2      int a' ← ∅;
3      uENV E ← ∅;
4      while true do
5          if bNorm then
6              E ← uENV_norm(D, s, ℓ_min, ℓ_max, γ, a');
7          else
8              E ← uENV(D, s, ℓ_min, ℓ_max, γ, a');
9          a' ← a' + γ + 1 ;
10         if E == ∅ then
11             break;
12         bulkLoadingIndexing(I, E);
13         I.inMemoryList.add(maxCardinality(E));
```

Complexity Analysis

Given w, the number of PAA segments in the window of length ℓ_{max}, and $M = \ell_{max} - \ell_{min} + \gamma$, the number of master series we need to consider, building a normalized Envelope, $uENV_{norm}$, takes $O(M\gamma w)$ time.

3.4.3 Building the index

We now introduce the algorithm, which builds a *ULISSE* index upon a data series collection. We maintain the structure of the $iSAX$ index [16], introduced in the preliminaries.

Each *ULISSE* internal node stores the Envelope $uENV$ that represents all the sequences in the subtree rooted at that node. Leaf nodes contain several Envelopes, which by construction have the same $iSAX(L)$. On the contrary, their $iSAX(U)$ varies, since it get updated with every new insertion in the node. (Note that, inserting by keeping the same $iSAX(U)$ and updating $iSAX(L)$ represents a symmetric and equivalent choice.)

In Figure 3.8, we show the structure of the *ULISSE* index during the insertion of an Envelope (rectangular/yellow box). Note that insertions are performed based on $iSAX(L)$ (underlined in the figure). Once we find a node with the same $iSAX(L) = (1-0-0-0)$ (Figure 3.8, $1^{st}step$) if this is an inner node, we descend its subtree (always following the $iSAX(L)$ representations) until we encounter a leaf. During this path traversal, we also update the $iSAX$ representation of the Envelope we are inserting, by increasing the number of bits of the segments, as necessary. In our example, when the Envelope arrives at the leaf, it has increased the cardinality of the second segment to two bits: $iSAX(L) = (1-\mathbf{10}-0-0)$, and similarly for $iSAX(U)$ (Figure 3.8, $2^{nd}step$). Along with the Envelope, we store in the leaf a pointer to the location on disk for the corresponding raw data series. We note that, during this operation, we do not move any raw data into the index.

To conclude the insertion operation, we also update the $iSAX(U)$ of the nodes visited along the path to the leaf, where the insertion took place. In our example, we update the upper part of the leaf Envelope to $iSAX(U) = (1-\mathbf{11}-0-0)$, as well as the upper part of the Envelope of the leaf's parent to $iSAX(U) = (1-\mathbf{1}-0-0)$ (Figure 3.8, $3^{rd}step$). This brings the *ULISSE* index to a consistent state after the insertion of the Envelope.

Algorithm 3 describes the procedure, which iterates over the series of the input collection C, and inserts them in the index. Note that function *bulkLoadingIndexing* in Line 12 may use different bulk loading techniques. In our experiments, we used the iSAX 2.0 bulk loading algorithm [7]. Alongside the index, we also keep in memory (using the raw data order) all the Envelopes, represented by the symbols of the highest $iSAX$ cardinality available (Line 13). This information is used during query answering.

Space complexity analysis

The index space complexity is equivalent for the case of Z-normalized and non Z-normalized sequences. The choice of γ determines the number of *Envelopes* generated and thus the index size. Hence, given a data series collection $C = \{D^1, ..., D^{|C|}\}$ the number of extracted Envelopes is given by $N = (\sum_i^{|C|} \lfloor \frac{|D^i|}{\ell_{min}+\gamma} \rfloor)$. If w PAA segments are used to discretize the series, each $iSAX$ symbol is represented by a single byte (binary label) and the disk pointer in each Envelope occupies b bytes (in general 8 bytes are used). The final space complexity is $O((2w)bN)$.

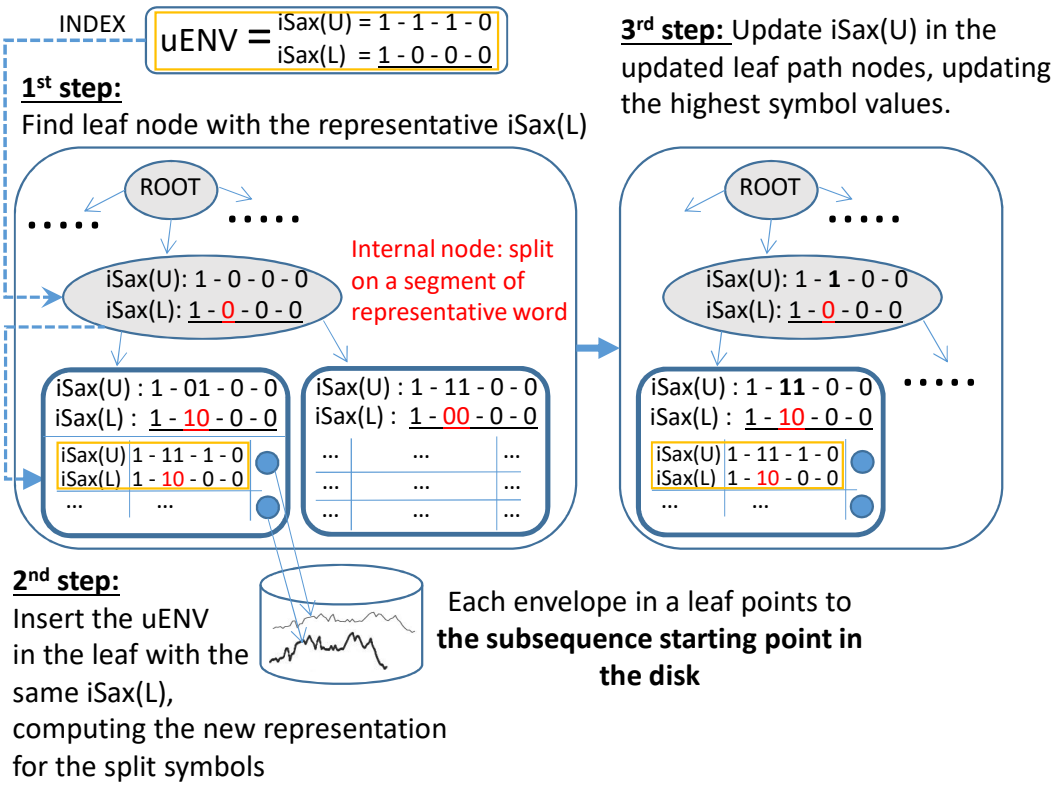


Figure 3.8: Envelope insertion in an *ULISSE* index. $iSAX(L)$ is chosen to accommodate the Envelopes inside the nodes.

3.5 Similarity Search with ULISSE

In this section, we present the building blocks of the similarity search algorithms we developed for the *ULISSE* index, for both the Euclidean and the DTW distances, and both *k-NN* and ϵ-range queries.

We note that the same index structure supports both distance measures. When the query arrives, and depending on the distance measure we have chosen, we use the corresponding lower bounding and real distance formulas. We elaborate on these procedures in the following sections.

3.5.1 Lower Bounding Euclidean Distance

The iSAX representation allows the definition of a distance function, which lower bounds the true Euclidean [90]. This function compares the PAA coefficients of the first data series, against the iSAX breakpoints (values) that delimit the symbol regions of the second data series.

Let $\beta_u(S)$ and $\beta_l(S)$ be the breakpoints of the iSAX symbol S. We can compute the distance between a PAA coefficient and an iSAX region using:

$$distLB(PAA(D)_i, iSAX(D')_i) = \begin{cases} (\beta_u(iSAX(D')_i) - PAA(D)_i)^2 & if \beta_u(iSAX(D')_i) < PAA(D)_i \\ (\beta_l(iSAX(D')_i) - PAA(D)_i)^2 & if \beta_l(iSAX(D')_i) > PAA(D)_i \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

In turn, the lower bounding distance between two equi-length series D,D', represented by w PAA segments and w $iSAX$ symbols, respectively, is defined as:

$$mindist_{PAA\_iSAX}(PAA(D), iSAX(D')) = \sqrt{\frac{|D|}{w}} \sqrt{\sum_{i=1}^{w} distLB(PAA(D)_i, iSAX(D')_i)}. \tag{3.3}$$

We rely on the following proposition [54]:

Proposition 1 *Given two data series D, D', where $|D| = |D'|$, $mindist_{PAA\_iSAX}(PAA(D), iSAX(D')) \leq ED(D, D')$.*

Since our index contains Envelope representations, we need to adapt Equation 3.3, in order to lower bound the distances between a data series Q, which we call query, and a set of subsequences, whose iSAX symbols are described by the Envelope $uENV_{paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}} = [iSAX(L), iSAX(U)]$.

Therefore, given w, the number of PAA coefficients of Q, that are computed using the Envelope PAA segment length s on the longest multiple prefix, we define the following function:

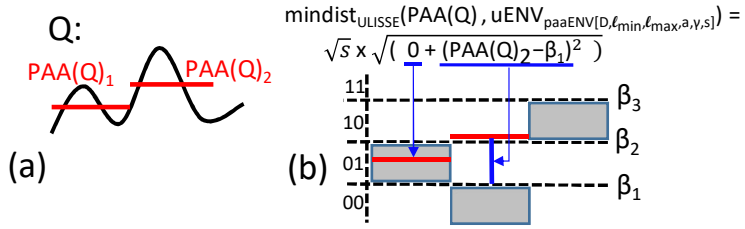


Figure 3.9: Given the PAA representation of a query Q (a) and $uENV_{paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}}$ (b) we compute their $mindist_{ULiSSE}$.

$$mindist_{ULiSSE}(PAA(Q), uENV_{paaENV_{...}}) = \sqrt{s}\sqrt{\sum_{i=1}^{w}\begin{cases}(PAA(Q)_i-\beta_u(iSAX(U)_i))^2, & if(*)\\ (PAA(Q)_i-\beta_u(iSAX(L)_i))^2, & if(**)\\ 0 & \text{otherwise.}\end{cases}}$$
$$(*)\beta_u(iSAX(U)_i)<PAA(Q)_i$$
$$(**)\beta_l(iSAX(L)_i)>PAA(Q)_i \tag{3.4}$$

In Figure 3.9, we report an example of $mindist_{ULiSSE}$ computation between a query Q, represented by its PAA coefficients, and an Envelope in the iSAX space, which is delimited with dashed lines and the relative breakpoints β_i.

Proposition 2 *Given two data series Q,D,* $mindist_{ULiSSE}(PAA(Q), uENV_{paaENV_{[D,\ell_{min},\ell_{max},a,\gamma,s]}}) \leq ED(Q, D_{i,|Q|})$, *for each i such that* $a \leq i \leq a+\gamma+1$ *and* $|D|-(i-1) \geq \ell min$.

Proof 3 *(sketch) We may have two cases, when* $mindist_{ULiSSE}$ *is equal to zero, the proposition clearly holds, since Euclidean distance is non negative. On the other hand, the function yields values greater than zero, if one of the first two branches is true. Let consider the first (the second is symmetric). If we denote with* D'' *the subsequence in D, such that* $\beta_l(iSAX(U)_i) \leq PAA(D'')_i \leq \beta_u(iSAX(U)_i)$, *we know that the upper breakpoint of the* i^{th} *iSAX symbol, of each subsequence in D, which is represented by the Envelope, must be less or equal than* $\beta_u(iSAX(U)_i)$. *It follows that, for this case, Equation 3.4 is equivalent to* $distLB(PAA(Q)_i, iSAX(D'')_i)$, *which yields the shortest lower bounding distance between the* i^{th} *segment of points in D and Q.*

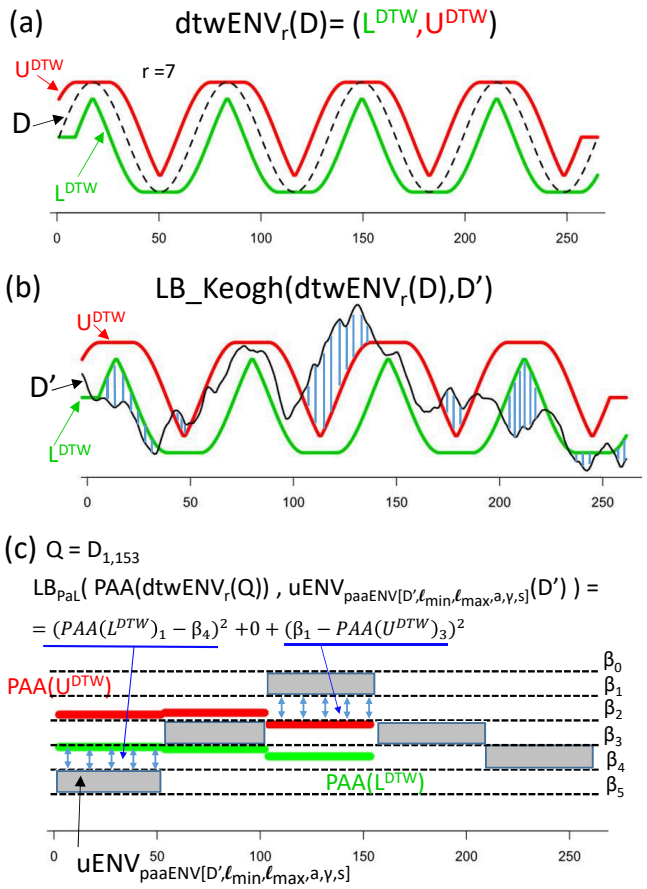


Figure 3.10: *(a)* DTW Envelope (L^{DTW}, U^{DTW}) of a series D. *(b)* LB_{Keogh} distance between DTW Envelope and D'. *(c)* LB_{PaL} between the DTW Envelope of Q (prefix of D) and the *ULISSE* Envelope of D'.

3.5.2 Lower Bounding Dynamic Time Warping

We present here a lower bound for the true DTW distance between two data series. Keogh et al. [46] introduced the LB_{Keogh} function, which provides a measure that is always smaller or equal than the true DTW, between two equi-length series. To compute this measure, we need to account for the valid warping alignments of two data series. Recall that the indexes of a valid path are confined by the Sakoa-Chiba band, where they are at most r points far from the diagonal (Euclidean Distance alignment). Given a data series D, we can build an envelope, $dtwENV_r(D)$, composed by two data series: L^{DTW} and U^{UDTW}, which delimit the space generated by the points of D that have indexes in the valid warping paths, constrained by

the window r. Therefore, the i^{th} point of the two envelope sequences are computed as follows: $L_i^{DTW} = min(D_{(i-r,2r+1)})$ and $U_i^{DTW} = max(D_{(i-r,2r+1)})$. Intuitively, each i^{th} value of L^{DTW} and U^{DTW} represent the minimum and the maximum values, respectively, of the points in D that can be aligned with the i^{th} position of a matching series. In Figure 3.10(a), we report a data series D (plotted using a dashed line), contoured by its $dtwENV_r(D)$ envelope ($r = 7$).

Lower bounding DTW. We can thus define the LB_{Keogh} distance [46], which is computed between a DTW envelope of a series D and a data series D', where $|D| = |D'|$ and the warping window is r:

$$LB_{Keogh}(dtwENV_r(D), D') = \sqrt{\sum_{i=1}^{|D|} \begin{cases} (D'_i - U_i^{DTW})^2, & if D'_i > U_i^{DTW} \\ (D'_i - L_i^{DTW})^2, & if D'_i < L_i^{DTW} \\ s0 & \text{otherwise.} \end{cases}} \tag{3.5}$$

The LB_{Keogh} distance between $dtwENV_r(D)$ and D' is guaranteed to be always smaller than, or equal to $DTW(D, D')$, computed with warping window r. In Figure 3.10(b), we depict the LB_{Keogh} distance between $dtwENV_r(D)$ (from Figure 3.10(a)), and a new series D'. The vertical (blue) lines represent the positive differences between D' and the DTW envelope of D, in Equation 3.5. Note that the computation of LB_{Keogh} takes $O(\ell)$ time (linear), whereas the true DTW computation runs in $O(\ell r)$ time using dynamic programming [46, 78].

Lower bounding DTW in ULISSE. We now propose a new lower bounding measure for the true DTW distance between a data series and all the sequences (of the same length) represented by an *ULISSE* Envelope. To that extent, we first introduce a measure based on LB_{Keogh} distance, which is computed between the PAA representation of $dtwENV_r(D)$ and the $iSAX$ representation of D'. Given w, the number of PAA coefficients of each dtw envelope series (U^{DTW},L^{DTW}) that is equivalent to the number of iSAX coefficients of D', we have:

$$LB_{Keogh_{PAA\_iSAX}}(PAA(dtwENV_r(D)), iSAX(D')) =$$
$$\sqrt{\frac{|D|}{w}} \sqrt{\sum_{i=1}^{w} \begin{cases} (\beta_{\ell}(iSAX(D')_i) - PAA(U^{DTW})_i)^2, & if \quad \beta_{\ell}(iSAX(D')_i) > PAA(U^{DTW})_i) \\ (PAA(L^{DTW})_i - \beta_u(iSAX(D')_i))^2, & if \quad PAA(L^{DTW})_i > \beta_u(iSAX(D')_i) \\ 0 & \text{otherwise.} \end{cases}} \tag{3.6}$$

We know that $LB_{Keogh_{PAA\_iSAX}}(PAA(dtwENV_r(D)),$
$iSAX(D')) \leq LB_{Keogh}(dtwENV_r(D), D')$ as proven by Keogh et al. [46]. Given the PAA representation of $dtwENV_r(D)$ (of w coefficients), and an *ULISSE* Envelope built on D': $uENV_{paaENV_{[D',\ell_{min},\ell_{max},a,\gamma,s]}}) = [L, U]$, we define:

$$LB_{PaL}(PAA(dtwENV_r(D)), uENV_{paaENV[D',...]}) = \sqrt{s}\sqrt{\sum_{i=1}^{w} \begin{cases} (\beta_\ell(iSAX(L)_i) - PAA(U^{DTW})_i)^2, & if \quad \beta_\ell(iSAX(L)_i) > PAA(U^{DTW})_i) \\ (PAA(L^{DTW})_i - \beta_u(iSAX(L)_i))^2, & if \quad PAA(L^{DTW})_i > \beta_u(iSAX(U)_i) \\ 0 & \text{otherwise.} \end{cases}} \tag{3.7}$$

Lemma 3 *Given two data series D and D', where $\ell min \leq |D| \leq \ell max$, the distance $LB_{PaL}(PAA(dtwENV_r(D)), uENV_{paaENV[D',\ell_{min},\ell_{max},a,\gamma,s]})$ is always smaller or equal to $DTW(D, D'_{i,|D|})$, for each i such that $a \leq i \leq a + \gamma + 1$ and $|D'| - (i-1) \geq \ell min$.*

Intuitively, the lemma states that the LB_{PaL} function always provides a measure that is smaller than the true DTW distance between D and each subsequence in D' of the same length, represented by $uENV_{paaENV[D',\ell_{min},\ell_{max},a,\gamma,s]})$.

Proof 4 *(sketch): We want to prove that*

$$LB_{PaL}(PAA(dtwENV_r(D)), uENV_{paaENV[D',\ell_{min},\ell_{max},a,\gamma,s]})$$

is equal to

$$\underset{i}{\text{argmin}}\{LB_{Keogh_{PAA\_iSAX}}(PAA(dtwENV_r(D)), iSAX(D'_{i,|D|}))\},$$

where $D'_{i,|D|}$ is a subsequence of D' represented by $uENV_{paaENV[D',\ell_{min},\ell_{max},a,\gamma,s]})$. The lemma clearly holds if LB_{PaL} yields zero, since the DTW distance between two series is always positive, or equal to zero. We thus test the case, where Equation 3.7 provides a strictly positive value. In the first case, the i^{th} lower iSAX breakpoint of L in the ULISSE Envelope ($\beta_\ell(iSAX(L)_i)$) is greater than the i^{th} PAA coefficient of the U^{DTW}, namely $PAA(U^{DTW})_i$. This implies that any other i^{th} iSAX coefficient, which is contained in the ULISSE Envelope is necessarily greater than $\beta_\ell(iSAX(L)_i)$ and $PAA(U^{DTW})_i$. Hence, the Equation 3.7 is equivalent to the smallest value we can obtain from the first branch of $LB_{Keogh_{PAA\_iSAX}}$ computed

between each i^{th} iSAX coefficient of the subsequences in D' (represented in the $ULISSE$ Envelope) to the i^{th} PAA coefficient of $PAA(U^{DTW})$. $LB_{Keogh_{PAA\_iSAX}}$ always yields a value that is smaller or equal to the true DTW distance, with warping window r.

The second case is symmetric. Here, the $\beta_u(iSAX(L)_i)$ coefficient is the closest to $PAA(L^{DTW})_i$, and greater than any other i^{th} iSAX coefficient of the $ULISSE$ Envelope. Therefore, Equation 3.7 is equivalent to the smallest value we can obtain on the second branch of $LB_{Keogh_{PAA\_iSAX}}$ computed between each i^{th} iSAX coefficient of the subsequences in D' (represented in the $ULISSE$ Envelope) to the i^{th} coefficient of $PAA(L^{DTW})$. ■

In Figure 3.10(c), we depict an example that shows the computation of LB_{PaL} between the DTW Envelope that is built around the prefix of D (*153* points) and the *ULISSE* Envelope of the series D'. For this latter, the settings are: $a = 1$, $\ell_{min} = 153$, $\ell_{max} = 255$, $\gamma = 0$ and $s = 51$. In the figure, we represent the iSAX coefficients of the *ULISSE* Envelope, with (gray) rectangles delimited by their breakpoints (dashed horizontal lines). The coefficients of $PAA(U^{DTW})$ and $PAA(L^{DTW})$ are represented by red and green solid segments.

3.5.3 Approximate search

Similarity search performed on *ULISSE* index relies on Equation 3.4 (Euclidean distance) and Equation 3.7 (DTW distance) to prune the search space. This allows to navigate the tree, visiting the most promising nodes first. We thus provide a fast approximate search procedure we report in Algorithm 4. In Line 7 (or Line 9 if DTW distance is used), we start to push the internal nodes of the index in a priority queue, where the nodes are sorted according to their lower bounding distance to the query. Note that in the comparison, we use the largest prefix of the query, which is a multiple of the PAA segment length, used at the index building stage (Line 1). Recall that when the search is performed using the DTW measure, the PAA representation of the query is computed on the DTW envelope ($dtwENV_r$) of the segment-length multiple that completely contains the query (Line 2). This envelope is composed by two series, which encode the possible warping alignment according the warping window r. Therefore, the PAA representation is composed by two sets of coefficients, e.g., $PAA(L^{DTW})$ and $PAA(U^{DTW})$, as we depict in Figure 3.10.(c). Then, the algorithm pops the ordered nodes from the queue,

visiting their children in the loop of Line 10. In this part, we still maintain the internal nodes ordered (Lines 34-35).

As soon as a leaf node is discovered (Line 12), we check if its lower bound distance to the query is shorter than the *bsf*. If this is verified, the dataset does not contain any data series that are closer than those already compared with the query. In this case, the approximate search result coincides with that of the exact search. Otherwise, we can load the raw data series pointed by the Envelopes in the leaf, which are in turn sorted according to their position, to avoid random disk reads. We visit a leaf only if it contains Envelopes that represent sequences of the same length as the query. Each time we compute either the true Euclidean distance (Line 19) or the true DTW distance ((Line 21)), the best-so-far distance (*bsf*) is updated, along with the R^a vector. Since priority is given to the most promising nodes, we can terminate our visit, when at the end of a leaf visit the k *bsf*'s have not improved (Line 22). Hence, the vector R^a contains the k approximate query answers.

3.5.4 Exact search

Note that the approximate search described above may not visit leaves that contain answers better than the approximate answers already identified, and therefore, it will fail to produce exact, correct results. We now describe an exact nearest neighbor search algorithm, which finds the k sequences with the absolute smallest distances to the query.

In the context of exact search, accessing disk-resident data following the lower bounding distances order may result in several leaf visits: this process can only stop after finding a node, whose lower bounding distance is greater than the *bsf*, guaranteeing the correctness of the results. This would penalize computational time, since performing many random disk I/O might unpredictably degenerate.

We may avoid such a bottleneck by sorting the Envelopes, and in turn the disk accesses. Moreover, we can exploit the *bsf* provided by approximate search, in order to perform a sequential search with pruning over the sorted Envelopes list (this list is stored across the *ULISSE* index). Intuitively, we rely on two aspects. First, the *bsf*, which can translate into a tight-enough bound for pruning the candidate answers. Second, since the list has no hierarchy structure, any Envelope is stored with the highest cardinality available, which guarantees a fine representation of

Algorithm 4: *ULISSE K-nn-Approx*

Input: **int** k, **float** [] Q, ***ULISSE*** **index** I, **int** r `// warping window`
Output: **float** $[k][|Q|]$ R^a, **float** [] *bsf*

```
1  float [] Q* ← PAA(Q_{1,..,⌊|Q|/I.s⌋});
2  float [][] Q*dtw ← PAA(dtwENV_r(Q_{1,..,⌊|Q|/I.s⌋}));
3  float[k] bsf ← {∞, ..., ∞} ;
4  PriorityQueue nodes;
5  foreach node in I.root.children() do
6      if Euclidean distance search then
7          nodes.push(node, mindist_ULiSSE(Q*, node));
8      else if DTW search then
9          nodes.push(node, LB_PaL(Q*dtw, node));
10 while n = nodes.pop() do
11     if n.isLeaf() and n.containsSize(|Q|) then
12         if n.lowerBound < bsf[k] then
               // sort according disk pos.
13             uENV [] Envelopes = sort(n.Envelopes);
               // iterate the Env.  and compute true ED
14             oldBSF ← bsf[k];
15             foreach E in Envelopes do
16                 float [] D ← readSeriesFromDisk(E);
17                 for i ← E.a to min(E.a+E.γ+1,|D| − (|Q| − 1)) do
18                     if Euclidean distance search then
19                         ED_updateBSF(Q, E.D_{i,|Q|}, k, bsf, R^a);
20                     else if DTW search then
21                         DTW_updateBSF(Q, E.D_{i,|Q|}, k, bsf, R^a, r);
               // if bsf has not improved end visit.
22             if oldBSF == bsf[k] then
23                 break;
24         else
25             break; // Approximate search is exact.
26     else
27         LBleft ← 0, LBright ← 0;
28         if Euclidean distance search then
29             LBleft ← mindist_ULISSE(Q*, n.left);
30             LBright ← mindist_ULISSE(Q*, n.right);
31         else if DTW search then
32             LBleft ← LB_PaL(Q*dtw, n.left);
33             LBright ← LB_PaL(Q*dtw, n.right);
34         nodes.push(n.left, LBleft);
35         nodes.push(n.right, LBright);
```

the series, and can contribute to the pruning process.

Algorithm 5 describes the exact search procedure. In the case of Euclidean distance search, in Line 8 we compute the lower bound distance between the Envelope and the query. On the other hand, when DTW distance is used, we compute the lower bound distance in Line 10. If it is not smaller than the k^{th} *bsf*, we do not access the disk, pruning Euclidean Distance computations as well. Note that while we are computing the true distances, we can speed-up computations using the *Early Abandoning* technique [78], which works both for Euclidean and DTW distances. In the case of DTW distance, prior to computing the raw distance, we have a

Algorithm 5: *ULISSE K-nn-Exact*

Input: **int** k, **float** [] Q, ***ULISSE*** **index** I, **int** r `// warping window`
Output: **float** $[k][|Q|]$ R

```
1  float [] Q* ← PAA(Q_{1,..,⌊|Q|/I.s⌋});
2  float [][] Q*dtw ← PAA(dtwENV_r(Q_{1,..,⌊|Q|/I.s⌋}));
3  float [] bsf, float [k][|Q|] R ← K-nn-Approx(k, Q, I) ;
4  if bsf is not exact then
5      foreach E in I.inMemoryList do
6          LBDist← 0;
7          if Euclidean distance search then
8              LBDist←mindist_ULiSSE(Q*, E);
9          else if DTW search then
10             LBDist←LB_PaL(Q*dtw, E);
11         if LBDist< bsf[k] then
12             float [] D ← readSeriesFromDisk(E);
13             for i ← E.a to min(E.a+E.γ+1,|D| − (|Q| − 1)) do
14                 if Euclidean distance search then
15                     ED_updateBSF(Q, D_{i,|Q|}, k,bsf, R);
16                 else if DTW search then
17                     l ← LB_Keogh(dtwENV_r(Q), D_{i,|Q|});
18                     if l <bsf[k] then
19                         DTW_updateBSF(Q, D_{i,|Q|}, k,bsf, R);
```

further possibility to prune computations using the LB_{Keogh} (Equation 3.5) in Line 17. This permits to obtain a lower bounding measure in linear time, avoiding the full DTW calculation.

3.5.5 Complexity of query answering

We provide now the time complexity analysis of query answering with *ULISSE*. Both the approximate and exact query answering time strictly depend on data distribution as shown in [115]. We focus on exact query answering, since approximate is part of it.

Best Case. In the best case, an exact query will visit one leaf at the stage of the approximate search (Algorithm 4), and during the second leaf visit will fulfill the stopping criterion (i.e., the *bsf* distance is smaller than the lower bounding distance between the second leaf and the query). Given the number of the first layer nodes (root nodes) N, the length of the first leaf path L, and its size S, the best case complexity is given by the cost to iterate the first layer node and descend to the leaf keeping the nodes sorted in the heap: $O(w(N + LlogL))$, where w is the number of symbols checked at each lower bounding distance computation. We recall that computing the lower bound of Euclidean or DTW distance has equal

time complexity. Moreover we need to take into account the additional cost of sorting the disk accesses and computing the true distances in the leaf, which is $O(S(logS + \ell_{max}))$ in the case of Euclidean distance, and $O(SlogS + Sr\ell_{max})$ for DTW distance, where r is the warping window length.

Worst Case. The worst case for exact search takes place when at the approximate search stage, the complete set of leaves that we denote with T, need to be visited. This has a cost of $O(w(N+TLlogL))$ plus the cost of computing the true distances, which takes either $O(T(S(logS + \ell_{max})))$ (Euclidean distance), or $O(T(SlogS + Sr\ell_{max}))$ (DTW distance). Note though that this worst case is pathological: for example, when all the series in the dataset are the same straight lines (only sligthly perturbed). Evidently, the very notion of indexing does not make sense in this case, where all the data series look the same. As we show in our experiments on several datasets, in practice, the approximate algorithm always visits a very small number of leaves.

***ULISSE* K-nn Exact complexity.** So far we have considered the exact K-nn search with regards to Algorithm 4 (approximate search). When this algorithm produces approximate answers, providing just an upper bound *bsf*, in order to compute exact answers we must run Algorithm 5 (exact search). The complexity of this procedure is given by the cost of iterating over the Envelopes and computing the *mindist*, which takes $O(Mw)$ time, where M is the total number of Envelopes. Let's denote with V the number of Envelopes, for which the raw data are retrieved from disk and checked. Then, the algorithm takes an additional $O(V\ell_{max})$ time to compute the true Euclidean distances, or $O(Vr\ell_{max})$ to compute the true DTW distances.

3.6 Experimental Evaluation

Setup. All the experiments presented in this section are completely reproducible: the code and datasets we used are available online [94]. We implemented all algorithms (indexing and query answering) in C (compiled with gcc 4.8.2). We ran experiments on an Intel Xeon E5-2403 (4 cores @ 1.9GHz), using the x86_64 GNU/Linux OS environment.

Algorithms. We compare *ULISSE* to *Compact Multi-Resolution Index (CMRI) [36]*, which is the current state-of-the-art index for similarity search with varying-length queries (recall that *CMRI* constructs a limited number of distinct

indexes for series of different lengths). We note though, that in contrast to our approach, *CMRI* can only support non Z-normalized sequences. Furthermore, we compare *ULISSE* to *KV-Match* [18], which is the state-of-the-art indexing technique for ϵ-range queries that support the Euclidean and DTW measures over non Z-normalized sequences (remember that, as we discussed in Section 2.1, for Z-normalized data *KV-Match* only supports exact search for the constrained ϵ-range queries).

In addition, we compare to the current state-of-the-art algorithms for subsequence similarity search, the *UCR suite* [78], and *MASS* [68]. Note that only *UCR suite* works with the Euclidean and DTW measures, whereas *MASS* supports only similarity search using Euclidean distance. These algorithms do not use an index, but are based on optimized serial scans, and are natural competitors, since they can process overlapping subsequences very fast.

Datasets. For the experiments, we used both synthetic and real data. We produced the synthetic datasets with a generator, where a random number is drawn from a Gaussian distribution $N(0, 1)$, then at each time point a new number is drawn from this distribution and added to the value of the last number. This kind of data generation has been extensively used in the past [115], and has been shown to effectively model real-world financial data [22].

The real datasets we used are:

- (GAP), which contains the recording of the global active electric power in France for the period 2006-2008. This dataset is provided by EDF (main electricity supplier in France) [53];
- (CAP), the Cyclic Alternating Pattern dataset, which contains the EEG activity occurring during NREM sleep phase [93];
- (ECG) and (EMG) signals from Stress Recognition in Automobile Drivers [31];
- (ASTRO), which contains data series representing celestial objects [19];
- (SEISMIC), which contains seismic data series, collected from the IRIS Seismic Data Access repository [33].

In our experiments, we test queries of lengths *160-4096* points, since these cover at least 90% of the ranges explored in works about data series indexing in the last two decades [43, 4, 96].

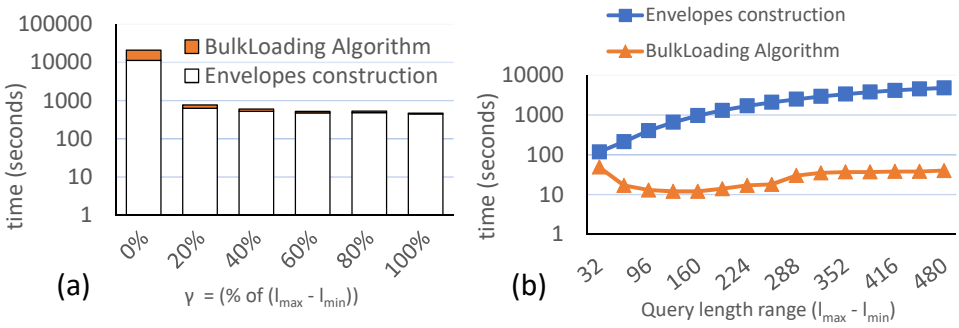


Figure 3.11: (a) Construction and bulk Loading time (log scale) of Envelopes varying γ. (b) Construction and Bulk Loading time (log scale) of Envelopes varying lengths range.

3.6.1 Envelope Building

In the first set of experiments, we analyze the performance of the *ULISSE* indexing algorithm. Note that the indexing algorithm is oblivious to the distance measure used at query time.

In Figure 3.11(a) we report the indexing time (Envelope Building and Bulk loading operations) when varying γ. We use a dataset containing *5M* series of length *256*, fixing $\ell_{min} = 160$ and $\ell_{max} = 256$. Observe that, when $\gamma = 0$, the algorithm needs to extract as many Envelopes as the number of master series of length ℓ_{min}. This generates a significant overhead for the index building process (due to the maximal Envelopes generation), but also does not take into account the contiguous series of same length, in order to compute the statistics needed for Z-normalization. A larger γ speeds-up the Envelope building operation by several orders of magnitude, and this is true for a very wide range of γ values (Figure 3.11(a)). These results mean that the $uENV_{norm}$ building algorithm can achieve good performance in practice, despite its complexity that is quadratic on γ.

In Figure 3.11(b) we report an experiment, where γ is fixed, and the query length range ($\ell_{max} - \ell_{min}$) varies. We use a dataset, with the same size of the previous one, which contains *2.5M* series of length *512*. The results show that increasing the range has a linear impact on the final running time.

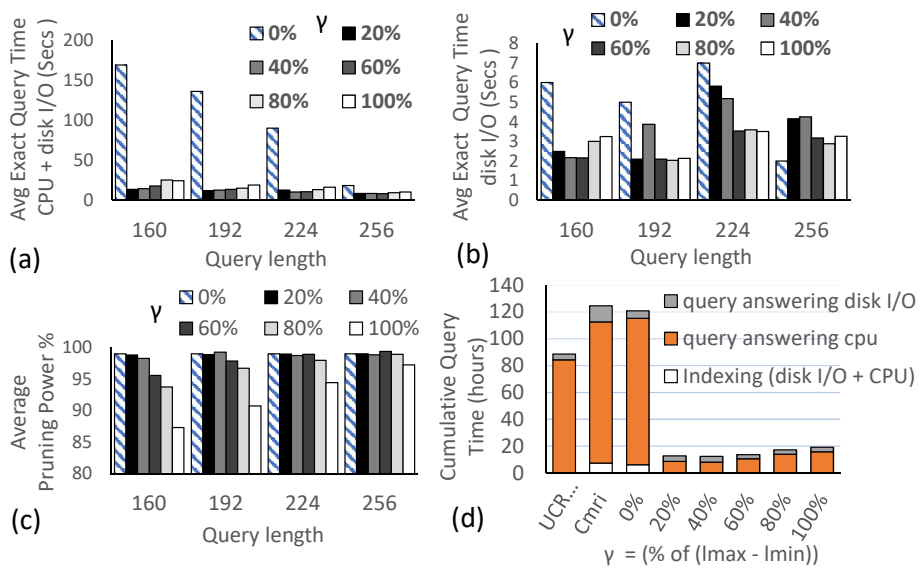
3.6.2 Exact Search Similarity Queries with Euclidean Distance

We now test *ULISSE* on exact 1-Nearest Neighbor queries using Euclidean distance. We have repeated this experiment varying the *ULISSE* parameters along predefined ranges, which are (default in bold) γ : $[0\%, 20\%, 40\%, 60\%, 80\%, \mathbf{100}\%]$, where the percentage is referring to its maximum value, ℓ_{min} : $[96, 128, \mathbf{160}, 192, 224, 256]$, ℓ_{max} : $[256]$, dataset series length (ℓ_S): $[\mathbf{256}, 512, 1024, 1536, 2048, 2560]$ and dataset size of $5GB$. Here, we use synthetic datasets containing random walk data in binary format, where a single point occupies 4 bytes. Hence, in each dataset C, where $|C|^{Bytes}$ denotes the corresponding size in bytes, we have a number of subsequences of length ℓ given by $N^{seq} = (\ell_S - \ell + 1) \times ((|C|^{Bytes}/4)/\ell_S)$. For instance, in a *5GB* dataset, containing series of length 256, we have $\sim$*500 Million* subsequences of length *160*.

We record the average *CPU time*, *query disk I/O time* (time to fetch data from disk: Total time - CPU time), and *pruning power* (percentage of the total number of Envelopes in the index that do not need to be read), of *100* queries, extracted from the datasets with the addition of Gaussian noise. For each index used, the *building time* and the relative *size* are reported. Note that we clear the main memory cache before answering each set of queries. We have conducted our experiments using datasets that are both smaller and larger than the main memory. In all experiments, we report the cumulative running time of *1000* random queries for each query length.

3.6.3 Query Answering Varying γ

We first present results for similarity search queries on *ULISSE* when we vary γ, ranging from *0* to its maximum value, i.e., $\ell_{max} - \ell_{min}$. In Figure 3.12, we report the results concerning non Z-normalized series (for which we can compare to *CMRI*). We observe that grouping contiguous and overlapping subsequences under the same summarization (Envelope) by increasing γ, affects positively the performance of index construction, as well as query answering (Figures 3.12(a) and (d)). The latter may seem counterintuitive, since γ influences in a negative way pruning power, as depicted in Figure 3.12(c). Indeed, inserting more master series into a single *Envelope* is likely to generate large containment areas, which are not tight representations of the data series. On the other hand, it leads to an overall number of *Envelopes* that is several orders of magnitude smaller than the one for



(e)

Method	γ	Indexing time (h)	Indexing size (GB)	No. of Records/ Envelopes	No. of indexes
Cmri	-	7	90	2B	4
Ulisse	**0%**	6	58	485M	1
Ulisse	**20%**	0.2	3	25M	1
Ulisse	**40%**	0.11	1.7	15M	1
Ulisse	**60%**	0.08	1	10M	1
Ulisse	**80%**	0.08	1	10M	1
Ulisse	**100%**	0.04	0.5	5M	1

Figure 3.12: Query answering time performance and pruning power varying γ on non Z-normalized data series.

$\gamma = 0\%$. In this last case, when $\gamma = 0$, the algorithm inserts in the index as many records as the number of master series present in the dataset (*485*M), as reported in (Figure 3.12(e)).

We note that the disk I/O time on compact indexes is not negatively affected at the same ratio of pruning power. On the contrary, in certain cases it becomes faster. For example, the results in Figure 3.12(b) show that for query length 160, the $\gamma = 100\%$ index is more than 2x faster in disk I/O than the $\gamma = 0\%$ index, despite the fact that the latter index has an average pruning power that is 14% higher (Figure 3.12(c)). This behavior is favored by disk caching, which translates to a higher hit ratio for queries with slightly larger disk load. We note that we repeated this experiment several times, with different sets of queries that hit different disk locations, in order to verify this specific behavior. The results showed that this disk I/O trend always holds.

While disk I/O represents on average the $3-4\%$ of the total query cost, computational time significantly affects the query performance. Hence, a compact index, containing a smaller number of *Envelopes*, permits a fast in memory sequential scan, performed by Algorithm 5.

In Figure 3.12(d) we show the cumulative time performance (i.e., $4,000$ queries in total), comparing *ULISSE*, *CMRI*, and *UCR Suite*. Note that in this experiment, *ULISSE* indexing time is negligible w.r.t. the query answering time. *ULISSE*, outperforms both *UCR Suite* and *CMRI*, achieving a speed-up of up to $12x$.

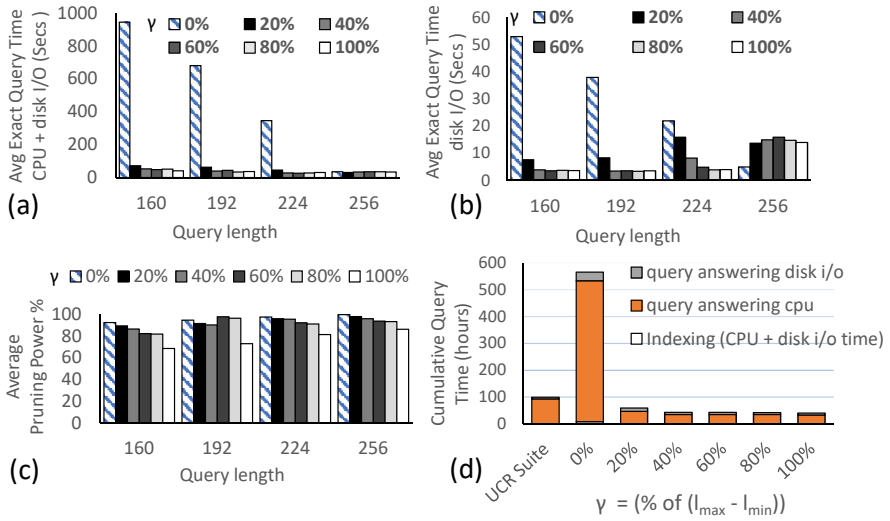
Further analyzing the performance of *CMRI*, we observe that it constructs four indexes (for four different lengths), generating more than $2B$ index records. Consequently, it is clear that the size of these indexes will negatively affect the performance of *CMRI*, even if it achieves reasonable pruning ratios.

These results suggest that the idea of generating multiple copies of an index for different lengths, is not a scalable solution.

In Figure 3.13, we show the results of the previous experiment, when using Z-normalization. We note that in this case the query answering time has an overhead generated by the Z-normalization that is performed on-the-fly, during the similarity search stage. Overall, we observe exactly the same trend as in non Z-normalized query answering. *ULISSE* is still $2x$ faster than the state-of-the-art, namely *UCR Suite*.

Varying Length of Data Series. In this part, we present the results concerning the query answering performance of *ULISSE* and *UCR Suite*, as we vary the length of the sequences in the indexed datasets, as well as the query length (refer to Figure 3.14). In this case, varying the data series length in the collection, leads to a search space growth, in terms of overlapping subsequences, as reported in Figure 3.14(e). This certainly penalizes index creation, due to the inflated number of Envelopes that need to be generated. On the other hand, *UCR Suite* takes advantage of the high overlapping of the subsequences during the in-memory scan. Note that we do not report the results for $CMRI$ in this experiment, since its index building time would take up to *1 day*. In the same amount of time, *ULISSE* answers more than $1,000$ queries.

Observe that in Figures 3.14(a) and (c), *ULISSE* shows better query performance than the UCR suite, growing linearly as the search space gets exponentially larger. This demonstrates that *ULISSE* offers a competitive advantage in terms of pruning



(e)

γ	Indexing time (hours)	Indexing size (GB)	Number of Envelopes
0%	8,05	83,00	485M
20%	0,20	3,70	25M
40%	0,16	2,10	15M
60%	0,14	1,00	10M
80%	0,14	1,00	10M
100%	0,13	0,50	5M

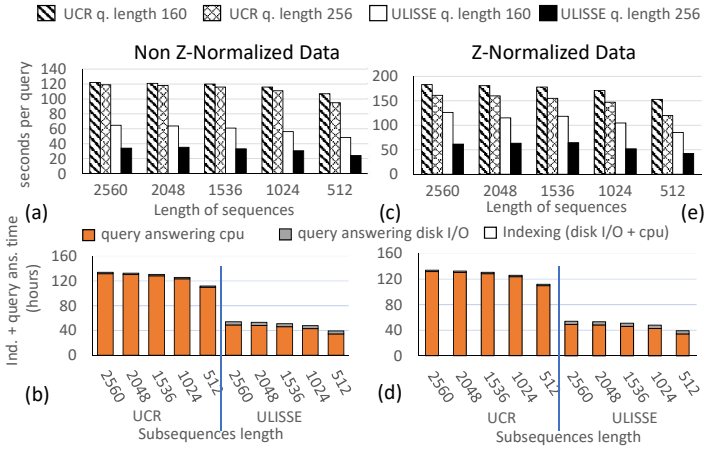
Figure 3.13: Query answering time performance and pruning power varying γ on Z-normalized data series.

the search space that eclipses the pruning techniques *UCR Suite*. The aggregated time for answering $4,000$ queries ($1,000$ for each query length) is 2x for *ULISSE* when compared to *UCR Suite* (Figures 3.14(b) and (d)).

3.6.4 Comparison to Serial Scan Algorithms using Euclidean Distance

We now perform further comparisons to serial scan algorithms, namely, *MASS* and *UCR Suite*, with varying query lengths.

MASS [68] is a recent data series similarity search algorithm that computes the distances between a Z-normalized query of length l and all the Z-normalized overlapping subsequences of a single sequence of length $n \geq l$. *MASS* works by cal-



# subsequences in 5GB			Indexing Ulisse		
Length	160	256	Non Z-norm. time (secs)	Z-norm. time (secs)	#Envelopes
2560	1,200M	1,152M	377	2273	13,5M
2048	1,180M	1,120M	382	2187	13,75M
1536	1,147M	1,067M	392	2106	13,3M
1024	1,081M	961M	355	1977	13,75M
512	882M	642M	307	1445	12.5M

Figure 3.14: Query answering time performance of *ULISSE* and *UCR Suite*, varying the data seres size.

culating the dot products between the query and n overlapping subsequences in frequency domain, in $logn$ time, which then permits to compute each Euclidean distance in constant time. Hence, the time complexity of *MASS* is $O(nlogn)$, and is independent of the data characteristics and the length of the query (l). In contrast, the *UCR Suite* effectiveness of pruning computations may be significantly affected by the data characteristics.

We compared *ULISSE* (using the default parameters), MASS and *UCR Suite* on a dataset containing *5M* data series of length *4096*. In Figure 3.14(f), we report the average query time (CPU + disk/io) of the three algorithms.

We note that MASS, which in some cases is outperformed by *UCR Suite* and *ULISSE*, is strongly penalized, when ran over a high number of non overlapping series. The reason is that, although MASS has a low time complexity of $O(nlogn)$, the Fourier transformations (computed on each subsequence) have a non negligible constant time factor that render the algorithm suitable for computations on very long series.

Varying Range of Query Lengths. In the last experiment of this subsection, we investigate how varying the length range $[\ell min; \ell max]$ affects query answering performance.

In Figure 3.15, we depict the results for Z-normalized sequences. We observe that enlarging the range of query length, influences the number of Envelopes we need to

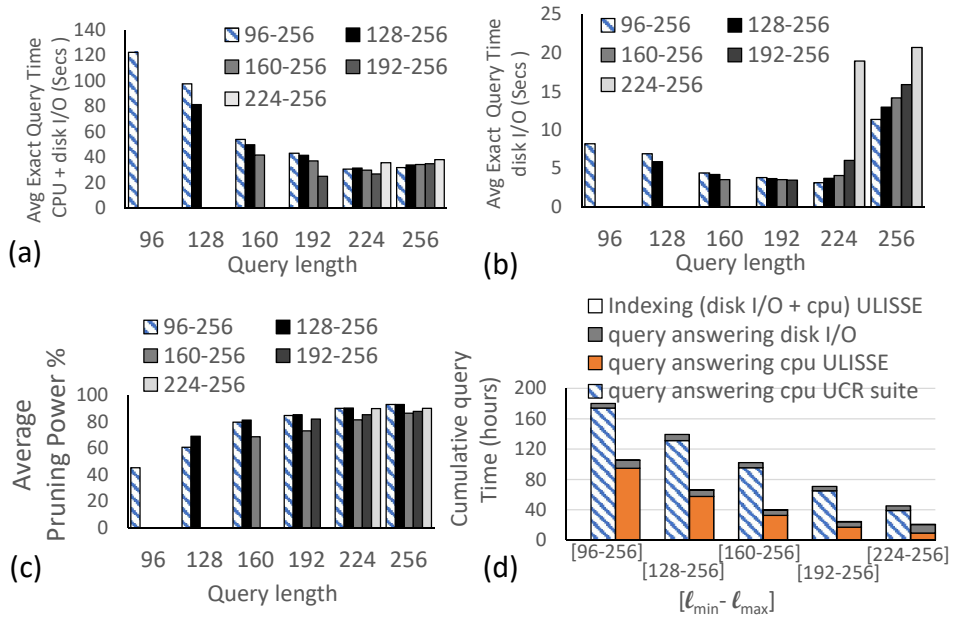


Figure 3.15: Query answering time, varying the range of query length on Z-normalized data series.

accommodate in our index. Moreover, a larger query length range corresponds to a higher number of Series (different normalizations), which the algorithms needs to consider for building a single Envelope (loop of line 16 of Algorithm 2). This leads to large containment areas and in turn, coarse data summarizations. In contrast, Figure 3.15(c) indicates that pruning power slightly improves as query length range increases. This is justified by the higher number of Envelopes generated, when the query length range gets larger. Hence, there is an increased probability to save disk accesses. In Figure 3.15(a) we show the average query time (CPU + disk I/O) on each index, observing that this latter is not significantly affected by the variations in the length range. The same is true when considering only the average query disk I/O time (Figure 3.15(b)), which accounts for $3-4\%$ of the total query cost. We note that the cost remains stable as the query range increases, when the query length varies between *96-192*. For queries of length *224* and *256*, when the range is the smallest possible the disk I/O time increases. This is due to the high pruning power, which translates into a higher rate of cache misses. In Figure 3.15(d), the aggregated time comparison shows *ULISSE* achieving an up to $2x$ speed-up over *UCR Suite*.

In Figure 3.16 we present the results for non Z-normalized sequences, where the same observations hold. Moreover, as we previously mentioned, when Z-

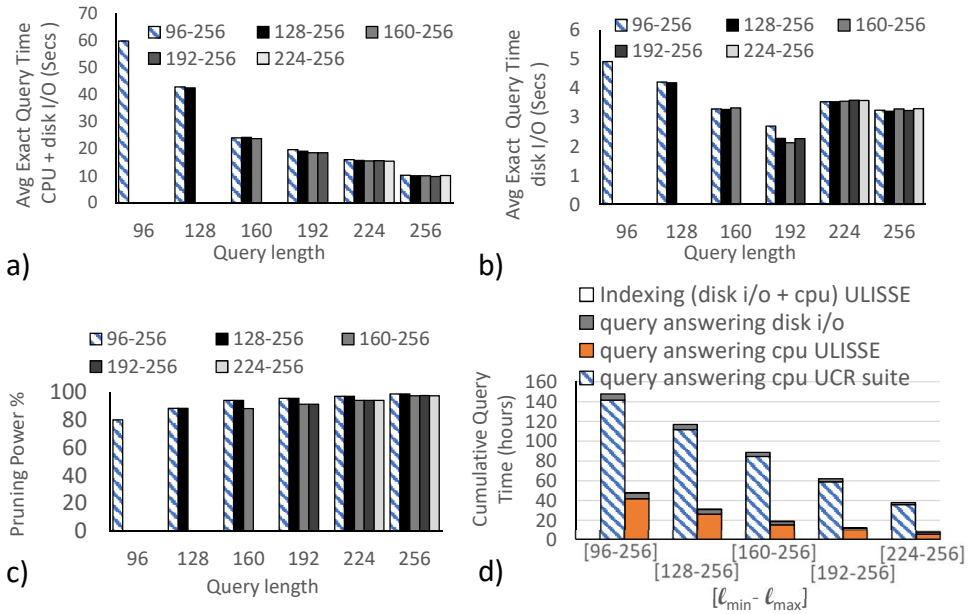


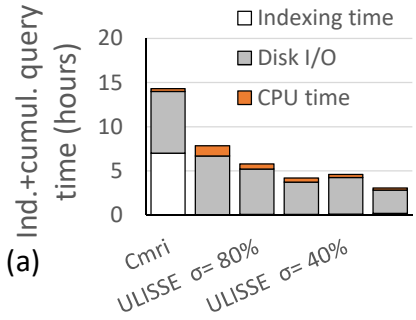
Figure 3.16: Query answering time, varying the range of query length on Z-normalized data series.

normalization is not applied the pruning power slightly increases. This leads ULISSE to a performance up to $3x$ faster than *UCR Suite*.

3.6.5 Approximate Search Similarity Queries with Euclidean Distance

In this subsection, we evaluate *ULISSE* approximate search. Since we compare our approach to CMRI, Z-normalization is not applied. Figure 3.17(a) depicts the cumulative query answering time for $4,000$ queries. As previously, we note that the indexing time for *ULISSE* is relatively very small. On the other hand, the time that CMRI needs for indexing is 2x more than the time during which *ULISSE*s has finished indexing and answering $4,000$ queries.

In Figure 3.17(b), we measure the quality of the Approximate search. In order to do this, we consider the exact query results ranking, showing how the approximate answers are distributed along this rank, which represents the ground truth. We note that CMRI answers have slightly better positions than the *ULISSE* ones. This happens thanks to the tighter representation generated by the complete sliding



(b)

Query length		160	192	224	256
Ranking position	Ulisse				
	1-25	92%	90%	92%	100%
	26-50	8%	10%	8%	0%
	Cmri				
	1-25	100%	100%	100%	100%

Figure 3.17: Approximate query answering on non Z-normalized data series.

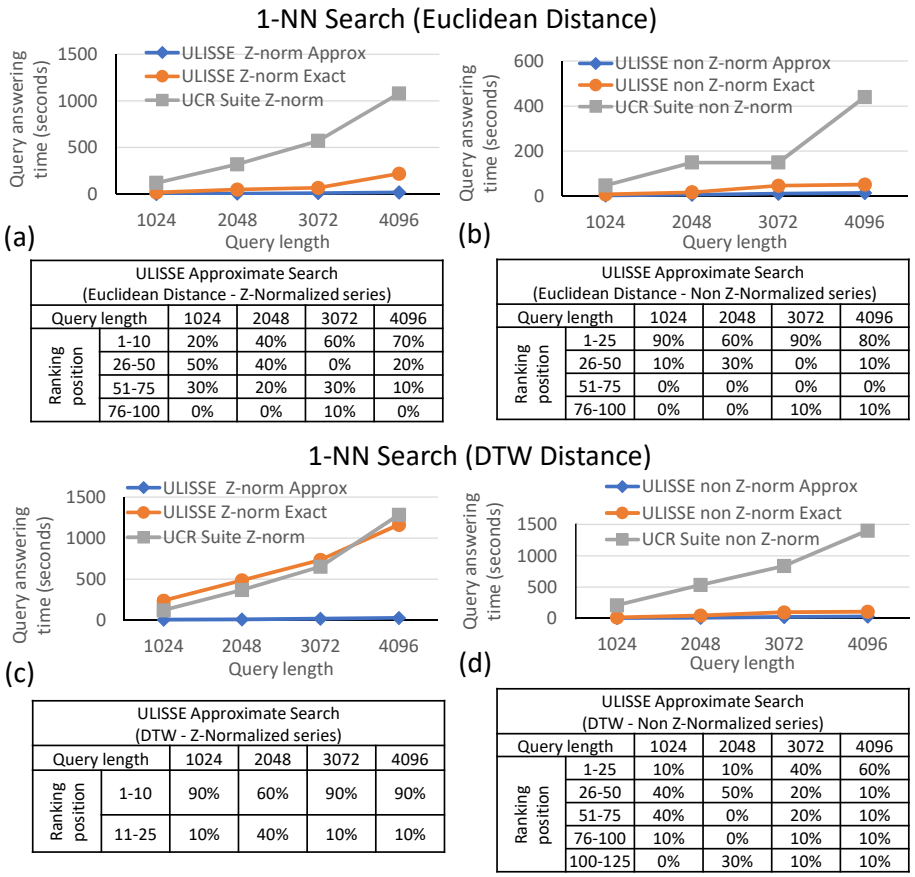
window extraction of each subsequence, employed by CMRI. Nevertheless, this small penalty in precision is balanced out by the considerable time performance gains: *ULISSE* is up to 15x faster than CMRI. When we use a smaller γ, (e.g., 20), *ULISSE* shows its best time performance. This is due to tighter *Envelopes* containment area, which permits to find a better best-so-far with a shorter tree index visit.

3.6.6 Approximate Search Similarity Queries with Euclidean Distance and DTW

Here we evaluate, the time performance of query answering, along with the quality of approximate search. We test the search using both the Euclidean and DTW measures, on a synthetic series composed of $100M$ points. We test a query length range between $\ell_{min} = 1024$ and $\ell_{max} = 4096$. The other parameters are set to their default value.

In Figures 3.18(a) and (b), we report the average query answering time for the Z-normalized and non Z-normalized cases, respectively. The results show that *ULISSE* answers queries up to one order of magnitude faster than *UCR Suite*. Furthermore, we note that *ULISSE* scales better as the query length increases. This shows that our pruning strategy over summarized data, as well as having a good *bsf* approximate answer early on, represent a concrete advantage when pruning the search space.

In Figures 3.18(c) and (d), we report the time performance of query answering with the DTW measure, considering both Z-normalized and non Z-normalized search.



ULISSE Approximate Search (Euclidean Distance - Z-Normalized series)

Ranking position	Query length 1024	2048	3072	4096
1-10	20%	40%	60%	70%
26-50	50%	40%	0%	20%
51-75	30%	20%	30%	10%
76-100	0%	0%	10%	0%

ULISSE Approximate Search (Euclidean Distance - Non Z-Normalized series)

Ranking position	Query length 1024	2048	3072	4096
1-25	90%	60%	90%	80%
26-50	10%	30%	0%	10%
51-75	0%	0%	0%	0%
76-100	0%	0%	10%	10%

ULISSE Approximate Search (DTW - Z-Normalized series)

Ranking position	Query length 1024	2048	3072	4096
1-10	90%	60%	90%	90%
11-25	10%	40%	10%	10%

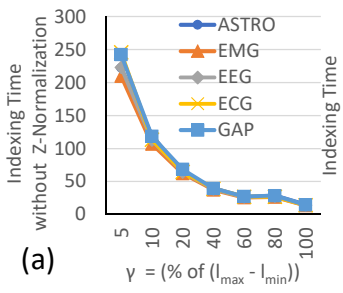
ULISSE Approximate Search (DTW - Non Z-Normalized series)

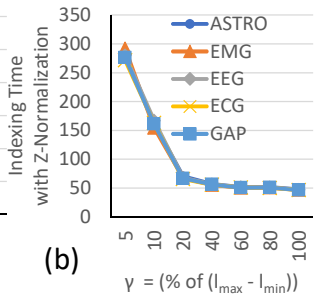
Ranking position	Query length 1024	2048	3072	4096
1-25	10%	10%	40%	60%
26-50	40%	50%	20%	10%
51-75	40%	0%	20%	10%
76-100	10%	0%	10%	10%
100-125	0%	30%	10%	10%

Figure 3.18: Average query answering and approximate quality varying query length.

In Figure 3.18(c), we observe that *ULISSE* answers queries slightly slower than *UCR Suite*, for three of the query lengths. This behavior is explained by the fact that the (overlapping) subsequences represented by the Envelopes have a total size $\sim$ 43x bigger than the original data points. In this case, the pruning power does not mitigate this disadvantage.

Overall, the results show that *ULISSE* is a scalable solution. Moreover, the approximate search, which in this experiment does not visit more than 5 leaves in the tree, represents a very fast solution, approximating well the exact answer (refer to the tables below each plot of Figure 3.18).





(c)

γ(%)	Number of Envelopes Generated
5	9,600,000
10	4,800,000
20	2,400,000
40	1,200,000
60	800,000
80	600,000
100	500,000

Figure 3.19: Indexing time of five real datasets (ASTRO, EMG, EEG, ECG, GAP) varying the number of master series in the Envelope (γ).

3.6.7 Experiments with Real Datasets

In this part, we discuss the results of indexing and query answering performed on real datasets. Here, we also consider the use of the Dynamic Time Warping (DTW) distance measure, along with Euclidean distance.

We start the evaluation by considering five different real datasets that fit the main memory. In the next sections, we will additionally consider real data series collections that do not fit in the available main memory. The objective of this experiment is to firstly assess the benefit of maximizing the number of subsequences represented by a *ULISSE* Envelope on query answering time. Moreover, we want to analyze the impact of the DTW measure on query time performance.

Indexing. For this experiment, we used five real datasets, where each one contains *500K* data series of length 256 (ASTRO, EMG, EEG, ECG, GAP). We show in Figure 3.19.(a,b) the indexing time performance, varying γ for both non Z-Normalized and Normalized sequences. Recall that γ is expressed as the percentage of the maximum number of master series that is $\ell_{max} - \ell_{min}$. The results confirm the trend depicted in Figure 3.11, where the time of building *ULISSE* Envelopes that contain all the master series of each series is one order of magnitude smaller than the time of building the most compact Envelopes, obtained with $\gamma = 5\%$. We also note that the overhead generated by the Z-normalization operations, which have an additional γ factor in the time complexity of the indexing algorithm, is amortized by the generation of $\sim$ 20x less Envelopes in the index, as depicted in Figure 3.19(c).

Query Answering with Euclidean Distance. We report in Figure 3.20 the

results obtained for *1-NN* search over Z-normalized sequences, with Euclidean distance. All parameters are set to their default values. Therefore, in these experiments we used queries of length between $\ell_{min} = 160$ and $\ell_{max} = 256$; the series in the datasets have length 256. In Figure 3.20(a), we report the query pruning power as the number of master series (γ) in each Envelope varies. As expected, we can prune less candidates when the Envelopes contain more sequences. Recall that when a candidate (subsequence) is pruned, the search does not consider its raw values, thus avoiding both Z-normalization and Euclidean distance computations. If a candidate is not pruned, the search can abandon the computations earlier, when the running Euclidean Distance is greater than the k^{th} *bsf* distance.

In Figure 3.20(b), we report the average *abandoning power*, which measures the percentage of the total number of real Euclidean distance computations that are *not* performed. When the search processes an increased number of overlapping subsequences, we expect a decrease in the number of computations performed. We note that the search avoids computations when the Envelopes contain a large number of subsequences, namely, as γ increases.

In Figure 3.20(c), we report the average query time varying γ. We obtain the highest speed-up, with the most compact index (largest γ value), which is more that 2x faster than the state-of-the-art (*UCR Suite* algorithm). This confirms the trend we observed in the previous results conducted over synthetic data. We report the average query time for each dataset in Figure 3.20(d), and for each query length in Figure 3.20(e). In Figure 3.20(f), we show the average number of Euclidean distance and lower bound computations performed by *ULISSE* ($\gamma = 100\%$) and *UCR Suite*, as the query length varies (this corresponds to the average number of points on which the distance to the query is computed), as well as the number of points that are loaded from disk and Z-normalized (this corresponds to the overhead generated by the Z-normalization operations). The goal of this experiment is to quantify the overall benefit of *ULISSE* pruning and abandoning power. (Recall that *UCR Suite* does not perform any lower bound distance computations when using the Euclidean distance.)

First, we observe that *ULISSE* performs half of the Euclidean distance computations of *UCR Suite*, and considers up to seven time less points for the Z-normalization phase. Furthermore, we note that the computation of lower bound distances has a negligible impact on the query workload, especially when the query length is smaller than the length of the series in the dataset (256), in which case the number of candidate subsequences can be orders of magnitude more.

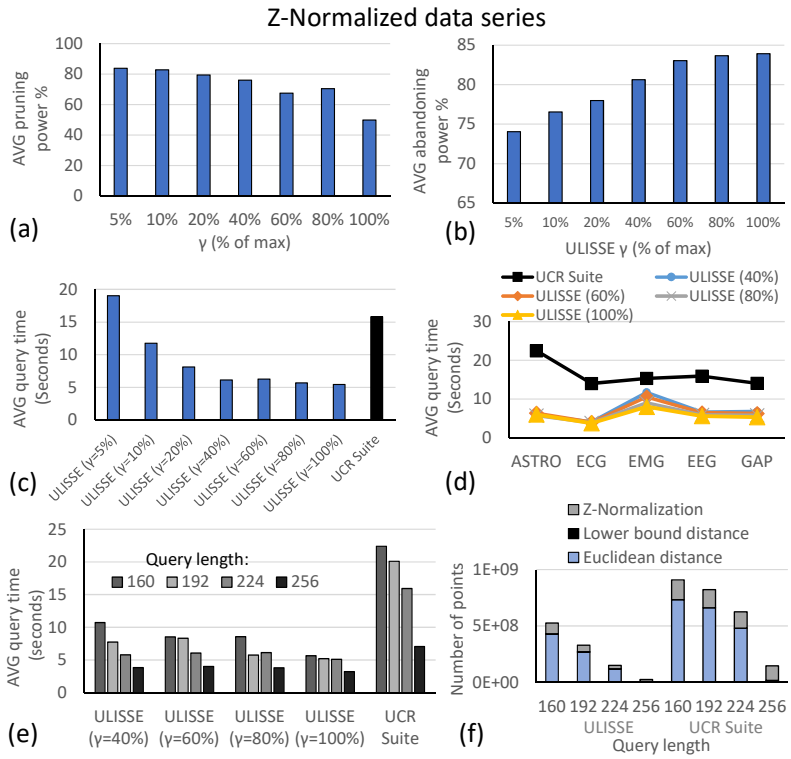


Figure 3.20: Exact (Z-normalized) query answering and pruning power, with Euclidean distance on real datasets.

In Figure 3.21, we depict the results of query answering, without the use of Z-normalization. In this case, the results exhibit a small difference in terms of absolute pruning power values, which is higher when the search is performed on absolute series values. The average query answering time maintains the same trend we observe in Z-normalized query answering. On average, *ULISSE* has a 3x speed-up factor when compared to *UCR Suite*.

Query Answering with DTW Distance. We now report the results of query answering using the DTW measure (Figure 3.22). For this experiment, we used the default parameter settings, and the same real datasets considered in the previous two experiments. We study the efficiency of query answering (*1-NN* query), which uses the DTW lower bounding measures to prune the search space.

In Figure 3.22(a) we report the average pruning power, when varying the DTW warping windows from 5% to 15% of the subsequence length. (These values for the warping window have commonly been used in the literature [46].) We vary γ

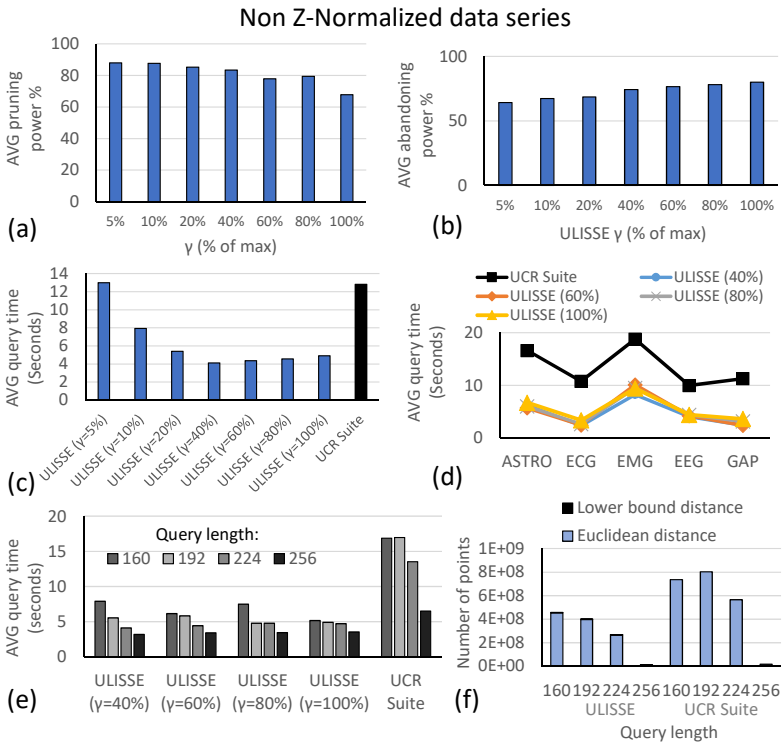


Figure 3.21: Exact (Non Z-normalized) query answering and pruning power, with Euclidean distance on real datasets.

between 60% and 100% of its maximum value, which give the best running time in this experiment. To avoid an unnecessary overload in the plot, we omit the results for γ smaller than 60%.

Once again, we note that the pruning power is negatively affected by the size of the Envelope (γ), and under DTW search the abandoning power slightly decreases as the *gamma* and the *warping* window get larger (see Figure 3.22(b)). This suggests that the DTW lower bound measure we propose is more sensitive than the one used for Euclidean Distance. Nevertheless, in the worst case *ULISSE* is still able to prune 20% of the candidates, and to abandon more than 80% of the *DTW* computations on raw values.

In Figure 3.22(c) we report the average query answering time varying γ, and in Figures 3.22(d) and (e) the average time for each dataset and for different query lengths, respectively, for $\gamma = 100\%$. For these last two experiments, we observe no significant difference for the other values of γ we tested.

We first note that, despite the loss of pruning power of ULISSE when increasing γ, the query answering time is not significantly affected (refer to Figures 3.22(c) and (e)). As in the case of Euclidean distance search, the compactness of the *ULISSE* index plays a fundamental role in determining the query time performance, along with the pruning and abandoning power.

In Figure 3.22(d), we note that only in the ECG and GAP datasets, enlarging the warping window has a substantial negative effect on query time ($2x$ slower), whereas in the other datasets, and in the worst case the time loss is equivalent to 10%.

In Figure 3.22(e), we report the average query workload of *ULISSE* and *UCR Suite*. In contrast to Euclidean distance queries, we notice that the largest amount of work corresponds to lower bounding distance computations. Recall that *ULISSE* prunes the search space in two stages: first comparing the query and the data in their summarized versions using LB_{PaL} (Equation 3.7), and then computing in linear time the LB_{Keogh} between the query and the non pruned candidates. In the worst case, the DTW distance point-wise computation are 10% of those performed for calculating the Lower Bound (query length 160). In general, the total number of points considered for the whole workload is up to 5x smaller than for *UCR Suite*. We note that the pruning strategy of *UCR Suite* is still very competitive, since it avoids a high number of true distance computations using the LB_{Keogh} lower bound. Nonetheless, it has to compute the lower bound distance on the entire set of candidates. The pruning strategy implemented in *ULISSE* permits to achieve up to $10x$ speedup over *UCR Suite*.

In Figure 3.23, we report the results of DTW search, without the application of Z-Normalization. Also in this case, we note that the average pruning power of *ULISSE* is higher than the one we previously observed in the Z-normalized search (Figure 3.23(a)). On the other hand, the average abandoning power is less effective, as shown in Figure 3.23(b). As a consequence, we can see that the *ULISSE* search performs more DTW distance computations (refer to Figure 3.23(c)). Nevertheless, Figure 3.23(e) shows that on average *ULISSE* is up to 10x faster than *UCR Suite*, for all query lengths we tested.

Query over Large datasets with Euclidean Distance. Here, we test *ULISSE* on three large synthetic datasets of sizes *100*GB, *500*GB, and *750*GB, as well as on two real series collections, i.e., ASTRO and SEISMIC (described earlier). The other parameters are the default ones. For each generated index and for the *UCR Suite*, we ran a set of 100 queries, for which we report the average exact search

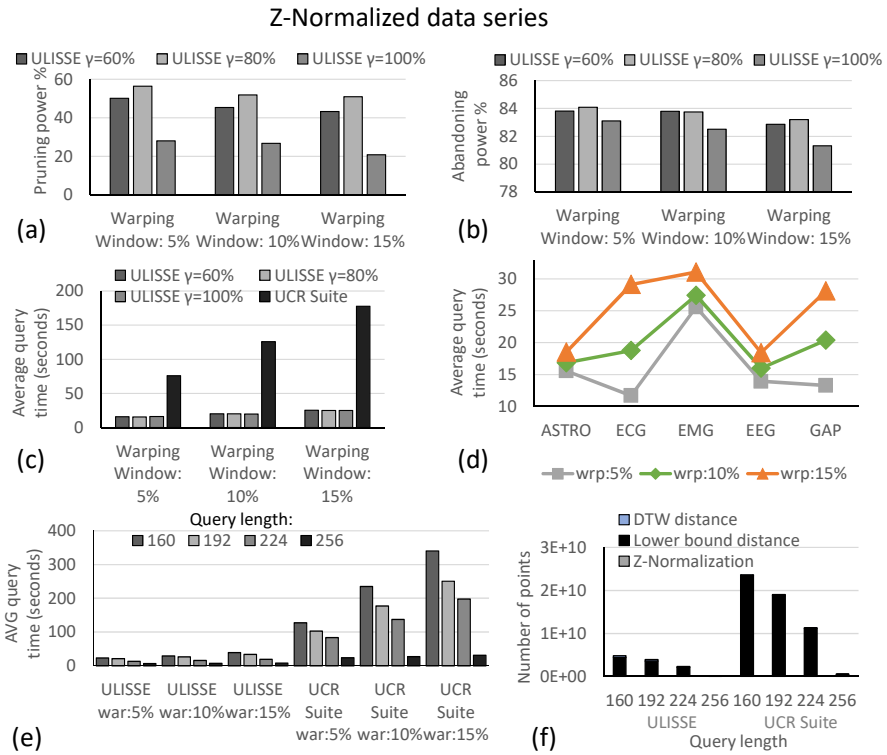


Figure 3.22: Exact (Z-normalized) query answering and pruning power using DTW measure on real datasets.

time.

In Figure 3.24(a) we report the average query answering time (*1-NN*) on synthetic datasets, varying the query length. These results demonstrate that *ULISSE* scales better than *UCR Suite* across all query lengths, being up to 5x faster.

In Figure 3.24(b), we report the *k-NN* exact search time performance, varying k and picking the smallest query length, namely *160*. Note that, this is the largest search space we consider in these datasets, since each query has *9.7* billion of possible candidates (subsequences of length *160*). The experimental results on real datasets confirm the superiority of *ULISSE*, which scales with stable performance, also when increasing the number k of nearest neighbors. Once again it is up to 5x faster than *UCR Suite*, whose performance deteriorates as k gets larger.

In Figure 3.24(c) we report the number of disk accesses of the queries considered

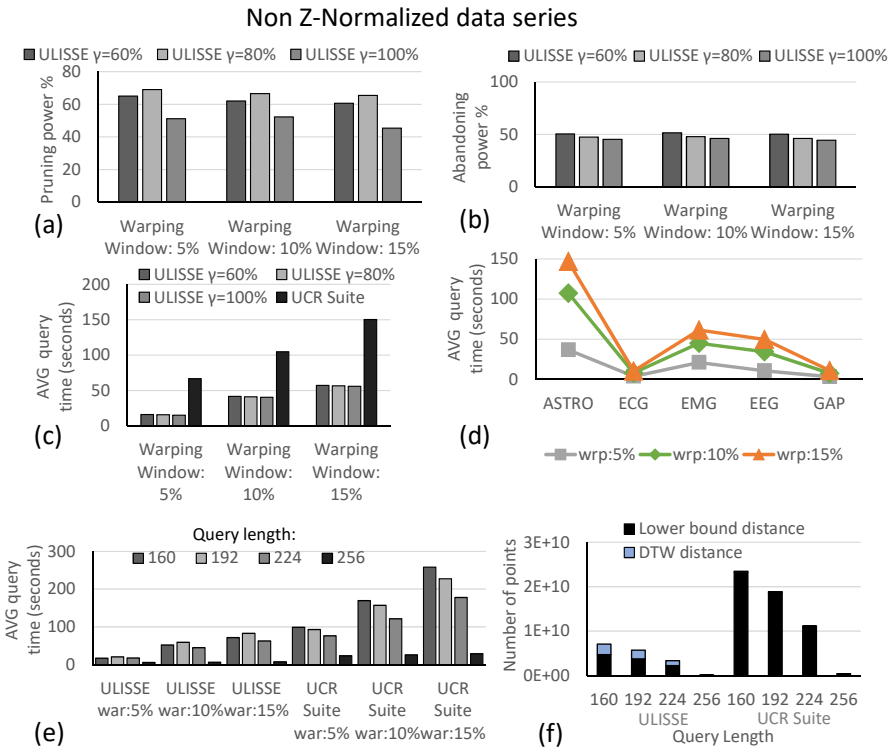
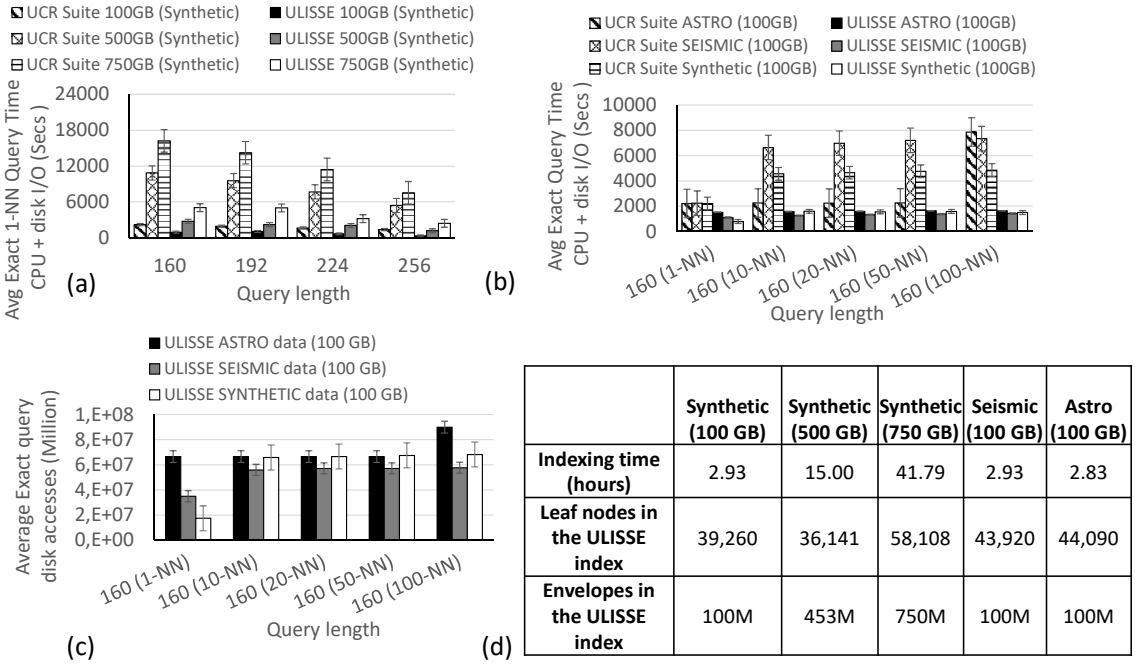


Figure 3.23: Exact (Non Z-normalized) query answering and pruning power using DTW measure on real datasets.

in Figure 3.24(b). Here, we are counting the number of times that we follow a pointer from an envelope to the raw data on disk, during the sequential scan in Algorithm 5. Note that the number of disk accesses is bounded by the total number of Envelopes, which are reported in Figure 3.24(d) (along with the number of leaves and the building time for each index).

We observe that in the worst case, which takes place for the ASTRO dataset for $k = 100$, we retrieve from disk $\sim$*82%* of the total number of subsequences. This still guarantees a remarkable speed-up over *UCR Suite*, which needs to consider all the raw series.

Moreover, since *ULISSE* can use Early Abandoning during exact query answering, we observe during our empirical evaluation that disposing of the approximate answer distance prior the start of the exact search, permits to abandon on average *20%* of points more than *UCR Suite* for the same query.



(d)

	Synthetic (100 GB)	Synthetic (500 GB)	Synthetic (750 GB)	Seismic (100 GB)	Astro (100 GB)
Indexing time (hours)	2.93	15.00	41.79	2.93	2.83
Leaf nodes in the ULISSE index	39,260	36,141	58,108	43,920	44,090
Envelopes in the ULISSE index	100M	453M	750M	100M	100M

Figure 3.24: Exact and Approximate similarity search on Z-normalized synthetic and real datasets.

Query over Large datasets with DTW. We conclude this part of the evaluation reporting the results of query answering on large datasets using the DTW distance.

In Figure 3.25, we report the time performance of (*1-NN* search) on the ASTRO, SEISMIC and synthetic datasets, each one containing $100M$ data series of length 256 (100GB). Also in this case, *ULISSE* guarantees a consistent speed-up over *UCR Suite*, which is at least $\sim$ 1.5x faster in the worst case (ASTRO dataset, query length 160), and up to one order of magnitude faster (synthetic dataset, query length 256).

3.6.8 ϵ-Range Queries

In this last part, we test the *ULISSE* search algorithm for the ϵ-Range query task. To that extent we adapted Algorithm 5, so that given as input $\epsilon \in \mathbb{R}$, it computes the set of subsequences that have a distance to the query smaller than or equal to ϵ. Similarly, we also adapted the *UCR Suite* algorithm to support ϵ-Range

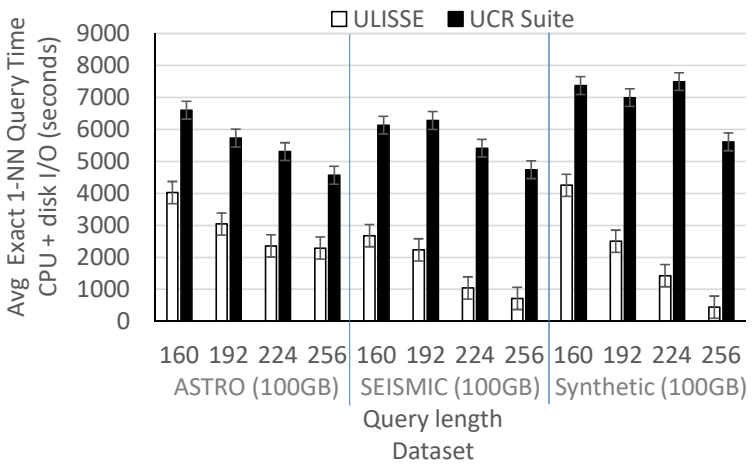


Figure 3.25: Average exact query time with DTW distance (CPU + disk I/O) on real and synthetic datasets.

search. As a third competitor, we consider *KV-Match*, which is the state-of-the art index-based solution for exact ϵ-Range queries on non Z-normalized data series.

In this experiment, we used five different real datasets, composed by a single data series of different lengths, as reported in Figure 3.26(a). For each of these datasets, we can see that *ULISSE* builds its index 5 times faster than *KV-Match*. This is because *KV-Match* is based on the construction of multiple indexes. Specifically, it builds different indexes performing a sliding windows extraction at different lengths. At query answering time, *KV-Match* performs a recombination of query answers coming from the different indexes.

For our ϵ-Range queries, we set the ϵ parameter to twice the *NN* distance of each query. In this manner, we simulate an exploratory analysis task. We report the average value of query selectivity in Figure 3.26(b). We note that in the ECG dataset the selectivity is very high. This is due to the periodic/cyclical nature of this kind of data, which contain repeating heartbeats subsequences that are very similar. In the other datasets, we have different values of selectivity ranging from 0.5% to 15%, when using Euclidean distance. On the other hand, when the DTW measure is considered, we observe a significant increase of the answer-set cardinality.

In Figures 3.26(c) and (d), we show the average query answering time for Euclidean distance, when varying the query length and the dataset, respectively.

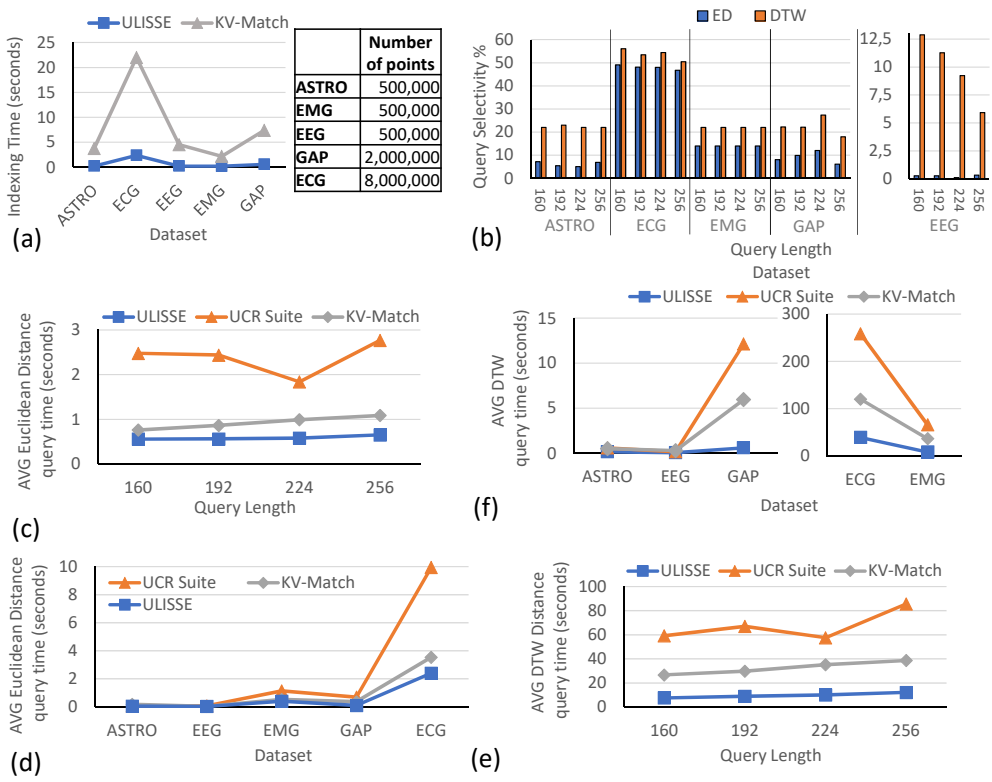


Figure 3.26: Results of ϵ-range search on non Z-normalized real datasets.

We note that in this case *ULISSE* and *KV-match* have no substantial difference in their time performance. However, when we consider the DTW distance, *ULISSE* becomes up to one order of magnitude faster than *KV-Match* (see Figures 3.26(e) and (f)). This difference becomes pronounced for the two largest datasets: *ULISSE* is 3x faster for ECG, and 10x faster for GAP. It is important to note that since *KV-Match* needs to recombine the answers from the different index structures, its time performance is affected by this refinement phase of query answering, and is rather sensitive to dataset size and query selectivity.

3.7 Conclusions

Similarity search is one of the fundamental operations for several data series analysis tasks. Even though much effort has been dedicated to the development of indexing techniques that can speed up similarity search, all existing solutions are limited by the fact that they can only support queries of a fixed length.

In this chapter, we proposed *ULISSE*, the first index able to answer similarity search queries of variable-length, over both Z-normalized and non Z-normalized sequences, supporting the Euclidean and DTW distances, for answering exactly, or approximately both k-*NN* and ϵ-range queries. We experimentally evaluated, our indexing and similarity search algorithms, on synthetic and real datasets, demonstrating the effectiveness and efficiency (in space and time cost) of the proposed solution.

Chapter 4

VALS: Scalable VAriable-Length Similarity Search Suite

Data series similarity search is an important operation and at the core of several analysis tasks and applications related to data series collections. In this chapter, we present VALS, our prototype system designed to support similarity search queries of *variable length*. VALS employs the *ULISSE* index in order to allow users to interactively run and explore the results of approximate and exact subsequence similarity search in both non Z-normalized and Z-normalized large data series collections.

4.1 The VALS System

VALS is the first system that supports scalable VAriable-Length Similarity search in long data series. In detail, this prototype system implements the *ULISSE* indexing and search algorithms, we presented in Chapter 3. We describe here the architecture and GUI of VALS, shown in Figure 4.1.

[**Collections Import and Indexing**] VALS permits to easily import the data series collections to analyze. Users can index a dataset using the ULISSE algorithm. They can also use the indexing techniques provided by the competitors (CMRI). On the other hand, if the user does not build an index on top of a dataset, the system will apply by default the UCR Suite algorithm. We show in Figure 4.2,

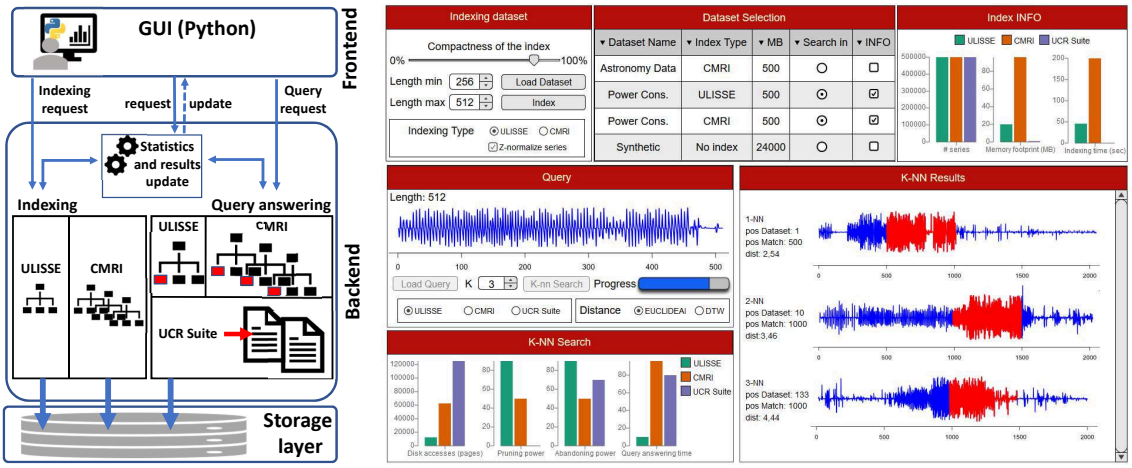


Figure 4.1: (*left*) VALS architecture. (*right*) A screen-shot of the VALS GUI during a ULISSE K-NN search.

the indexing parameters setting in the VALS GUI.

Once the indexes are created, our system also permits to compare the different properties of the generated indexes, such as storage, main memory consumption, and index building time. In Figure 4.3 we show the indexes comparison performed on the VALS GUI.

[**Query Answering and Evaluation**] The users, with the aid of the VALS GUI can load and issue a k-NN search query. We depict in Figure 4.4 the query loading box of VALS. Note that, when the query is loaded in the bottom-left area, the user can visualize the performance of different similarity search approaches priorly used to answer the same query.

While the search executes using the ULISSE Index, the application shows the approximate answers results in the central panel. In Figure 4.5 we depict the query result panel update, during the search operation. Note that, until the search is over the results shown are guaranteed to be approximate. At the same time, various performance indicators (i.e., number of disk accesses, pruning power, number of Euclidean Distance computations, and time) are shown, as well, and are compared to those of the competitors.

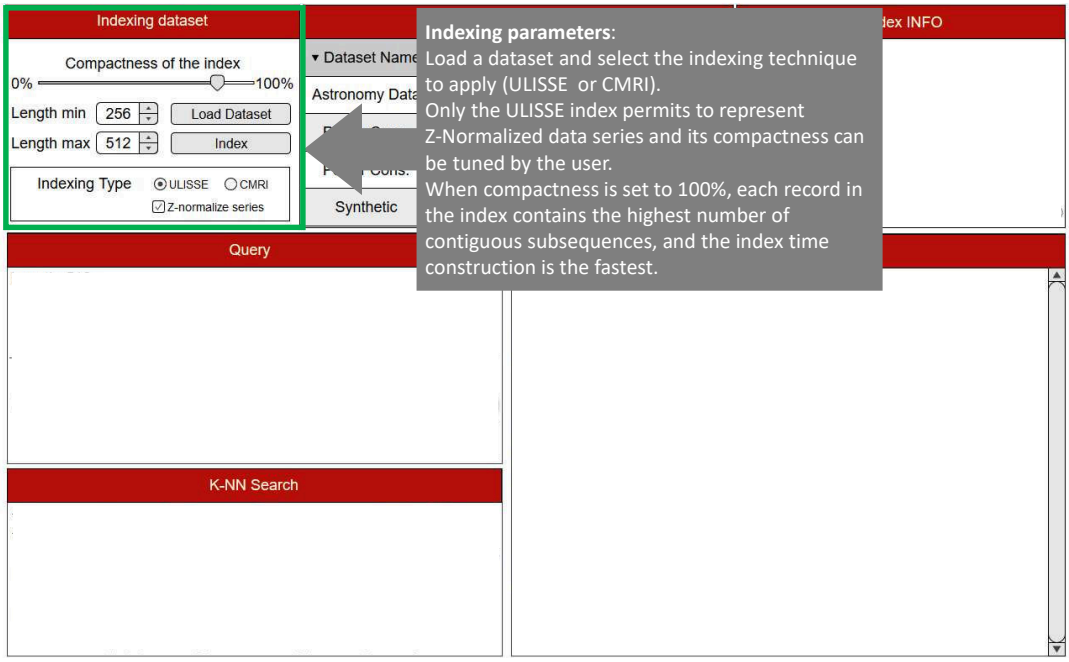


Figure 4.2: VALS indexing parameters settings.

Once the search is terminated, the final (exact) results and the complete statistics are available on the VALS GUI, as we report in Figure 4.6.

[**Implementation**] The GUI interface of VALS is implemented in Python, which disposes of a wrapper that interacts with the Indexing and Query answering algorithms (implemented in C/C++). A separate thread interacts with the algorithms in order to update all the statistics and the query results in real time.

4.2 Prototype Functionality

The objective of this prototype is twofold: first, to underline the importance of the variable length similarity search, and second, to reveal to users the details and features of the VALS engine. We use several real datasets from different domains (such as Energy Consumption analysis, Healthcare, Astrophysics and Seismology). Below, we list the scenarios that showcase the functionality of VALS.

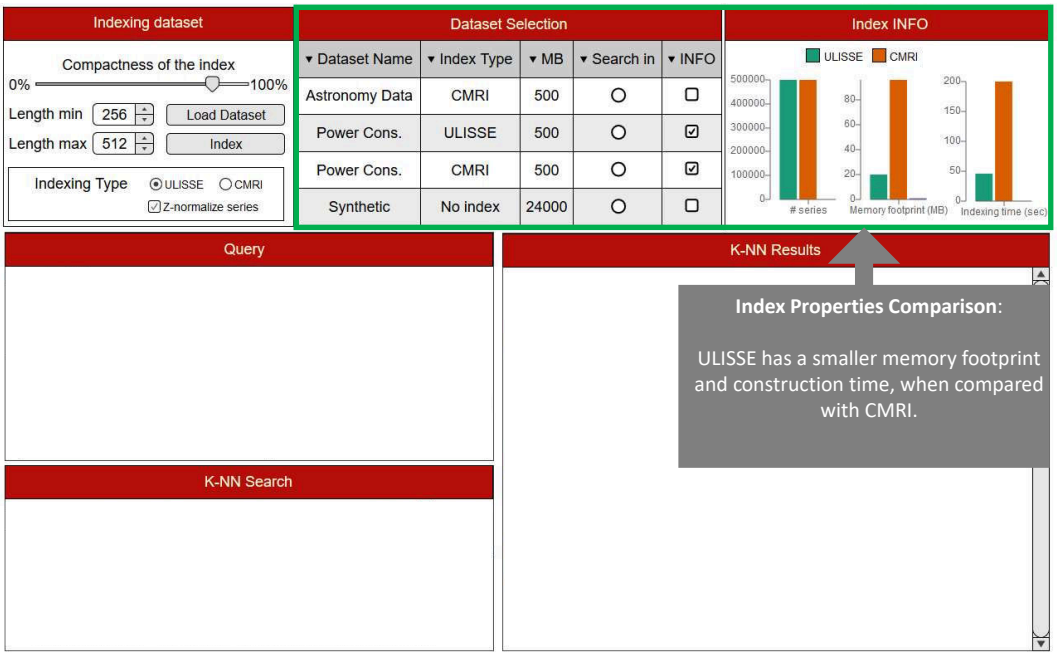


Figure 4.3: VALS indexes properties comparison.

1. Indexing Performance. The users can test the ULISSE and CMRI indexes construction over different datasets and configurations, and observe the memory footprint (ratio of raw data and index size) and index construction time. It will be obvious that ULISSE scales better than CMRI, for various length ranges, while at the same time occupying less space, which the users control by the γ parameter (specifying the number of contiguous series (master series) represented by the same index record, namely the envelope).

2. Variable-Length Query Answering.

With VALS, the users can experience the flexibility of a variable-length index. The users can issue queries of different lengths, and get richer insights from the data, obtained from even slight variations in the query length. By executing several queries on the different real datasets, the participants will realize the limitations that fixed query length search poses to data exploration, by missing valuable results.

3. Query Answering Performance. With VALS, the users can also monitor

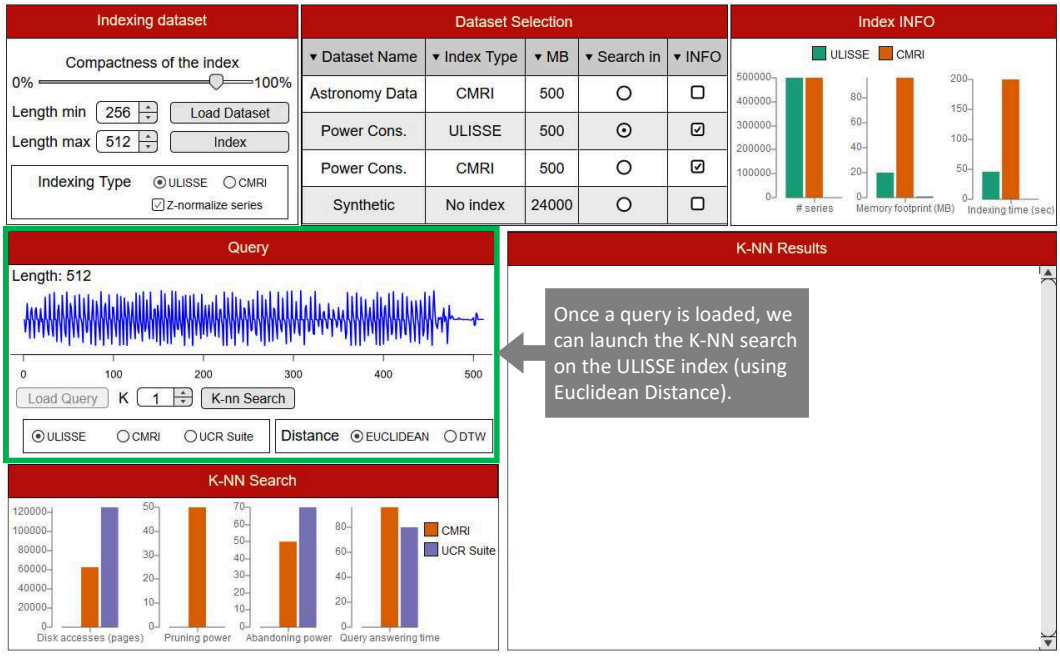


Figure 4.4: VALS query loading and performance visualization.

and analyze the performance of query answering. Before issuing a query, users can select the distance measure, as well as the number of nearest neighbor the query has to return. The users can then inspect the information and statistics on the query execution provided by the VALS interface, and they will observe the influence of the different configuration settings of each index on query answering performance. Specifically, they can notice that increasing the compression of the ULISSE index, permits to reduce the disk accesses, thanks to the smaller search space generated by the index. This in turn has a positive impact on the pruning capability of the index. Moreover, users can notice the interactive response times of ULISSE approximate query answering. The VALS interface will also report the percentage of saved Euclidean and DTW distance computations (marked as *abandoning power*), as well as the *pruning power* of the selected query answering method, and the number of *disk accesses* performed for answering the query.

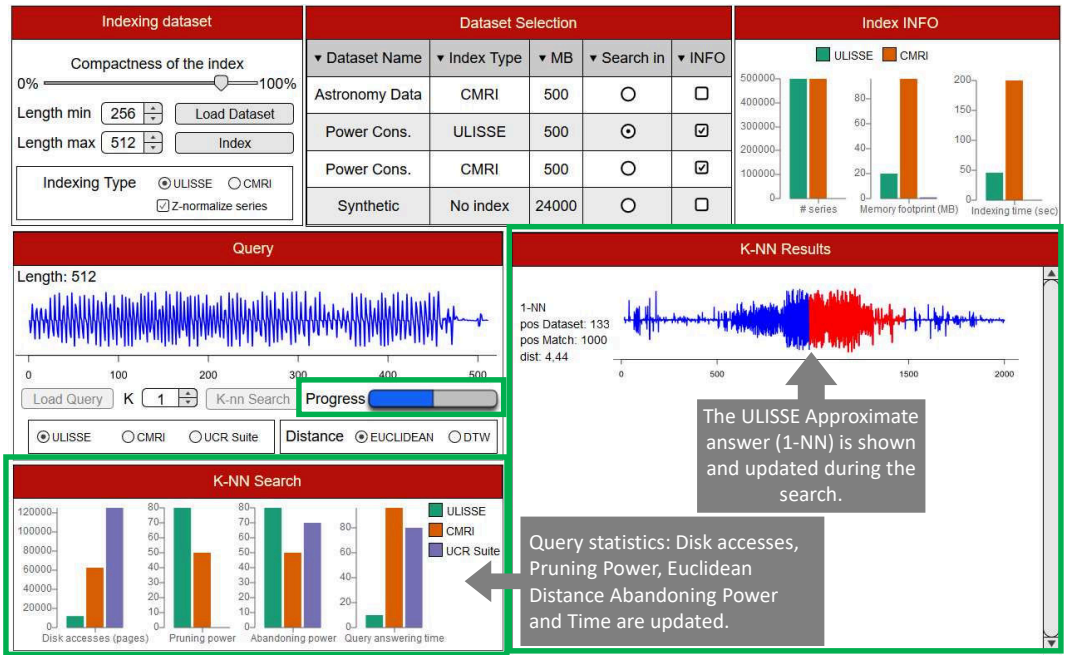


Figure 4.5: VALS query answering progression.

4.3 Conclusion

Similarity search is a fundamental operation for several data series analysis tasks. Even though much effort has been dedicated to indexing techniques that can speed up similarity search, all existing solutions are limited to queries of fixed length. In this chapter, we present VALS, a system based on the *ULISSE* index, which is the first system able to answer similarity search queries of variable-length, over both Z-normalized and non Z-normalized sequences.

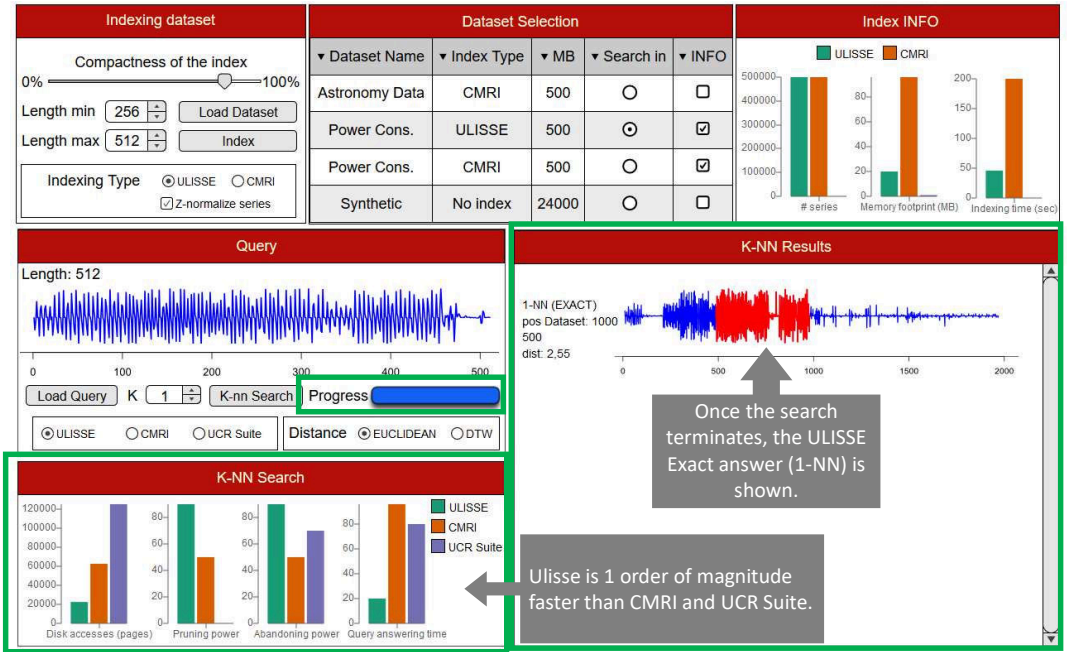


Figure 4.6: VALS K-NN exact results.

Chapter 5

Variable Length Motif and Discord Discovery

In the last fifteen years, data series *motif* and *discord* discovery have emerged as two useful and well-used primitives for data series mining, with applications to many domains, including robotics, entomology, seismology, medicine, and climatology. Nevertheless, the state-of-the-art motif and discord discovery tools still require the user to provide the relative length. Yet, in several cases, the choice of length is critical and unforgiving. Unfortunately, the obvious brute-force solution, which tests all lengths within a given range, is computationally untenable. In this chapter, we propose a new framework, which provides an exact and scalable motif and discord discovery algorithm that efficiently finds all motifs and discords in a given range of lengths.We evaluate our techniques using five diverse real datasets, and demonstrate the scalability of our approach. The results show that VALMOD is up to 20x faster than the state-of-the-art techniques. Furthermore, we present real case studies with datasets from entomology, seismology, and traffic data analysis, which demonstrate the usefulness of our approach. Our results also show that removing the unrealistic assumption that the user knows the correct length, can often produce more intuitive and actionable results, which could have otherwise been missed.

5.1 Chapter Organization

The remainder of this chapter is organized as follows. We introduce the notation needed for the rest of the chapter and formally define our problem in Section 5.2. In Section 5.3, we show our approach to rank motifs of different lengths, and in Sections 5.4 and 5.5, we describe the details of our motif discovery algorithms. In Section 5.6 we present and discuss our discord discovery solution. In the trailing part, with Section 5.7 we conclude this chapter showing the results of our extensive empirical evaluation.

5.2 Problem Definition

We begin by defining the data type of interest, data series:

Definition 3 (Data series) *A data series $T \in \mathbb{R}^n$ is a sequence of real-valued numbers $t_i \in \mathbb{R}$ $[t_1, t_2, ..., t_n]$, where n is the length of T.*

We are typically not interested in the global properties of a data series, but in the local regions known as subsequences:

Definition 4 (Subsequence) *A subsequence $T_{i,\ell} \in \mathbb{R}^\ell$ of a data series T is a continuous subset of the values from T of length ℓ starting from position i. Formally, $T_{i,\ell} = [t_i, t_{i+1}, ..., t_{i+\ell-1}]$.*

5.2.1 Motif Discovery

In this work, a particular local property we are interested in is data series motifs. A data series motif pair is the pair of the most similar subsequences of a given length, ℓ, of a data series:

Definition 5 (Data series motif pair) *$T_{a,\ell}$ and $T_{b,\ell}$ is a motif pair iff $dist(T_{a,\ell}, T_{b,\ell}) \leq dist(T_{i,\ell}, T_{j,\ell})$ $\forall i, j \in [1, 2, ..., n - \ell + 1]$, where $a \neq b$ and $i \neq j$,*

and dist is a function that computes the z-normalized Euclidean distance between the input subsequences [12, 69, 95, 100, 104].

Note, that if we remove the motif pair from the dataset, the pair with the second smallest distance will become the new motif pair. In this way, we can produce a ranked list of subsequence pairs, which we call motif pairs of length ℓ.

We store the distance between a subsequence of a data series with all the other subsequences from the same data series in an ordered array called a distance profile.

Definition 6 (Distance profile) *A distance profile* $D \in \mathbb{R}^{(n-\ell+1)}$ *of a data series* T *regarding subsequence* $T_{i,\ell}$ *is a vector that stores* $dist(T_{i,\ell}, T_{j,\ell})$, $\forall j \in [1, 2, ..., n-\ell+1]$, *where* $i \neq j$.

One of the most efficient ways to locate the exact data series motif is to compute the matrix profile [109, 21], which can be obtained by evaluating the minimum value of every distance profile in the time series.

Definition 7 (Matrix profile) *A matrix profile* $MP \in \mathbb{R}^{(n-\ell+1)}$ *of a data series* T *is a meta data series that stores the z-normalized Euclidean distance between each subsequence and its nearest neighbor, where* n *is the length of* T *and* ℓ *is the given subsequence length. The data series motif can be found by locating the two lowest values in* MP.

To avoid trivial matches [4], in which a pattern is matched to itself or a pattern that largely overlaps with itself, the matrix profile incorporates an "exclusion-zone" concept, which is a region before and after the location of a given query that should be ignored. The exclusion zone is heuristically set to $\ell/2$. The recently introduced STOMP algorithm [21] offers a solution to compute the matrix profile MP in $\mathcal{O}(n^2)$ time. This may seem untenable for data series mining, but several factors mitigate this concern. First, note that the time complexity is independent of ℓ, the length of the subsequences. Secondly, the matrix profile can be computed with an anytime algorithm, and in most domains, in just $\mathcal{O}(nc)$ steps the algorithm converges to what would be the final solution [109] (c is a small constant). Finally, the matrix profile can be computed with GPUs, cloud computing, and other HPC environments that make scaling to at least tens of millions of data points trivial [21].

We can now formally define the problems we solve.

Problem 2 (Variable-Length Motif Pair Discovery) *Given a data series T and a subsequence length-range $[\ell_{min}, ..., \ell_{max}]$, we want to find the data series motif pairs of all lengths in $[\ell_{min}, ..., \ell_{max}]$, occurring in T.*

One naive solution to this problem is to repeatedly run the state-of-the art motif discovery algorithms for every length in the range. However, note that the size of this range can be as large as $\mathcal{O}(n)$, which makes the naive solution infeasible for even middle-size data series. We aim at reducing this $\mathcal{O}(n)$ factor to a small value.

Note that the motif pair discovery problem has been extensively studied in the last decade [109, 21, 67, 51, 69, 65, 64]. The reason is that if we want to find a collection of recurrent subsequences in T, the most computationally expensive operation consists of identifying the motif pairs [21], namely, solving Problem 2. Extending motif pairs to sets incurs a negligible additional cost (as we also show in our study).

Given a motif pair $\{T_{\alpha,\ell}, T_{\beta,\ell}\}$, the data series motif set S_r^ℓ, with radius $r \in \mathbb{R}$, is the set of subsequences of length ℓ, which are in distance at most r from either $T_{\alpha,\ell}$, or $T_{\beta,\ell}$. More formally:

Definition 8 (Data series motif set) *Let $\{T_{\alpha,\ell}, T_{\beta,\ell}\}$ be a motif pair of length ℓ of data series T. The motif set S_r^ℓ is defined as: $S_r^\ell = \{T_{i,\ell} | dist(T_{i,\ell}, T_{\alpha,\ell}) < r \vee dist(T_{i,\ell}, T_{\beta,\ell}) < r\}$.*

The cardinality of S_r^ℓ, $|S_r^\ell|$, is called the frequency of the motif set.

Intuitively, we can build a motif set starting from a motif pair. Then, we iteratively add into the motif set all subsequences within radius r. We use the above definition to solve the following problem (optionally including a constraint on the minimum frequency for motif sets in the final answer).

Problem 3 (Variable-Length Motif Sets Discovery) *Given a data series T and a length range $[\ell_{min}, \ldots, \ell_{max}]$, we want to find the set $S^* = \{S_r^\ell | S_r^\ell$ is a motif set, $\ell_{min} \leq \ell \leq \ell_{max}\}$. In addition, we require that if $S_r^\ell, S_{r'}^{\prime\ell'} \in S^* \Rightarrow S_r^\ell \cap S_{r'}^{\prime\ell'} = \emptyset$.*

Thus, the variable-length motif sets discovery problem results in a set, S^*, of motif sets. The constraint at the end of the problem definition restricts each subsequence to be included in at most one motif set. Note that in practice we may not be interested in all the motif sets, but only in those with the k smallest distances, leading to a *top-k* version of the problem. In our work, we provide a solution for the *top-k* problem (though, setting k to a very large value will produce all results).

5.2.2 Discord Discovery

In order to introduce the problem of discord discovery, we first define the notion of *best match*, or *nearest neighbor*.

Definition 9 (m^{th} best match) *Given a subsequence $T_{i,\ell}$, we say that its m^{th} best match, or Nearest Neighbor (m^{th} NN) is $T_{j,\ell}$, if $T_{j,\ell}$ has the m^{th} shortest distance to $T_{i,\ell}$, among all the subsequences of length ℓ in T, excluding trivial matches.*

In the distance profile of $T_{i,\ell}$, the m^{th} smallest distance, is the distance of the m^{th} best match of $T_{i,\ell}$. We are now in the position to formally define the discord primitives, we use in our work.

Definition 10 (m^{th}*discord* [40]) *The subsequence $T_{i,\ell}$ is called the m^{th} discord of length ℓ, if its m^{th} best match is the largest among the best match distances of all subsequences of length ℓ in T.*

Intuitively, discovering the m^{th}*discord* enables us to find an isolated group of m subsequences, which are far from the rest of the data. Furthermore, we can rank the m^{th}*discords*, according to their m^{th} best matches. This allows us to define the *Top-k* m^{th} discords.

Definition 11 (*Top-k* m^{th} discords) *We call the k subsequences, with the k largest distances to their m^{th} best matches, the Top-k m^{th} discords.*

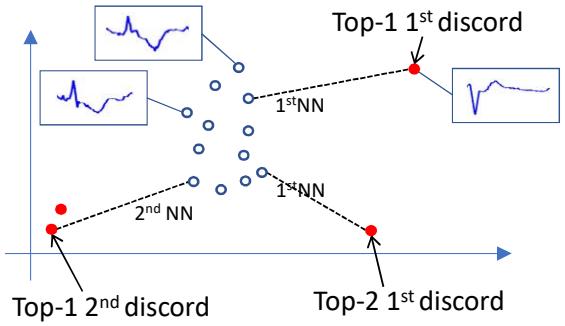


Figure 5.1: A dataset with 12 subsequences (of the same length ℓ) depicted as points in 2-dimensional space. We report the *Top-k* m^{th} discords.

In Figure 5.1, we plot a group of *12* subsequences (represented in a 2-dimensional space), and we depict three *Top-k* m^{th} discords (groups of red/dark circles). Remember that m represents the number of anomalous subsequences in a discord group. On the other hand, k ranks the discords and implicitly the groups, according to their m^{th} best match distances, in descending order (e.g., $Top-1$ 1^{st} discord and $Top-1$ 2^{nd}).

Given these definitions, we can formally introduce the following problem:

Problem 4 (Variable-Length *Top-k* m^{th} Discord Discovery) *Given a data series T, a subsequence length-range $[\ell_{min}, ..., \ell_{max}]$ and the parameters $a, b \in \mathbb{N}^+$ we want to enumerate the Top-k m^{th} discords for each $k \in \{1, .., a\}$ and each $m \in \{1, .., b\}$, and for all lengths in $[\ell_{min}, ..., \ell_{max}]$, occurring in T.*

Observe that solving the Variable-Length *Top-k* m^{th} Discord Discovery problem is relevant to solving the Variable-Length Motif Set Discovery problem: in the former case we are interested in the subsequences with the most distant neighbors, while in the latter case we seek the subsequences with the most close neighbors. Therefore, the Matrix Profile, which contains all this information, can serve as the basis to solve both problems.

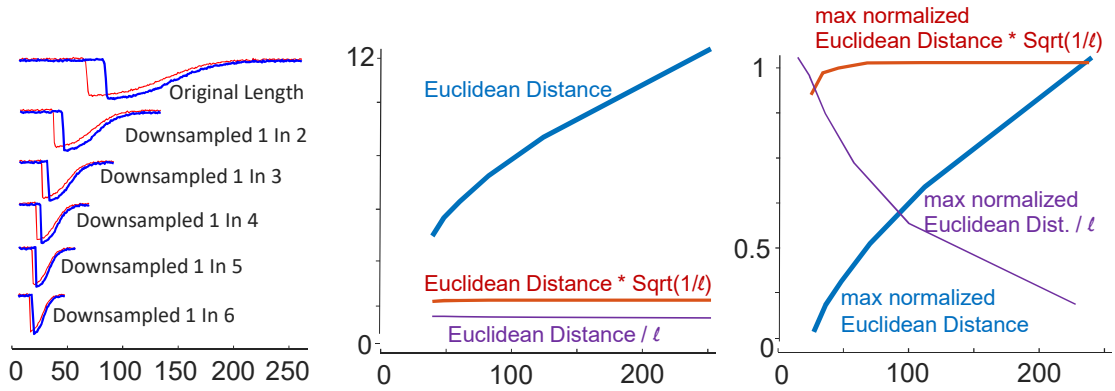


Figure 5.2: (*left*) Two series from the TRACE dataset at various speeds. (*center*) Euclidean distance. (*right*) Max normalized Euclidean distance.

5.3 Comparing Motifs of Different Lengths

Before introducing our solutions to the problems outlined above, we first discuss the issue of comparing motifs of different lengths. This becomes relevant when we want to rank motifs of different lengths (within the given range), which is useful in order to identify the most prominent motifs, irrespective of their length. In this section, we propose a length-normalized distance measure that the VALMOD algorithm uses in order to produce such rankings.

The increased expressiveness of VALMOD offers a challenge. Since we can *discover* motifs of different lengths, we also need to be able to *rank* motifs of different lengths. A similar problem occurs in string processing, and a common solution is to replace the edit-distance by the length-normalized edit-distance, which is the classic distance measure divided by the length of the strings in question [61]. This correction would find the pair {concatenation, concameration} *more* similar than {cat, cot}, matching our intuition, since only 15% of the characters are different in the former pair, as opposed to 33% in the latter.

Researchers have suggested this length-normalized correction for time series, but as we will show, the correction factor is incorrect. To illustrate this, consider the following thought experiment. Imagine that some process in the system we are monitoring occasionally "injects" a pattern into the time series. As a concrete example, washing machines typically have a prototypic signature (as exhibited in the TRACE dataset [82]), but the signatures express themselves more slowly on a cold day, when it takes longer to heat the cooler water supplied from the city [27]. We would like all equal length instances of the signature to have approximately

the same distance. As a consequence, we factorize the Euclidean distance by the following quantity: $sqrt(1/\ell)$, where ℓ is the length of the sequences. This aims to favor longer and similar sequences in the ranking process of matches that have different lengths.

In Figure 5.2(*left*) we show two examples from the TRACE dataset [82], which will act as proxies for a variable length signature. We produced the variable lengths by down sampling. In Figure 5.2(*center*), we show the distances between the patterns as their length changes. With no correction, the Euclidean distance is obviously biased to the shortest length. The length-normalized Euclidean distance looks "flatter" and suggests itself as the proper correction. However, its variation over the sequence length change is not visible due to the small scale. In Figure 5.2(*right*), we show all of the measures after dividing them by their largest value. Now we can see that the length-normalized Euclidean distance has a strong bias toward the longest pattern. In contrast to the other two approaches, the $sqrt(1/length)$ correction factor provides a near perfect invariant distance over the entire range of values.

5.4 Proposed Approach for Motif Discovery

Our algorithm, VALMOD (Variable Length Motif Discovery), starts by computing the *matrix profile* on the smallest subsequence length, namely ℓ_{min}, within a specified range $[\ell_{min}, \ell_{max}]$. The key idea of our approach is to minimize the work that needs to be done for subsequent subsequence lengths ($\ell_{min}+1$, $\ell_{min}+2$, ..., ℓ_{max}). In Figure 5.3, it can be observed that the motif of length 8 ($T_{33,8} - T_{97,8}$) has the same offsets as the motif of length 9 ($T_{33,9} - T_{97,9}$). Can we exploit this property to accelerate our computation?

It seems that if the nearest neighbor of $T_{i,\ell_{min}}$ is $T_{j,\ell_{min}}$, then probably the nearest neighbor of $T_{i,\ell_{min}+1}$ is $T_{j,\ell_{min}+1}$. For example, as shown in Figure 5.3(bottom), if we sort the distance profiles of $T_{33,8}$ and $T_{33,9}$ in ascending order, we can find that the nearest neighbor of $T_{33,8}$ is $T_{97,8}$, and the nearest neighbor of $T_{33,9}$ is $T_{97,9}$.

One can imagine that if the location of the nearest neighbor of $T_{i,\ell}$ ($i = 1, 2, ..., n-m+1$) remains the same as we increase ℓ, then we could obtain the matrix profile of length $\ell + k$ in $\mathcal{O}(n)$ time ($k = 1, 2, \ldots$). However, this is not always true. The location of the nearest neighbor of $T_{i,\ell}$ may not change as we slightly increase ℓ, if there is a substantial margin between the first and second entries of $D_{ranked}(T_{i,\ell})$.

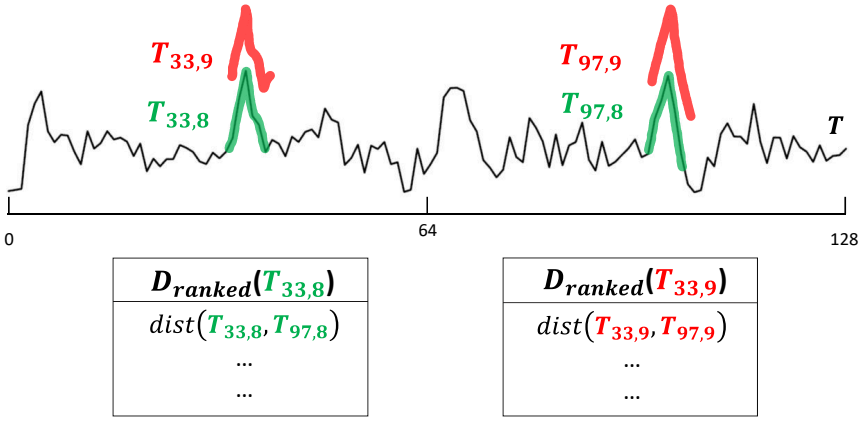


Figure 5.3: (*top*) The top motifs of length 9 and 8 in an example data series. (*bottom*) The sorted distance profiles of $T_{33,8}$ and $T_{33,9}$.

But, as ℓ gets larger, the nearest neighbor of $T_{i,\ell}$ is likely to change. For example, as shown in Figure 5.4, when the subsequence length grows to 19, the nearest neighbor of $T_{33,19}$ is no longer $T_{97,19}$, but $T_{1,19}$. We observe that the ranking of the distance profile values may change, even when the data is relatively smooth. When the data is noisy and skewed, this ranking can change even more often. Is there any other rank-preserving measure that we can exploit to accelerate the computation?

The answer is yes. Instead of sorting the entries of the distance profile, we create and sort a new vector, called the *lower bound distance profile*. Figure 5.4(bottom) previews the rank-preserving property of the lower bound distance profile. As we will describe later, once we know the distance between $T_{i,\ell}$ and $T_{j,\ell}$, we can evaluate a lower bound distance between $T_{i,\ell+k}$ and $T_{j,\ell+k}$, $\forall k \in [1,2,3,\ldots]$. The rank-preserving property of the lower bound distance profile can help us prune a large number of unnecessary computations as we increase the subsequence length.

5.4.1 The Lower Bound Distance Profile

Before introducing the lower bound distance profile, let us first investigate its basic element: the lower bound Euclidean distance.

Assume that we already know the z-normalized Euclidean distance $d_{i,j}^{\ell}$ between two subsequences of length ℓ: $T_{i,\ell}$ and $T_{j,\ell}$, and we are now estimating the distance

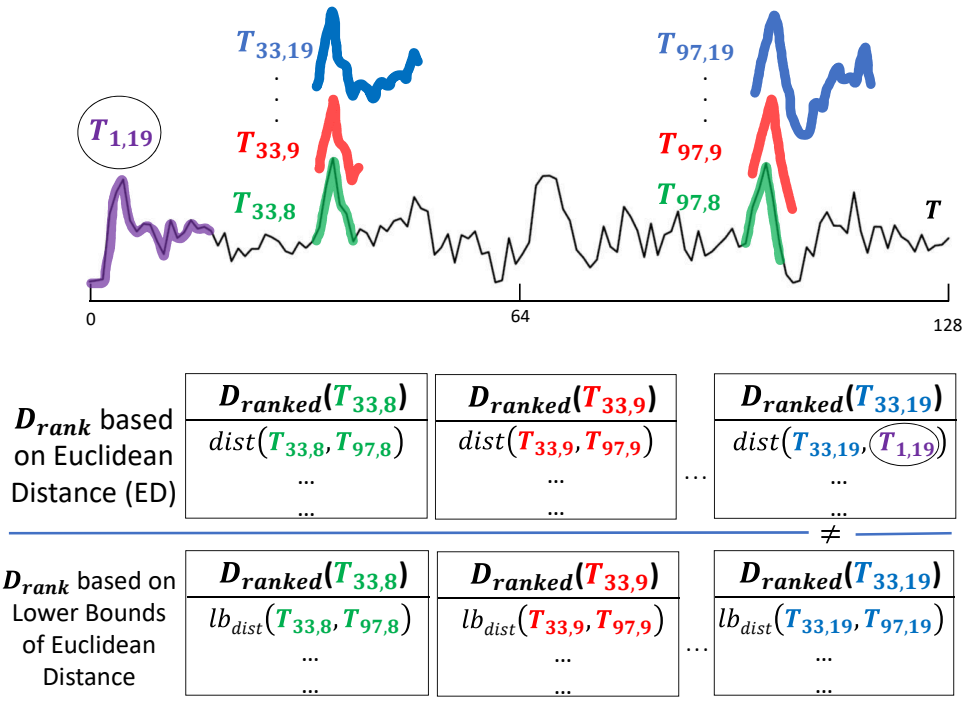


Figure 5.4: (*top distance profiles*) Ranking by true distances leads to changes in the order of the pairs. (*bottom distance profiles*) Ranking by lower bound distances maintains the same order of pairs over increasing lengths.

between two longer subsequences of length $\ell + k$: $T_{i,\ell+k}$ and $T_{j,\ell+k}$. Our problem can be stated as follows: given $T_{i,\ell}$, $T_{j,\ell}$ and $T_{j,\ell+k}$ (but not the last k values of $T_{i,\ell+k}$), is it possible to provide a lower bound function $LB(d_{i,j}^{\ell+k})$, such that $LB(d_{i,j}^{\ell+k}) \leq d_{i,j}^{\ell+k}$? This problem is visualized in Figure 5.5 .

One may assume that we can simply set $LB(d_{i,j}^{\ell+k}) = d_{i,j}^{\ell}$ by assuming that the last k values of $T_{i,\ell+k}$ are the same as the last k values of $T_{j,\ell+k}$. However, this is not an answer to our problem, as we need to evaluate z-normalized Euclidean distances, which are not simple Euclidean distances. The mean and standard deviation of a subsequence can change as we increase its length, so we need to re-normalize both $T_{i,\ell+k}$ and $T_{j,\ell+k}$. Assume that the mean and standard deviation of $T_{x,y}$ are $\mu_{x,y}$ and $\sigma_{x,y}$, respectively (i.e. $T_{j,\ell+k}$ corresponds to $\mu_{j,\ell+k}$ and $\sigma_{j,\ell+k}$). Since we do not know the last k values of $T_{i,\ell+k}$, both $\mu_{i,\ell+k}$ and $\sigma_{i,\ell+k}$ are unknown and can thus be regarded as variables. We recall that t_i denotes the i^{th} point of a generic

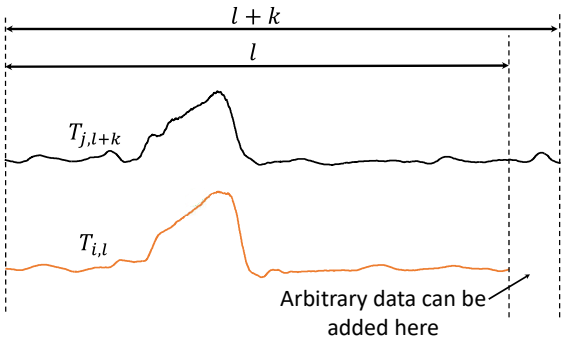


Figure 5.5: Increasing the subsequence length from ℓ to $\ell + k$.

sequence T (or a subsequence $T_{a,b}$), we thus have the following:

$$d_{i,j}^{\ell+k} \geq \min_{\mu_{i,\ell+k},\sigma_{i,\ell+k}} \sqrt{\sum_{p=1}^{\ell} \left(\frac{t_{i+p-1} - \mu_{i,\ell+k}}{\sigma_{i,\ell+k}} - \frac{t_{j+p-1} - \mu_{j,\ell+k}}{\sigma_{j,\ell+k}}\right)^2}$$

$$= \min_{\mu_{i,\ell+k},\sigma_{i,\ell+k}} \frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} \sqrt{\sum_{p=1}^{\ell} \left(\frac{t_{i+p-1} - \mu_{i,\ell+k}}{\frac{\sigma_{i,\ell+k}\sigma_{j,\ell}}{\sigma_{j,\ell+k}}} - \frac{t_{j+p-1} - \mu_{j,\ell+k}}{\sigma_{j,\ell}}\right)^2}$$

Here, we substitute the variables $\mu_{i,\ell+k}$ and $\sigma_{i,\ell+k}$, respectively with μ' and σ'. Hence, we obtain:

$$= \min_{\mu',\sigma'} \frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} \sqrt{\sum_{p=1}^{\ell} \left(\frac{t_{i+p-1} - \mu'}{\sigma'} - \frac{t_{j+p-1} - \mu_{j,\ell}}{\sigma_{j,\ell}}\right)^2} \tag{5.1}$$

Clearly, the minimum value shown in Eq. 5.1 can be set as $LB(d_{i,j}^{\ell+k})$. We can obtain $LB(d_{i,j}^{\ell+k})$ by solving $\frac{\partial LB(d_{i,j}^{\ell+k})}{\partial \mu'} = 0$ and $\frac{\partial LB(d_{i,j}^{\ell+k})}{\partial \sigma'} = 0$:

$$LB(d_{i,j}^{\ell+k}) = \begin{cases} \sqrt{\ell}\frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} & \text{if } q_{i,j} \leq 0 \\ \sqrt{\ell(1-q_{i,j}^2)}\frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} & \text{otherwise} \end{cases} \tag{5.2}$$

where $q_{i,j} = \frac{\sum_{p=1}^{\ell} \frac{(t_{j+p-1} t_{i+p-1})}{\ell} - \mu_{i,\ell} \mu_{j,\ell}}{\sigma_{i,\ell} \sigma_{j,\ell}}$.

$LB(d_{i,j}^{\ell+k})$ yields the minimum possible z-normalized Euclidean distance between $T_{i,\ell+k}$ and $T_{j,\ell+k}$, given $T_{i,\ell}$, $T_{j,\ell}$ and $T_{j,\ell+k}$ (but not the last k values of $T_{i,\ell+k}$). Now that we have obtained the lower bound Euclidean distance between two subsequences, we are able to introduce the lower bound distance profile.

Using Eq. 5.2, we can evaluate the lower bound Euclidean distance between $T_{j,\ell+k}$ and every subsequence of length $\ell + k$ in T. By putting the results in a vector, we obtain the lower bound distance profile $LB(D_j^{\ell+k})$ corresponding to subsequence $T_{j,\ell+k}$: $LB(D_j^{\ell+k}) = LB(d_{1,j}^{\ell+k})$, $LB(d_{2,j}^{\ell+k})$, ...,$LB(d_{n-\ell-k+1,j}^{\ell+k})$. If we sort the components of $LB(D_j^{\ell+k})$ in an ascending order, we can obtain the ranked lower bound distance profile: $LB_{ranked}(D_j^{\ell+k}) = LB(d_{r_1,j}^{\ell+k}), LB(d_{r_2,j}^{\ell+k})$, ..., $LB(d_{r_{n-\ell-k+1},j}^{\ell+k})$, where $LB(d_{r_1,j}^{\ell+k}) \leq LB(d_{r_2,j}^{\ell+k}) \leq ...$ $\leq LB(d_{r_{n-\ell-k+1},j}^{\ell+k})$.

We would like to use this ranked lower bound distance profile to accelerate our computation. Assume that we have a best-so-far pair of motifs with a distance $dist_{BSF}$. If we examine the p^{th} element in the ranked lower bound distance profile and find that $LB(d_{r_p,j}^{\ell+k}) > dist_{BSF}$, then we do not need to calculate the exact distance for $d_{r_p,j}^{\ell+k}, d_{r_{p+1},j}^{\ell+k}, ..., d_{r_{n-\ell-k+1},j}^{\ell+k}$ anymore, as they cannot be smaller than $dist_{BSF}$. Based on this observation, our strategy is as follows. We set a small, fixed value for p. Then, for every j, we evaluate whether $LB(d_{r_p,j}^{\ell+k}) > dist_{BSF}$ is true: if it is, we only calculate $d_{r_1,j}^{\ell+k}, d_{r_2,j}^{\ell+k}, ..., d_{r_{p-1},j}^{\ell+k}$. If it is not, we compute all the elements of $D_j^{\ell+k}$. We update $dist_{BSF}$ whenever a smaller distance value is observed. In the best case, we just need to calculate $\mathcal{O}(np)$ exact distance values to obtain the motif of length $l + k$. Note that the order of the ranked lower bound distance profile is preserved for every k. That is to say, if $LB(d_{a,j}^{\ell+k}) \leq LB(d_{b,j}^{\ell+k})$, then $LB(d_{a,j}^{\ell+k+1}) \leq LB(d_{b,j}^{\ell+k+1})$. This is because the only component in Eq. 5.2 related to k is $\sigma_{j,\ell+k}$. When we increase k by 1, we are just performing a linear transformation for the lower bound distance: $LB(d_{i,j}^{\ell+k+1}) = LB(d_{i,j}^{\ell+k}) \sigma_{j,\ell+k} / \sigma_{j,\ell+k+1}$. Therefore, we have $LB(d_{r_p,j}^{\ell+k+1}) = LB(d_{r_p,j}^{\ell+k}) \sigma_{j,\ell+k} / \sigma_{j,\ell+k+1}$, and the ranking is preserved for every k.

Algorithm 6: $VALMOD$

Input: **DataSeries** T, **int** ℓ_{min} **int** ℓ_{max}, **int** p
Output: $VALMP$

```
1  int nDP ← |T| − ℓ_min + 1 ;
2  VALMP ← new VALMP(nDP);
3  VALMP.MP = {⊥,...,⊥};
4  MaxHeap[] listDP , double [] MP, int [] IP ;
5  listDP, MP , IP ← ComputeMatrixProfile(T, ℓ_min, p); // listDP contains p
       entries of each distance profile
6  VALMP ← updateVALMP(VALMP,MP,IP,nDP) ;
7  for i ← ℓ_min + 1 to ℓ_max do
8      nDP ← |T| − i + 1 ;
       // compute SubMP and update listDP for the length i
9      bool bBestM, double [] SubMP, IP ← ComputeSubMP(T,nDP,listDP,i,p);
10     if bBestM then
           // SubMP surely contains the motif, update VALMP with it
11         updateVALMP(VALMP,SubMP,IP,nDP);
12     else
13         listDP,MP,IP ← ComputeMatrixProfile(T,i,p);
           // SubMP might not contain the motif, update VALMP computing MP
14         updateVALMP(VALMP,MP,IP,nDP);
```

Algorithm 7: $updateVALMP$

Input: $VALMP$, **double** [] $MPnew$, **int** [] IP, nDP, ℓ
Output: $VALMP$

```
1  for i ← 1 to nDP do
       // length normalize the Euclidean distance
2      double lNormDist ← MPnew[i] * √(1/ℓ);
       // if the distance at offset i of VALMP, surely computed with
          previous lengths, is larger than the actual, update it
3      if (VALMP.distances[i] > lNormDist or VALMP.MP[i] == ⊥) then
4          VALMP.distances[i] ← MPnew[i];
5          VALMP.normDistances[i] ← lNormDist;
6          VALMP.lengths[i] ← ℓ;
7          VALMP.indices[i] ← IP[i];
```

5.4.2 The VALMOD Algorithm

We are now able to formally describe the VALMOD algorithm. The pseudocode for VALMOD is shown in Algorithm 6. With the call of *ComputeMatrixProfile()*

in line 5, we build the matrix profile corresponding to ℓ_{min}, and in the meantime store the smallest p values of each distance profile in the memory. Note that the matrix profile is stored in the vector MP, which is coupled with the matrix profile index, IP, which is a structure containing the offsets of the nearest neighbor subsequences. We can easily find the motif corresponding to ℓ_{min} as the minimum value of MP. Then, in lines 7-14, we iteratively look for the motif of every length within ℓ_{min}+1 and ℓ_{max}. The $ComputeSubMP$ function in line 9 attempts to find the motif of length i only by evaluating a subset of the matrix profile corresponding to subsequence length i. Note that this strategy, which is based on the lower bounding technique introduced in Section 5.4.1, might not be able to capture the global minimum value within the matrix profile. In case that happens (which is rare), the Boolean flag $bBestM$ is set to false, and we compute the whole matrix profile with the $computeMatrixProfile$ procedure in line 13.

The final output of $VALMOD$ is a vector, which is called $VALMP$ (*variable length matrix profile*) in the pseudo-code. If we were interested in only one fixed subsequence length, *VALMP* would be the matrix profile normalized by the square root of the subsequence length. If we are processing various subsequence lengths, then as we increase the subsequence length, we update *VALMP* when a smaller length-normalized Euclidean distance is observed.

Algorithm 7 shows the routine to update the $VALMP$ structure. The final $VALMP$ consists of four parts. The i^{th} entry of the *normDistances* vector stores the smallest length-normalized Euclidean distance values between the i^{th} subsequence and its nearest neighbor, while the i^{th} place of vector *distances* stores their straight Euclidean distance. The location of each subsequence's nearest neighbor is stored in the vector *indices*. The structure *lengths* contains the length of the i^{th} subsequences pair.

In the next two subsections, we detail the two sub-routines, $computeMatrixProfile$ and the $ComputeSubMP$.

5.4.3 Computing The Matrix Profile

The routine $ComputeMatrixProfile$ (Algorithm 8) computes a matrix profile for a given subsequence length, ℓ. It essentially follows the STOMP algorithm [21], except that we also calculate the lower bound distance profiles in line 17. In line 5, the dot product between the sequence $T_{1,\ell}$ and the others in T is computed

Algorithm 8: *ComputeMatrixProfile*

Input: **DataSeries** T, **int** ℓ, **int** p
Output: MP, $listDP$

```
1  int nDP ← |T|-ℓ+1;
2  double [] MP ← double [nDP];
3  int [] IP ← int [nDP];
4  MaxHeap[] listDP= new MaxHeap(p)[nDP];
   // compute the dot product vector QT for the first distance profile
5  double [] QT ← SlidingDotProduct(T_{1,ℓ}, T);
   // compute sum and squared sum of the first subsequence of length ℓ
6  s ← sum(T_{1,ℓ}); ss ← squaredSum(T_{1,ℓ});
   // compute the first distance profile with distance formula (Eq.(5.3)) and
      store the minimum distance in MP and the offset of the nearest neighbor in
      IP
7  D(T_{i,ℓ}) ← CalcDistProfile(QT,T_{i,ℓ}, T, s, ss);
8  MP[1], IP[1] ← min(D(T_{i,ℓ}));
   // iterate over the subsequences of T
9  for i ← 2 to nDP do
       // update the dot product vector QT for the i^th subsequence
10     for j ← nDP down to 2 do
11         QT[j]←QT[j − 1] − T[j − 1] × T[i − 1] + T[j + ℓ − 1] × T[i + ℓ − 1] ;
       // update sum and squared sum of the i^th subsequence
12     s ← s − T[i − 1] + T[ℓ + i − 2];
13     ss ← ss − T[i − 1]^2 + T[ℓ + i − 2]^2;
14     D(T_{i,ℓ}) ← CalcDistProfile(QT,T_{i,ℓ}, T, s, ss);
15     MP[i], IP[i] ← min(D(T_{i,ℓ}));
       // Store in listDP[i] the p entries e with smallest lower bounding distance
16     int c ← 0;
17     for each entry e in D(T_{i,ℓ}) do
           // Compute the lower bound for the length ℓ + 1
18         e.LB ← compLB(ℓ, ℓ + 1, QT[c], e.s1, e.s2, e.ss1, e.ss2);
           // save the entry only if is smaller than the max lb so far or if
              listDP[i] contains fewer than p elements
19         if e.LB < max(listDP[i]) or |listDP[i]| < p then
20             insert(listDP[i], e);
21         c← c + 1;
```

in *frequency domain* in $\mathcal{O}(nlogn)$ time, where $n = |T|$. The dot product is computed in constant time in line 11 by using the result of the previous overlapping subsequences.

In line 7 we measure each z-normalized Euclidean distance, between $T_{i,\ell}$ and the other subsequence of length ℓ in T, avoiding trivial matches. The distance measure formula used is the following [68, 109, 21]:

$$dist(T_{i,\ell}, T_{j,\ell}) = \sqrt{2\ell(1 - \frac{QT_{i,j} - \ell\mu_i\mu_j}{\ell\sigma_i\sigma_j})} \tag{5.3}$$

In Eq. (5.3) $QT_{i,j}$ represents the dot product of the two sub-series with offset i and j respectively. It is important to note that, we may compute μ and σ in constant time by using the *running* plain and squared sum, namely s and ss (initialized in line 6). It follows that $\mu = s/\ell$ and $\sigma = \sqrt{(ss/\ell) - \mu^2}$.

In lines 8 and 15, we update both the matrix profile and the matrix profile index, which holds the offset of the closest match for each $T_{i,l}$.

Algorithm 8 ends with the loop in line 17, which evaluates the lower bound distance profile and stores the p smallest lower bound distance values in $listDP$. In line 18, the procedure *compLB* evaluates the lower bound distance profile introduced in Section 5.4.1 using Eq. (5.2). The structure $listDP$ is a Max Heap with a maximum capacity of p. Each entry e of the distance profile in line 17 is a tuple containing the Euclidean distance between a subsequence $T_{j,\ell}$ and its nearest neighbor, the location of that nearest neighbor, the lower bound Euclidean distance of the pair, the dot product of them, and the plain and squared sum of $T_{j,\ell}$. In Figure 5.6(b), we show an example of the distance profile in line 17. The distance profile is sorted according to the lower bound Euclidean distance values (shown as LB in the figure). The entries corresponding to the p smallest LB values are stored in memory to be reused for longer motif lengths.

Complexity Analysis. In line 14 of Algorithm 7, the time cost to compute a single distance profile is $\mathcal{O}(n)$, where n is the number of subsequences of length ℓ. Therefore computing the n distance profiles takes $\mathcal{O}(n^2)$ time. In line 17, computing the lower bounds of the smallest p entries of each distance profile takes $\mathcal{O}(n \log(p))$ additional time. The overall time complexity of the $ComputeMatrixProfile$ routine is thus $\mathcal{O}(n^2 \log(p))$. This routine is called at least once, for the first subsequence length of the range, namely $\ell = \ell_{min}$. In the worst case, it is executed for each length in the range (though, this never occurred in our experiments).

Algorithm 9: *ComputeSubMP*

Input: **DataSeries** T, **int** nDp, **MaxHeap**[] $listDP$, **int** $newL$, **int** p
Output: bBestM, $SubMP$, IP

double[] $SubMP \leftarrow$ **double**[nDp];
int[] $IP \leftarrow$ **int**[nDp];
double $minDistAbs \leftarrow$ inf, **double** $minLbAbs \leftarrow$ inf;
List $\langle$ **int,double** $\rangle$ $nonValidDP$;
`// iterate over the partial distance profiles in listDP`
for $i \leftarrow 1$ **to** nDp **do**
  double $minDist \leftarrow$ inf;
  int $ind \leftarrow 0$;
  double $maxLB \leftarrow$ **popMax**($listDP[i]$);
  `// update the partial distance profile for the length newL (true Euclidean and lower bounding distance )`
  for ***each entry*** e *in* $listDP[i]$ **do**
    $e.dist$, $e.LB \leftarrow updateDistAndLB(e, newL)$;
    $minDist \leftarrow$ **min**($minDist$,$e.dist$);
    if $minDist == e.dist$ **then**
      $ind = e.offset$;
  `// check if the min (minDist) of this partial distance profile is the min of the complete distance profile`
  if $minDist < maxLB$ **then**
    `// minDist is the real min; valid distance profile`
    $minDistABS \leftarrow$ **min**($minDistAbs$,$minDist$);
    $SubMP[i] = minDist$;
    $IP[i] = ind$;
  else
    `// minDist is not the real min; non-valid distance profile`
    $minLbAbs \leftarrow$ **min**($minLbAbs$, $maxLB$));
    $SubMP[i] = \perp$;
    $nonValidDP.add(\langle i, maxLB \rangle)$
bool $bBestM \leftarrow (minDistABS < minLbAbs)$;
`// if SubMP does not contain the motif distance (bBestM = false), compute the whole non-valid distance profiles, if it is faster then computeMatrixProfile (nDp / 2 = true)`
if !$bBestM$ ***and*** $nonValidDP.size() < (nDp/p)$ **then**
  for ***each pair*** $< ind, lbMax >$ ***in*** $nonValidDP$ **do**
    if $lbMax < minDistABS$ **then**
      $QT \leftarrow SlidingDotProduct(T_{ind,\ell}, T)$;
      double $s \leftarrow sum(T_{ind,\ell})$; **double** $ss \leftarrow squaredSum(T_{ind,\ell})$;
      $D(T_{ind,\ell}) \leftarrow CalcDistProf(QT, T_{ind,\ell}, T, s, ss)$;
      $SubMP[ind]$, $IP[ind] =$ **min**($D(T_{ind,\ell})$);
      $insert(listDP[ind], D(T_{ind,\ell}))$;
  $bBestM \leftarrow 1$;

5.4.4 Matrix Profile for Subsequent Lengths

We are now ready to describe our *ComputeSubMP* algorithm, which allows us to find the motifs for subsequence lengths greater than ℓ in linear time.

The input of *ComputeSubMP*, whose pseudo-code is shown in Algorithm 9, is the vector $listDp$ that we built in the previous step. In line 5, we start to iterate over the $p \times n$ elements of $listDp$ in order to find the motif pair of length $newL$, using a procedure that is faster than Algorithm 6, leading to a complexity that is now linear in the best case (as the experiments show, this is often the case). Since $listDP$ potentially contains enough elements to compute the whole matrix profile, it can provide more information than just the motif pair.

In the loop of line 9, we update all the entries of $listDP[i]$ by computing the Euclidean and lower bound distance for the length $newL$. This operation is valid, since the ranking of each $listDP[i]$ is maintained as the lower bound gets updated. Moreover, this latter computation is done in constant time (line 10), since the entries contain the statistics (i.e. sum, squared sum, dot product) for the length $newL-1$. Also note that the routine $updateDistAndLB$ avoids the trivial matches, which may result from the length increment.

Subsequently, the algorithm checks in line 14 if $minDist$ is smaller than or equal to $maxLB$, the largest lower bound distance value in $listDP[i]$. If this is true, $minDist$ is the smallest value in the whole distance profile. In lines 15 and 16, we update the best-so-far distance value and the matrix profile. On the other hand, we update the smallest max lower bounding distance in line 19, recording also that we do not have the true min for the distance profile with offset i (line 21). Here, we may also note that even though the local true min is larger than the max lower bound (i.e., the condition of line 14 is not true), $minDist$ may still represent an approximation of the true matrix profile point.

When the iteration of the partial distance profiles ends (end of for loop in line 5), the algorithm has enough elements to know if the matrix profile computed contains the real motif pair. In line 22, we verify if the smallest Euclidean distance we computed ($minDistABS$) is less than $minLbAbs$, which is the minimum lower bound of the *non-valid* distance profiles. We call non-valid all the partial distance profiles, for which the maximum lower bound distance (i.e., the p-th largest lower bound of the distance profile) is smaller than the minimum true distance (line 18); otherwise, we call them valid (line 14).

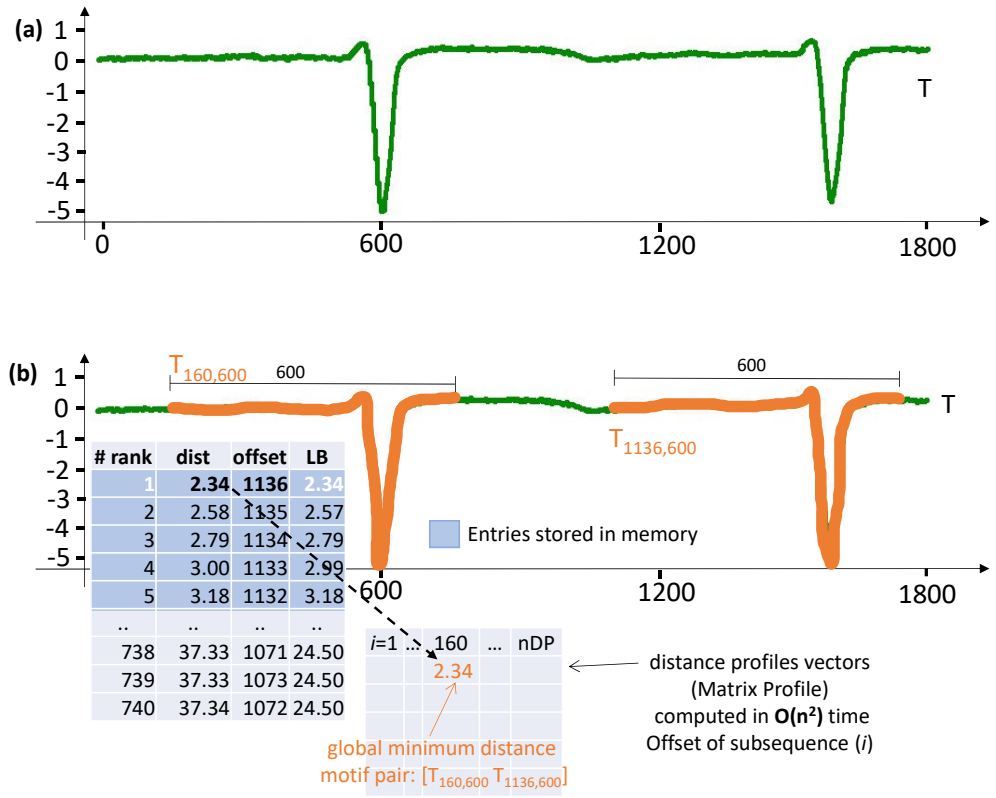


Figure 5.6: (*a*) Input time series, (*b*) Compute matrix profile snapshot: (on the left) distance profile of the subsequence $T_{160,600}$ which is part of the motif.

As a result of the ranking preservation of the lower bounding function, if the above criterion holds, we know that each true Euclidean distance in the non-valid distance profiles must be greater than $minDistABS$. In line 23, the algorithm has its last opportunity to exploit the lower bound in the distance profiles, in order to avoid computing the whole matrix profile. If $bBestM$ is false (the motif has not been found), we start to iterate through the non-valid distances profiles. Note that we perform this iteration when their number is less than half of the total distance profiles.

We present here two examples that explain the main procedures of $VALMOD$.

Example 1 *In Figure 5.6, we show a snapshot of a VALMOD run. In Figure 5.6(a), VALMOD receives as input a data series of length 1800. In Fig-*

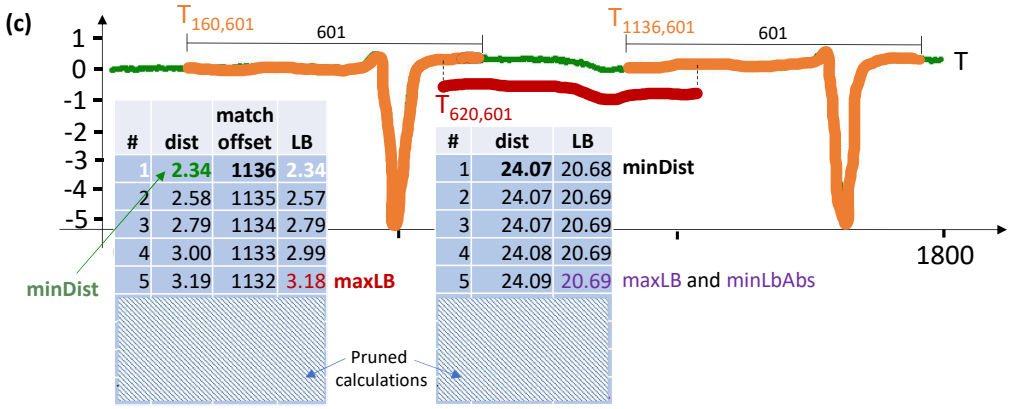


Figure 5.7: Compute Sub Matrix profile: the partial distance profile of $T_{160,601}$ contains the motif's subsequences distance.

ure 5.6(b), the matrix profile for subsequence length $\ell = 600$ is computed (Algorithm 8). On the left, we depict the distance profile regarding $T_{160,600}$, and rank it according to the lower bound (LB) distance values. Although we are computing the entire distance profile, we store only the first $p = 5$ entries in memory.

Example 2 *Figure 5.7 shows the execution of ComputeSubMP (Algorithm 9), taking place after the step illustrated in Figure 5.6(b). In this picture, we show the distance profile of a subsequence belonging to the motif pair, for subsequence length $\ell = 601$. This time it is built by computing $p = 5$ distances (left side of the picture). We can now make the following observations:*
(a) In the distance profile of the subsequence $T_{160,601}$ (left array): $minDist = 2.34 < maxLB = 3.18 \iff$ the value 2.34 is both a local and a global minimum (among all the distance profiles).
(b) Considering the partial distance profile of subsequence $T_{620,601}$ (right array), we do not know if its minDist is its real global minimum, since 20.69 ($maxLB$) < 24.07 ($minDist$).
(c) We know, that 20.69 ($maxLB$ of the distance profile of subsequence $T_{620,601}$) is the minLbAbs, or in other words, the smallest $maxLB$ distance among all the partial distance profiles in which $maxLB < minDist$ holds.
(d) We know that there are no true Euclidean distances (among those computed) smaller than 2.34. Since $minDist = 2.34 < minLbAbs = 20.69 \iff$ 2.34 is the distance of the motif $\{T_{160,601}; T_{1136,601}\}$.

Complexity Analysis. In the best case, *ComputeSubMP* can find the motif

pair in $\mathcal{O}(np)$ time, where n is the total number of distance profiles. This means that no distance profile computation takes place, since the condition in line 22 of Algorithm 10 is satisfied. Otherwise, if we need to iterate over the non-valid distance profiles for finding the answer (which occurs rarely in practice), the time complexity reaches its worst case, $\mathcal{O}(nC\log(n))$, with $C = n/p$. This is asymptotically faster than re-executing *ComputeMatrixProfile*, which takes $\mathcal{O}(n^2\log(p))$ time. Note that, each non-valid distance profile (starting in line 26) is computed by using the primitives introduced in the *ComputeMatrixProfile* algorithm, only if its maximum lower bound is less than the smallest true distance $minDistABS$. This indicates that the distance profile for length $newL$ may contain not yet computed distances smaller than $minDistABS$, which is our *best-so-far*. Therefore, the overall complexity of VALMOD is $\mathcal{O}(n^2\log(p) + (\ell_{max} - \ell_{min})np)$ in the best case, whereas the worst case time complexity is $\mathcal{O}((\ell_{max} - \ell_{min})n^2\log(p))$. Clearly, the $n^2\log(p)$ factor dominates, since $(\ell_{max} - \ell_{min})$ acts as a constant. Nevertheless, the length range is not negligible, w.r.t the time performance, when we need to run a quadratic routine over it. If the worst case occurs often, then the performance will degrade. However, this is not the case, as we show in the experimental evaluation.

5.5 Finding Motif Sets

We finally extend our technique in order to find the variable-length motif sets. In that regard, we start to consider the *top-k* motif pairs, namely the pairs having the k smallest length-normalized distances. The idea is to extend each motif pair to a motif set considering the subsequence's proximity as a quality measure, thus favoring the motif sets, which contain the closest subsequence pairs. Moreover, for each top-K motif pair ($T_{a,\ell}$,$T_{b,\ell}$), we use a radius $r = D * dist(T_{a,\ell}, T_{b,\ell})$, when we extend it to a motif set. We call the real variable D *radius factor*. This choice permits us to tune the radius r by the user defined radius factor, considering also the characteristics of the data. Setting a unique and non data dependent radius for all motif sets, would penalize the results of exploratory analysis.

First, we introduce Algorithm 10, a slightly modified version of the $updateValmp$ routine (Algorithm 7). The new algorithm is called $updateVALMPForMotifSets$, and its main goal is to keep track of the best k subsequence pairs (motif pairs) according to the $VALMP$ ranking, and the corresponding partial distance profiles. The idea is to later exploit the lower bounding distances for pruning computations, while computing the motif sets.

Algorithm 10: *updateVALMPForMotifSets*

Input: $VALMP$, **double** [] $MPnew$, **int** [] IP, nDP, ℓ, **MaxHeap**[] $listDP$,
Output: $VALMP$

```
1  for i ← 1 to nDP do
       // length normalize the Euclidean distance
2      double lNormDist ← MPnew[i] * sqrt(1/ℓ);
       // if the distance at offset i of VALMP, surely computed with
          previous lengths, is larger than the actual, update it
3      if (VALMP.distances[i] > lNormDist or VALMP.MP[i] == ⊥) then
4          entry pair;
5          pair.off1 ← i, pair.off2 ← IP[i] ;
6          pair.distance ← MPnew[i], pair.ℓ ← ℓ;
7          pair.partDP1 ← ⊥, pair.partDP2 ← ⊥ ;
8          insert(heapBestKPairs, pair);
9          VALMP.distances[i] ← MPnew[i];
10         VALMP.normDistances[i] ← lNormDist;
11         VALMP.lengths[i] ← ℓ;
12         VALMP.indices[i] ← IP[i];
13 for each pair in heapBestKPairs do
14     if (pair.partDP1== ⊥) then
15         pair.partDP1 ← listDP[pair.off1];
16         pair.partDP2 ← listDP[pair.off2];
```

In lines 4 to 7, we build a structure named *pair*, which carries the information of the subsequences pairs that appear in the $VALMP$ structure. During this iteration, we leave the fields $partDP1$ and $partDP2$ empty, since they will be later initialized with the partial distance profiles, if their *pair* is in the top k of $VALMP$. In order to enumerate the best k pairs, we use the global maximum heap $heapBestKPairs$ in line 8. Then, we assign (or update) the corresponding partial distance profiles (line 13) to each pair.

We are now ready to present the variable length motif sets discovery algorithm (refer to Algorithm 11). Starting at line 1, the algorithm iterates over the best pairs. For each one of those, we need to check if the search range is smaller than the maximum lower bound distances of both partial distance profiles. If this is true, we are guaranteed to have already computed all the subsequences in the range. Therefore, in lines 7 and 13 we filter the subsequences in the range, sorting the partial distance profile according to the offsets. This operation will permit us to find the trivial matches in linear time.

Algorithm 11: *computeVarLengthMotifSets*

Input: **DataSeries** T, **MaxHeap** $heapBestKPairs$, **double** D
Output: **Set** S^*

```
1  for each pair in heapBestKPairs do
2      double r ← pair.distance * D ;
3      double maxLB1 ← popMax(pair.partDP1);
4      double maxLB2 ← popMax(pair.partDP2);
5      D(T_{pair.off1,pair.ℓ}) ← ∅, D(T_{pair.off2,pair.ℓ}) ← ∅ ;
6      if maxLB1 > r then
           // sort according the offset, the partial distance profile
              contains all the elements in the range
7          D(T_{pair.off1,pair.ℓ}) ← sortAndFilterRange(r,pair.partDP1.toVector());
8      else
           // re-compute the mat
9          double s ← sum(T_{ind,ℓ});
10         double ss ← squaredSum(T_{ind,ℓ});
11         D(T_{pair.off1,pair.ℓ}) ← CalcDistProfInRange(r,QT,T_{pair.off1,pair.ℓ}, T, s, ss);
12     if maxLB2 > r then
13         D(T_{pair.off2,ℓ}) ← sortAndFilterRange(r,pair.partDP2.toVector());
14     else
15         double s ← sum(T_{ind,ℓ});
16         double ss ← squaredSum(T_{ind,ℓ});
17         D(T_{pair.off2,pair.ℓ}) ← CalcDistProfInRange(r,QT,T_{pair.off1,pair.ℓ}, T, s, ss);
18     Set S_r^{pair.ℓ} ← mergeRemoveTM(D(T_{pair.off1,ℓ}), D(T_{pair.off2,ℓ}));
19     S*.add(S_r^{pair.ℓ});
```

On the other hand, if the search range is larger than the maximum lower bound distances of both partial distance profiles, we have to re-compute the entire distance profile (lines 11 and 17), to find all the subsequences in the range. Once we have the distance profile pairs, we need to merge them and remove the trivial matches (line 18). Each time we add a subsequence in a motif set, we remove it from the search space: this guarantees the empty intersection among the sets in S^*.

Complexity Analysis. The complexity of the *updateVALMPForMotifSets* algorithm is $\mathcal{O}(n \log(k))$, where n is the length of the $VALMP$ structure, which is linearly scanned and updated. $\mathcal{O}(\log(k))$ time is needed to retain the k best pairs of $VALMP$, using the heap structure in line 8. The final algorithm *computeVarLengthMotifSets* takes $\mathcal{O}(k \times p \times \log(p))$ time, in the best case.

This occurs when, after iterating the k pairs in $heapBestKPairs$, each partial distance profile of length p, contains all the elements in the range r. In this case, we just need an extra $\mathcal{O}(p \log(p))$ time to sort its elements (line 7 and 13). On the other hand, the worst case time is bounded by $\mathcal{O}(k \times n \times \log(n))$, where n is the length of the input data series T. In this case, the algorithm needs to recompute k times the entire distance profile (line 11 and 17), at a unit cost of $\mathcal{O}(n \log(n))$ time.

5.6 Discord Discovery

We now describe our approach to solving the Variable-Length *Top-k* m^{th} Discord Discovery problem. First, we explain some useful notions, and we then present our discord discovery algorithm.

5.6.1 Comparing Discords of Different Lengths

Before introducing the algorithm that identifies discords (from the *Top-1* 1^{st} to the *Top-k* m^{th} one), we define the data structure that allows us to accommodate them. We can represent this structure as a $k \times m$ matrix, which contains the best match distance and the offset of each discord.

More formally, given a data series T, and a subsequence length ℓ we define: $dkm_\ell = \begin{bmatrix} \langle d, o \rangle_{1,1} & .. & \langle d, o \rangle_{1,m} \\ .. & .. & .. \\ \langle d, o \rangle_{k,1} & .. & \langle d, o \rangle_{k,m} \end{bmatrix}$, where a generic pair $\langle d, o \rangle_{i,j}$ contains the offset o and the corresponding distance d of the *Top-i* j^{th} discord of length ℓ ($1 \leq i \leq k$ and $1 \leq j \leq m$). In dkm_ℓ, rows rank the discords according to their positions (m^{th} discords), and the columns according to their best match distance ($Top\text{-}k$). For each pair $\langle d, o \rangle_{a,b}$, $\langle d', o' \rangle_{a',b'} \in dkm_\ell$, we require that $T_{o,\ell}$ and $T_{o',\ell}$ are not trivial matches.

Since we want to compute dkm_ℓ for each length in the range $[\ell_{min}, \ell_{max}]$, we also need to rank discords of different lengths. In that regard, we want to obtain a unique matrix that we denote by $dkm_{\ell_{min},\ell_{max}}$. Therefore, we can represent a discord by the triple $\langle d^*, o^*, \ell^* \rangle_{i,j} \in dkm_{\ell_{min},\ell_{max}}$, where d^* is the i^{th} greatest

length normalized j^{th} best match distance. More formally: $d^* = max\{\frac{d}{\sqrt{\ell_{min}}} : d \in dkm_{\ell_{min}}(i,j), ..., \frac{d}{\sqrt{\ell_{max}}} : d \in dkm_{\ell_{max}}(i,j)\}$. Each triple is also composed by the offset o^* and the length ℓ^* of the discord, where $\ell_{min} \leq \ell^* \leq \ell_{max}$.

By multiplying by the $1/\sqrt{\ell}$ ratio each distance, we want to favor the selection of shorter discords. This strategy is based on the following fact: if we compare two *Top-k* m^{th} discord subsequences of different lengths, but equal best match distances, the shorter subsequence is the one with the highest point-to-point dissimilarity to its best match. Consequently, we promote the shorter subsequence as the more anomalous one.

5.6.2 Discord Discovery Algorithm

We now describe our algorithm for the *Top-k* m^{th} discords discovery problem. We note that we can still use the lower bound distance measure, as in the motif discovery case. This allows us to efficiently build dkm_ℓ, for each ℓ in the $[\ell_{min}, \ell_{max}]$ range, incrementally reusing the distances computation performed. The final outcome of this procedure is the $dkm_{\ell_{min},\ell_{max}}$ matrix, which contains the variable length discord ranking. In this part, we introduce and explain the algorithms, which permit us to efficiently obtain dkm_ℓ for each length. We report the whole procedure in Algorithm 12.

Smallest Length Discords. We start to find discords of length ℓ_{min}, namely the smallest subsequence length in the range. We can thus run Algorithm 8 in line 1, which computes the list of partial distance profiles of each subsequence of length ℓ_{min} ($listDP$), in the input data series T. Each partial distance profile contains the p smallest nearest neighbor distances of each subsequence. To that extent, we set $p \geq m$ in Algorithm 8 ($ComputeMatrixProfile$).

We then iterate the subsequences of T in line 6, using the index i. For each subsequence $T_{i,\ell_{min}}$ that has no trivial matches in $dkm_{\ell_{min}}$, we invoke the routine $UpdateFixedLengthDiscords$ (line 8), which checks if $T_{i,\ell_{min}}$ can be placed in $dkm_{\ell_{min}}$ as a discord. When $dkm_{\ell_{min}}$ is built, we update the variable length discords ranking ($dkm_{\ell_{min},\ell_{max}}$ matrix in line 9), using the procedure $UpdateVariableLengthDiscords$.

In the loop of line 10, we iterate the discord lengths greater than ℓ_{min}. Since we

Algorithm 12: *Topkm_DiscordDiscovery* (*Compute Top-k m^{th} Discords of variable lengths*)

Input: DataSeries T, **int** ℓ_{min}, **int** ℓ_{max}, **int** k, **int** m , **int** p
Output: Matrix $dkm_{\ell_{min},\ell_{max}}$

```
1  MaxHeap[] listDP=ComputeMatrixProfile(T, ℓ_min, p);
2  int nDp = (|T| − ℓ_min) + 1;
3  Matrix dkm_{ℓ_min,ℓ_max} = {{⟨−∞, −∞, −∞⟩, ..., ⟨−∞, −∞, −∞⟩}, ..., {...}};
4  Matrix dkm_{ℓ_min} = {{⟨−∞, −∞⟩, ..., ⟨−∞, −∞⟩}, ..., {...}};
5  if p >= m then
       // iterate the partial distance profiles in listDP
       // and compute dkm_{ℓ_min}
6      for i ← 1 to nDp do
7          if T_{i,ℓ_min} has no Trivial matches in dkm_{ℓ_min} then
8              UpdateFixedLengthDiscords(dkm_{ℓ_min}, listDP[i],i,k,m );
9      UpdateVariableLengthDiscords(dkm_{ℓ_min}, dkm_{ℓ_min,ℓ_max},k,m );
       // compute dkm_{ℓ_nextL} for each length, pruning distance computations
10     for nextL ← ℓ_min + 1 to ℓ_max do
11         Matrix dkm_nextL = {{⟨−∞, −∞⟩, ..., ⟨−∞, −∞⟩}, ..., {...}};
12         nDp = (|T| − nextL) + 1;
13         dkm_{ℓ_nextL}=Topkm_nextLength( T,nDp,listDP,nextL,k,m);
14         UpdateVariableLengthDiscords(dkm_{ℓ_nextL}, dkm_{ℓ_min,ℓ_max},k,m);
```

want to prune the search space, we consider the list of distance profiles in $listDP$, which also contains the lower bound distances of the p ($p > m$) nearest neighbors of each subsequence. In that regard, we invoke the routine $Topkm\_nextLength$ (line 13). Before we introduce the details, we describe the two routines we introduced, which allow to rank the discords.

Ranking Fixed Length Discords. In algorithm 13, we report the pseudo-code of the routine $UpdateFixedLengthDiscords$. This algorithm accepts as input the matrix dkm_ℓ to update, and a partial distance profile of the subsequence with offset off. It starts iterating the rows of $dkm_{\ell_{min}}$ in reverse order (line 1). This is equivalent to considering the discords from the m^{th} one to the 1^{st}. Hence, at each iteration we get the j^{th} nearest neighbor of $T_{off,\ell_{min}}$ from its partial distance profile in line 2. Subsequently, the loop in line 3 checks if the $j^{th}dist$ is among the k largest ones in the j^{th} column of $dkm_{\ell_{min}}$. If it is true, the smallest elements in the column are shifted (line 6) and $T_{off,\ell_{min}}$ is inserted as the $Top\text{-}i$ j^{th} discord (line 7).

Algorithm 13: *UpdateFixedLengthDiscords (Update dkm_ℓ)*

Input: **Matrix** dkm_ℓ, **MaxHeap** $minMDist$, **int** off, **int** k, **int** m

```
1 for j ← m down to 1 do
2     double j^th dist ← minMDist.getMax(j);
3     for i ← 1 to k do
4         < d, o >_{i,j} = dkm_newL[i][j];
5         if j^th dist > d then
6             shiftRankingTopK(dkm_newL[i][j]);
              // update the ranking with the new Top-i j^th discord T_{off,ℓ}
7             dkm_ℓ[i][j] ← ⟨j^th dist, off⟩;
8             return;
```

Algorithm 14: *UpdateVariableLengthDiscords (Update $dkm_{\ell_{min},\ell_{max}}$)*

Input: **Matrix** $dkm_{\ell_{min},\ell_{max}}$, **Matrix** dkm_ℓ, **int** k, **int** m

```
1 for i ← 1 to k do
2     for j ← 1 to m do
3         < d, o >_{i,j} = dkm_newL[i][j];
4         < d*, o*, l* >_{i,j} = dkm_{ℓ_min,ℓ_max}[i][j];
          // if length normalized distance is greater or equal for length ℓ,
             update the rank.
5         if ((d/√ℓ) >= d* ) then
6             dkm_{ℓ_min,ℓ_max}[i][j] = ⟨(d/√ℓ), o, ℓ⟩
```

Ranking Variable Length Discords. Once we dispose of the matrix dkm_ℓ, we can invoke the procedure $UpdateVariableLengthDiscords$ for each length $\ell \in \{\ell_{min}, ..., \ell_{max}\}$ (Algorithm 14), in order to incrementally produce the final variable length discord ranking we store in $dkm_{\ell_{min},\ell_{max}}$. This algorithm accepts as input and iterates over the matrix $dkm_{\ell_{min},\ell_{max}}$. A position (discord) is updated if the length normalized best match distance of the discord in the same position of dkm_ℓ is larger (line 6).

Greater Length Discords. In Algorithm 15, we show the pseudo-code of the routine $Topkm\_nextLength$. It starts performing the same loop of line 9 in Algorithm 6, iterating over the partial distance profiles (line 3), and updating the true Euclidean distances for the new length ($newL$) and the lower bounds (line 9) for the subsequent length ($newL + 1$). Since we need to know the distances from each subsequence to their m nearest neighbors, for each subsequence $T_{i,newL}$ that does not have trivial matches in dkm_{newL}, we check if the m^{th} smallest distance is

smaller than the maximum lower bound in the partial distance profile (line 13). If this is true, we have the guarantee that the partial distance profile $minMDist$ contains the exact m nearest neighbor Euclidean distances. Hence, in line 14, we can update the matrix dkm_{newL}. On the other hand, if the distances are not verified to be correct, we keep $minMDist$ in memory, which becomes a non-valid partial distance profile, along with the offset of the corresponding subsequence (line 16). Once we have considered all the partial distance profiles, we need to iterate the non-valid partial distance profiles (line 17).

We therefore recompute those that contain at least one true Euclidean distance greater than the distances in the last row of dkm_{newL}. The correctness of this choice is guaranteed by the fact that the distances of a non-valid partial distance profile can be only larger than the non-computed ones. Hence, if the condition of line 21 is not verified, no updates in dkm_{newL} can take place. Otherwise, we recompute the non-valid distance profile starting at line 22 from scratch. Note that when we re-compute a distance profile, we globally update the corresponding position of the partial distance profiles $listDP$ (line 25) and dkm_{newL} in the vector as well (line 26).

Complexity Analysis. The time complexity of Algorithm 12 ($Topkm\_DiscordDiscovery$) mainly depends on the use of $ComputeMatrixProfile$ algorithm, which always takes $\mathcal{O}(n^2 \log(p))$ to compute the partial distance profiles for the n subsequences of length ℓ_{min} in T.

In order to compute the exact *Top-k* m^{th} discord ranking in dkm_{ℓ}, the routine $UpdateFixedLengthDiscords$ takes $\mathcal{O}(km)$ time in the worst case. Recall that this latter algorithm is called only for subsequences that do not have trivial matches in dkm_{ℓ}. Checking if two subsequences are trivial matches takes constant time, if for each dkm_{ℓ} update, we store the ℓ trivial match positions. Given a series T, and the discord (subsequence) length ℓ, we can represent by $S = \frac{|T|}{l/2}$, the number of subsequences that are not trivial matches with one another. Therefore, updating the discord rank of each length has a worst case time complexity of $\mathcal{O}((\ell_{max} - \ell_{min}) \times S \times \ell \times k \times m \times \log(m))$, where the $\log(m)$ factor represents the time to get the m^{th} largest distance in the partial distance profile (line 2 of Algorithm 13). Similarly, the construction of the variable length discord ranking in $dkm_{\ell_{min},\ell_{max}}$ takes: $\mathcal{O}((\ell_{max} - \ell_{min}) \times k \times m)$.

Observe also that the time performance of the $Topkm\_nextLength$ algorithm depends on the Euclidean distance computations pruning. If all the partial distance profiles contain the correct nearest neighbor's distances, computing the discords

Algorithm 15: *Topkm_nextLength* (Compute *Top-k* m^{th} Discords of greater lengths)

Input: **DataSeries** T, **int** nDp, **MaxHeap[]** $listDP$, **int** $newL$, **int** k, **int** m, **int** p
Output: **Matrix** dkm_{newL}

```
1  Matrix dkm_newL = {{⟨−∞,−∞⟩,...,⟨−∞,−∞⟩},...,{...}};
2  List ⟨MaxHeap,int⟩ nonValidMindistList;
   // iterate over the partial distance profiles in listDP
3  for i ← 1 to nDp do
4      MaxHeap minMDist ← new MaxHeap(p);
5      double minDist ← inf;
6      int ind ← 0;
7      double maxLB ← popMax(listDP[i]);
       /* update the partial distance profile for the length newL (true Euclidean
          and lower bounding distance )                                          */
8      for each entry e in listDP[i] do
9          e.dist, e.LB ← updateDistAndLB(e, newL);
           // the m shortest neighbor distances are stored in minMDist
10         minMDist.push(e.dist);
       // check if the m^th shortest distance of this partial distance profile is
          the true m^th shorthest.
11     mDist = minMDist.getMax(m);
12     if T_{i,newL} has no Trivial matches in dkm_newL then
13         if mDist < maxLB then
               /* the discord ranking can be updated, without computing the whole
                  distance profile                                               */
14             UpdateFixedLengthDiscords(dkm_newL, minMDist,i,k,m);
15         else
               /* minMDist might not be exact, store the partial distance profile in
                  memory.                                                        */
16             nonValidMindistList.add(< minMDist,i >);
17 for each < minMDist, i > in nonValidMindistList do
18     if T_{i,ℓ} has no Trivial matches in dkm_ℓ then
19         for j ← m down to 1 do
20             mDist = minMDist.getMax(j);
21             if mDist > dkm_newL[k][j].d then
22                 QT ← SlidingDotProduct(T_{i,newL}, T);
23                 double s ← sum(T_{ind,ℓ}); double ss ← squaredSum(T_{i,newL});
24                 D(T_{ind,ℓ}) ← CalcDistProfAndLB(QT,T_{i,newL}, T, s, ss);
25                 UpdatePartialDistanceProfile(listDP[i], D(T_{ind,ℓ})) ;
26                 UpdateFixedLengthDiscords(dkm_newL, listDP[i],i,k,m);
27                 break;
```

of each length greater than ℓ_{min} takes $\mathcal{O}(n \times p \times \log(m))$ time, with n equal to the number of subsequences in T. The worst case takes place when for each subsequence that can update dkm_ℓ (i.e., S), the complete distance profile is re-

computed (Algorithm 15, line 22); in this case the algorithm takes $\mathcal{O}(n^2 \times \log(n) \times p \times \log(m))$. In our experimental evaluation, we show that in all the cases we tested, the percentage of the recomputed distance profiles is indeed very low.

5.7 Experimental Evaluation

5.7.1 Setup

We implemented our algorithms in C (compiled with gcc 4.8.4), and we ran them in a machine with the following hardware: Intel Xeon E3-1241v3 (4 cores - 8MB cache - 3.50GHz - 32GB of memory). All of the experiments we present in this part are reproducible. In that regard, we reported the analyzed datasets and source code on a dedicated web page [55].

Datasets and Benchmarking Details. To benchmark our algorithm, we use five datasets:

- GAP, which contains the recording of the global active electric power in France for the period 2006-2008. This dataset is provided by EDF (main electricity supplier in France) [53];
- CAP, the Cyclic Alternating Pattern dataset, which contains the EEG activity occurring during NREM sleep phase [93];
- ECG and EMG signals from stress recognition in automobile drivers [31];
- ASTRO, which contains a data series representing celestial objects [19].

Table 5.1 summarizes the characteristics of the datasets we used in our experimental evaluation. For each dataset, we report the minimum and maximum values, the overall mean and standard deviation, and the total number of points.

The (CAP),(ECG) and (EMG) datasets are available in [17]. We use several prefix snippets of these datasets, ranging from 0.1M to 1M of points.

In order to measure the scalability of our motif discovery approach, we test its performance along four dimensions, which are depicted in Table 5.2. Each experiment is conducted by varying the parameter of a single column, while for the

	MIN	MAX	MEAN	STD-DEV	number of points
ECG	-2.182	1.543	0.006	0.24	1M
GAP	0.08	10.67	1.10	1.15	2M
ASTRO	-0.00867	0.00447	0.00003	0.00031	2M
EMG	-0.694	0.773	-0.005	0.041	1M
EEG	-966	920	3.34	41.36	0.5M

Table 5.1: Characteristics of the datasets used in the experimental evaluation.

Motif length (ℓ_{min})	Motif range ($\ell_{max} - \ell_{min}$)	Data series size (points)	p (elements of distance profiles stored)
256	**100**	0.1 M	5
512	150	0.2 M	10
1024	200	**0.5 M**	15
2048	400	0.8 M	20
4096	600	1 M	**50** , 100 , 150

Table 5.2: Parameters of VALMOD benchmarking (default values shown in bold).

others, the default value (in bold) is selected. In our benchmark, we have two types of algorithms to compare to VALMOD. The first are two state-of-the-art motif discovery algorithms, which receive a single subsequence length as input: QUICKMOTIF [51] and STOMP [109]. In our experiments, they have been run iteratively to find all the motifs for a given subsequence length range. The other approach in the comparative analysis is MOEN [67], which accepts a range of lengths as input, producing the best motif pair for each length.

For VALMOD, we report the total time, including the time to build the matrix profile (Algorithm 8).

5.7.2 Motif Discovery Results

Scalability over Motif Length. In Figure 5.8, we depict the performance results of the *four* motif discovery approaches, when varying the motif length. We note

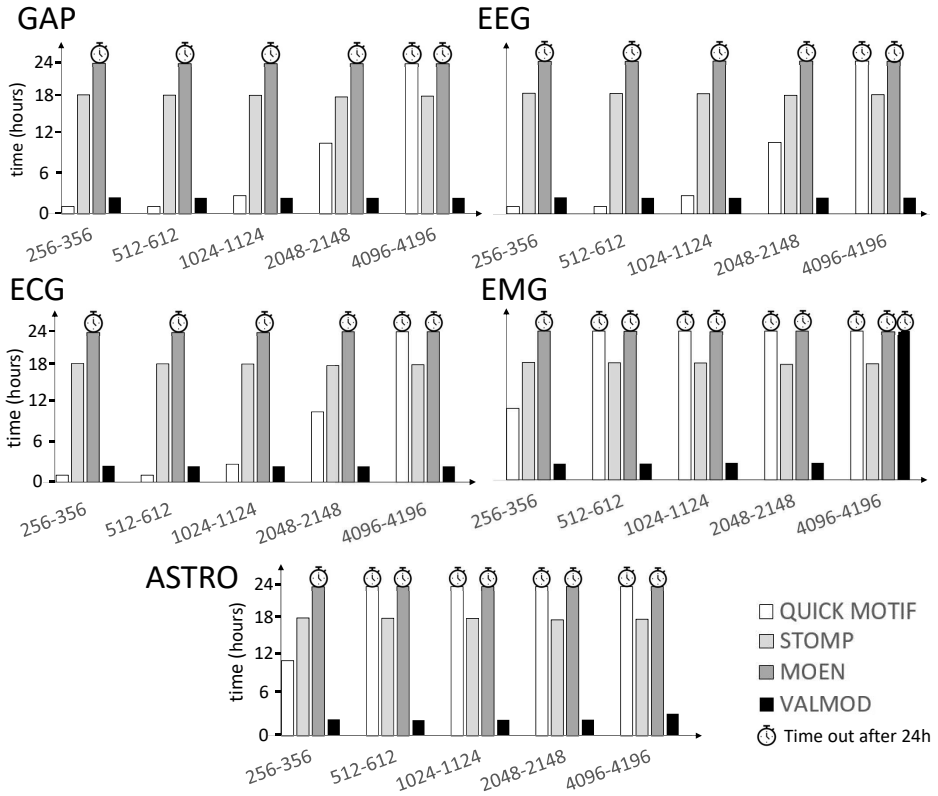


Figure 5.8: Scalability for various motif length ranges.

that the performance of VALMOD remains stable over the five datasets. On the other hand, we observe that a pruning strategy based on a summarized version of the data is sensitive to subsequence length variation. This is the case for QUICK MOTIF, which operates on PAA (Piecewise Aggregate Approximation) discretized data. Figure 5.8 shows that the performance of QUICK MOTIF varies significantly as a function of the motif length range, growing rapidly as the range increases, and failing to finish within a reasonable amount of time in several cases.

Moreover, we argue that our proposed lower bounding measure enables our method to improve upon MOEN, which clearly does not scale well in this experiment (see Figure 5.8). The main reason for this behavior is that the effectiveness of the lower bound of MOEN decreases very quickly as we increase the subsequence length ℓ. When we increase the subsequence length by 1, MOEN multiplies the lower bound by a value smaller than 1 ([67], Section IV.B), thus making it less tight. In contrast, the lower bound of VALMOD does not always decrease (refer to Eq. 5.2): $\frac{\sigma_{j,l}}{\sigma_{j,l+k}}$ may be larger than 1. Consequently, the lower bound of VALMOD can remain

effective (i.e., tight) even after several steps of increasing the subsequence length.

Concerning the VALMOD performance, we note a sole exception that appears for the noisy EMG data (Figure 5.8), for a relatively high motif length range (4096-4196). The explanation for this behavior is that the lower bounding distance used by VALMOD is coarse, or in other words, it is not a good approximation of the true distance. Figure 5.9 shows the difference between the greater lower bounding distance ($maxLB$) and the smaller true Euclidean distance for each distance profile. We use the subsequence lengths *356* and *4196*, which are respectively the range's smallest and largest extremes in this experiment. In this last plot, each value greater than 0 corresponds to a valid condition in line 14 of the *ComputeSubMP* algorithm. This indicates that we found the smallest value of a distance profile, while pruning computations over the entire subsequence length range. As the subsequence length increases, VALMOD's pruning becomes less effective for the EMG (observe that there are no, or very few values above zero in the distances profiles for subsequence length *4196*). On the other hand, we observe the presence of values above zero in the other datasets. This confirms that motifs in those cases are found, while pruning the search space.

In order to further evaluate the pruning capability of VALMOD, we report the measurements for the Tightness of the Lower Bound (TLB) [88, 115] performed during the previous experiment (Figure 5.8). The TLB is a measure of the lower bounding quality; given two data series t_1 and t_2, the TLB is computed as follows: $LB_{dist}(t_1, t_2)/EuclideanDistance(t_1, t_2)$. Note that TLB takes values between 0 and 1. A TLB value of 1 means that the lower bound distance is the same as the Euclidean distance; this corresponds to the optimal case.

In Figure 5.10, we show the average TLB for each (partial) distance profile. In the EMG dataset, when using the larger subsequence length, we observe a sharp decrease of the lower bounding quality (small TLB values), explaining the behavior observed for the EMG dataset (refer to Figure 5.9(top-left)). We also note similar results for the ASTRO dataset. As we have noted for this last case, the performance is not negatively affected, since we dispose of several partial distance profiles that provide the correct minimum distances, and thus permit us to find the motifs, without recomputing all the distance profiles. In contrast, in the other datasets, we note a smaller negative impact on TLB for the case of subsequence length *4196*.

In Figure 5.11, we also show the distance distribution of the pairwise subsequences, using the same datasets and subsequences lengths. Here, we plot the distances

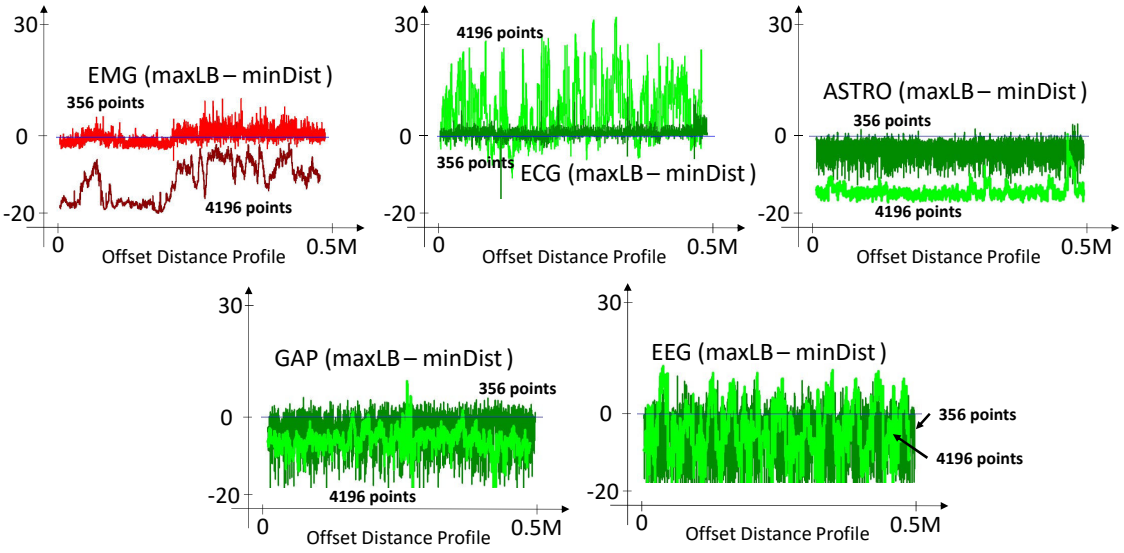


Figure 5.9: The difference between the max lower bounding distance (maxLB) and the min Euclidean distance of partial distance profiles in all the datasets. Subsequence lengths: 356/4196.

without length normalization, since the algorithm uses it to rank the motifs in the trailing part. For the EMG and ASTRO datasets, in the case of length *4196*, the distance distribution includes many small and large values, which does not suggest the presence of motifs, but affects VALMOD negatively. Observe that in the other datasets, the values are more uniformly distributed over all the subsequence lengths. This denotes the presence of subsequence pairs that are substantially closer than the rest, which typically identifies the occurrence of motifs. In this case, VALMOD is able to prune more distance profile computations, leading to better performance.

Scalability Over Motif Range. In Figure 5.12, we depict the performance results as the motif range increases. VALMOD gracefully scales on this dimension, whereas the other approaches can seldom complete the task. Not only does our technique address the intrinsic problem of STOMP and QUICK MOTIF, which independently process each subsequence length, but it also exhibits a substantial improvement over MOEN, the existing state-of-the-art approach for the discovery of variable length motifs.

Scalability Over Data Series Length. In Figure 5.14, we experiment with different data series sizes. For the EEG dataset we only report three measurements,

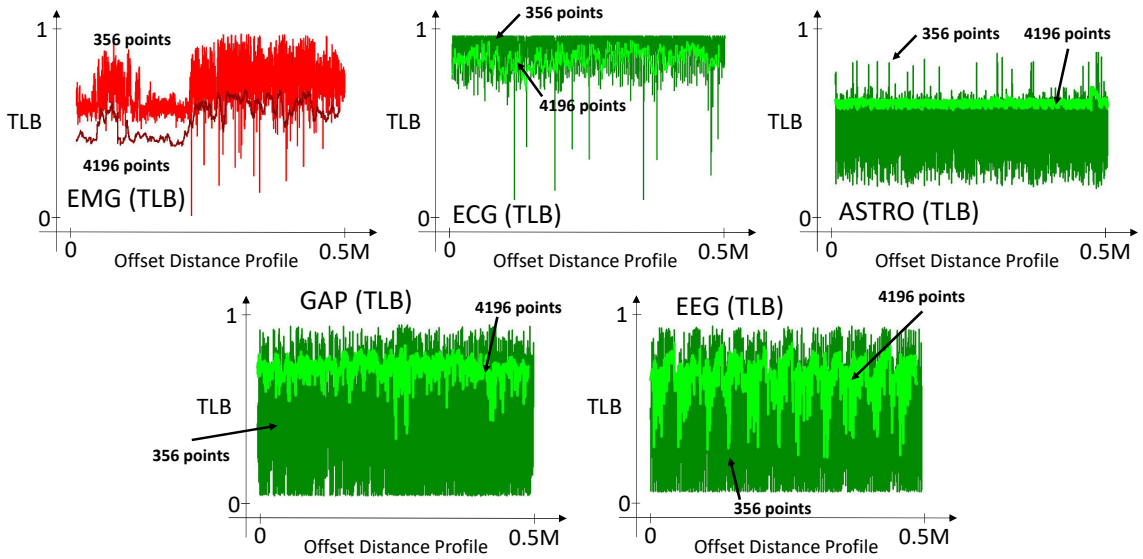


Figure 5.10: Average of the tightness of the lower bounding (TLB) for every Distance profile of all the datasets for subsequence lengths: 356/4196.

since this collection contains no more than 0.5M points. We observe that QUICK MOTIF exhibits high sensitivity, not only to the various data sizes, but also to the different datasets (as in the previous case, where we varied the subsequence length). It is also interesting to note that QUICK MOTIF is slightly faster than VALMOD on the ECG dataset, which contains regular and similar heartbeat patterns, and is a relatively easy dataset for motif discovery. Nevertheless, QUICK MOTIF, as well STOMP and MOEN, fail to terminate within a reasonable amount of time for the majority of our experiments. On the other hand, VALMOD does not exhibit any abrupt changes in its performance, scaling gracefully with the size of the dataset, across all datasets and sizes.

Large Datasets and Length ranges. Here we report here two further experiments that we have conducted on larger snippets of the datasets - namely, *2* million points - and over a larger range of motif lengths. To that extent, we want to test the scalability of our approach, considering two *extreme* cases. We compare VALMOD to QUICKMOTIF, since the latter is the sole approach that can scale to data series lengths beyond half a million points, and to motif length ranges larger than *100*.

In Figure 5.13.(a), we report the motif discovery time on four datasets that contain *2* million points. We pick the default length boundaries, namely $\ell_{min} = 1024$ and $\ell_{max} = 1124$, discovering motifs of each length in between them. The results show

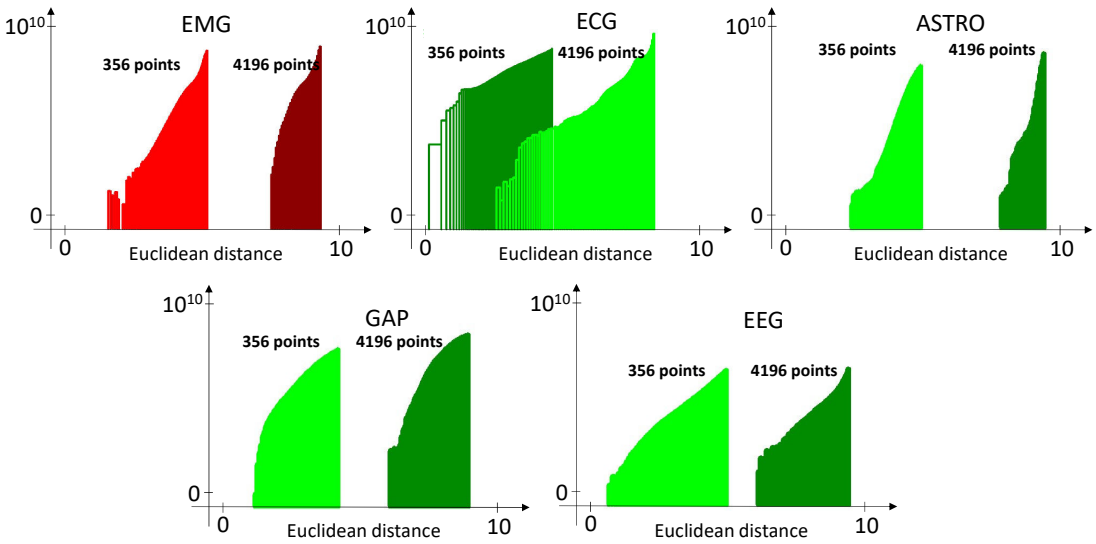


Figure 5.11: Distribution of Euclidean distance of pairwise subsequences in all the datasets. Subsequence lengths: 356/4196.

that VALMOD gracefully scales, and is always one order of magnitude faster than QUICKMOTIF, which does not reach the timeout only in the case of the ECG datasets.

The same observations hold for the results of the experiments that vary the motif length range. Figure 5.13.(b), shows the results for length ranges *2000* and *4000*, on all five datasets in our study (at their default sizes). Once again, QUICKMOTIF reaches the timeout state in all datasets, except for ECG, where for the larger length ranges is two times slower than VALMOD. On the other hand, VALMOD scales well and remains the method of choice (with the exception of the largest length ranges for the EMG and ASTRO datasets, where it reaches the timeout).

The above results demonstrate the superiority of VALMOD, but also show its limits, which open possibilities for future work.

Overall Pruning Power. In order to show the global effect of VALMOD's pruning power, we conduct an experiment recording the number of distance profile computations performed by procedure *ComputeSubMP*, which extracts motifs of length greater than ℓ_{min}, pruning the unpromising calculations. We recall that this algorithm computes for each subsequence $T_{i,\ell}$ with $\ell > \ell_{min}$ a subset of distances

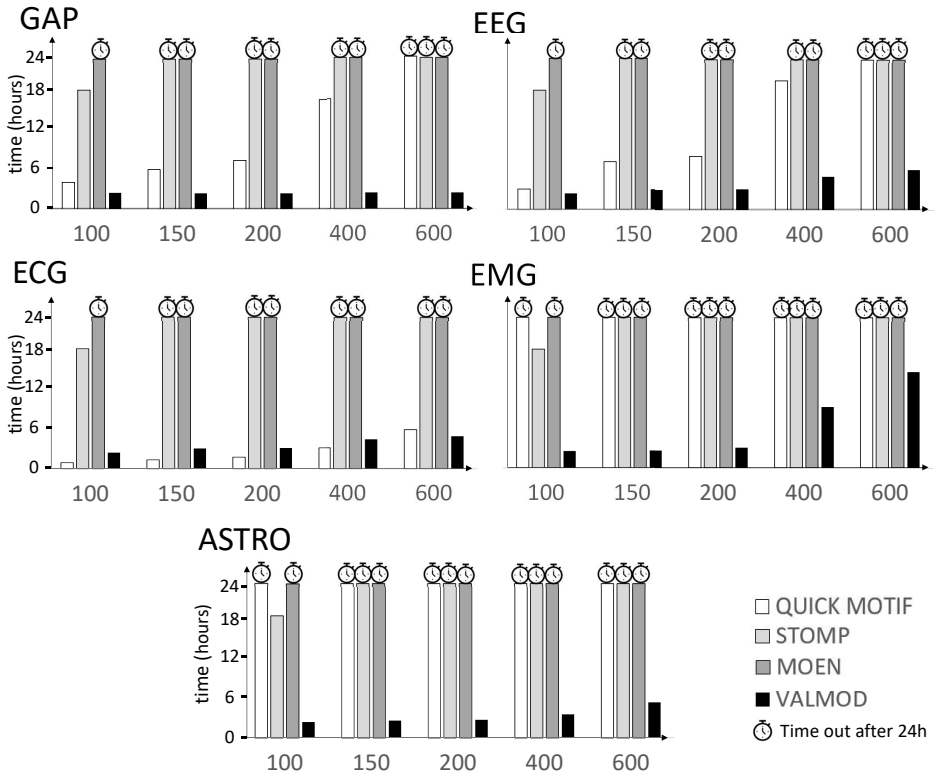


Figure 5.12: Scalability with increasing motif range.

(Euclidean and lower bounding), called partial distance profiles. If the smallest Euclidean distance computed is also smaller than the larger lower bounding distance, we know it is the true distance of the nearest neighbor of $T_{i,\ell}$. In this case, we call the partial distance profile *valid*. Otherwise, we do not know the true nearest neighbor distance, and we call the partial distance profile *non-valid*. In order to identify the correct motifs, the algorithm only needs to recompute the entire *non-valid* distance profiles that might contain distances shorter than those already found in the *valid* distance profiles.

In Figure 5.15, we depict the difference between the minimum Euclidean distance and the maximum lower bounding distance of each distance profile computed in the subsequence length range (*1025/1124*). In the plots, the values above zero refer to the *valid* ones (green points), whereas values under zero are either *non-valid* (black points) or *recomputed* (red/triangular points). We observe that in the first three datasets, namely EEG, ECG and GAP, there are no distance profiles that are recomputed, meaning that the motifs are always found in the *valid* (partial) distance profiles in the shortest time possible (*best case*). Concerning the EMG

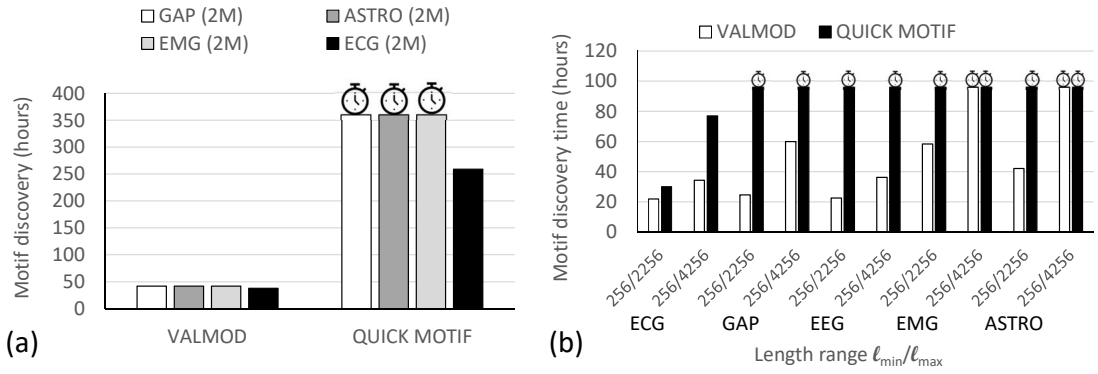


Figure 5.13: Scalability of VALMOD and QUICKMOTIF using large datasets (2M of points) and large length ranges.

and ASTRO datasets, several re-computations take place (red/triangle points). As we can see from the table in the bottom part of Figure 5.15 though, the computed distance profiles are not more than the *0.20%* of the total. This means that the algorithm successfully prunes a high percentage of the computations, thanks also to the effectiveness of the proposed lower bounding measure.

At this point, we can further analyze the reasons behind the pruning capability of our approach. To that extent, in Figure 5.16.(a) we plot the number of distance profiles that VALMOD recomputes at each subsequence length for the EMG and ASTRO datasets. These two datasets both contain noisy data, which influence re-computations. However, they differ according to the length for which these re-computations take place.

Figure 5.16.(b) shows the position of the $Top-1$ motif along the subsequence length. Note that the $Top-1$ motif is always placed around the same offset region in the ASTRO dataset, suggesting the presence of a few similar data segments, which is also verified by the high number of *non-valid* distance profiles we observe in Figure 5.15(ASTRO). On the other hand, in the EMG dataset, the motif location changes several times, denoting the presence of different segments, which contain motifs of different lengths. This is also confirmed by the more prevalent presence of *valid* distance profile in the EMG dataset. In this last case, the re-computation number drops to zero as soon as the motif positions start to change, i.e., at length *1058*, maintaining the same trend until the end.

Effect of Changing Parameter p. In Figure 5.17, we study the effect of param-

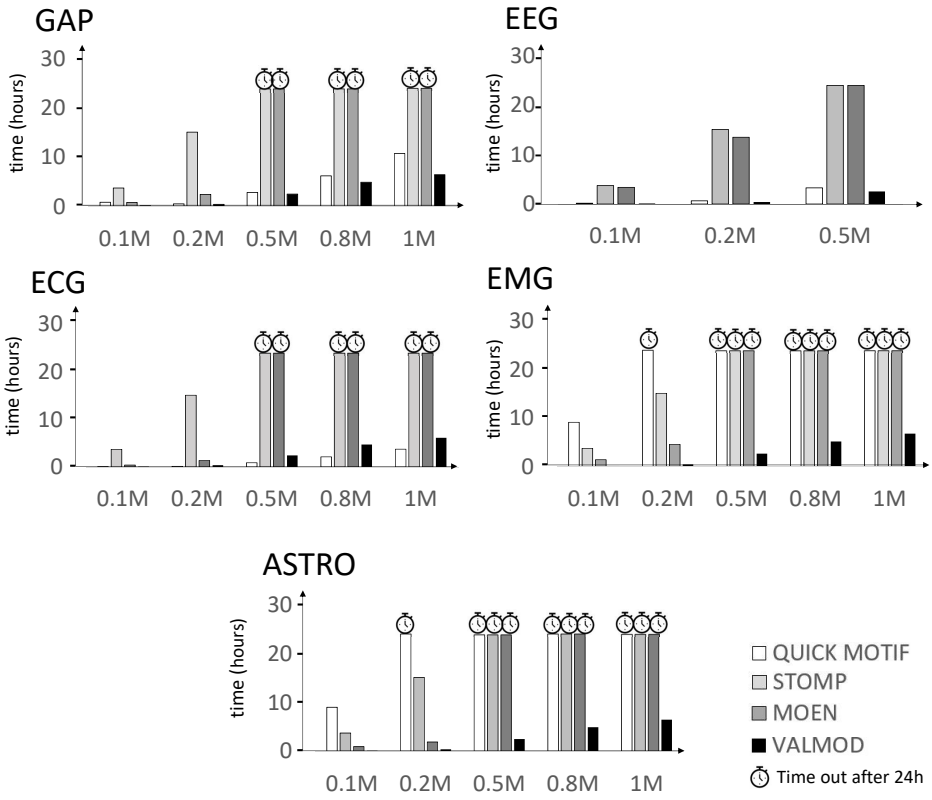
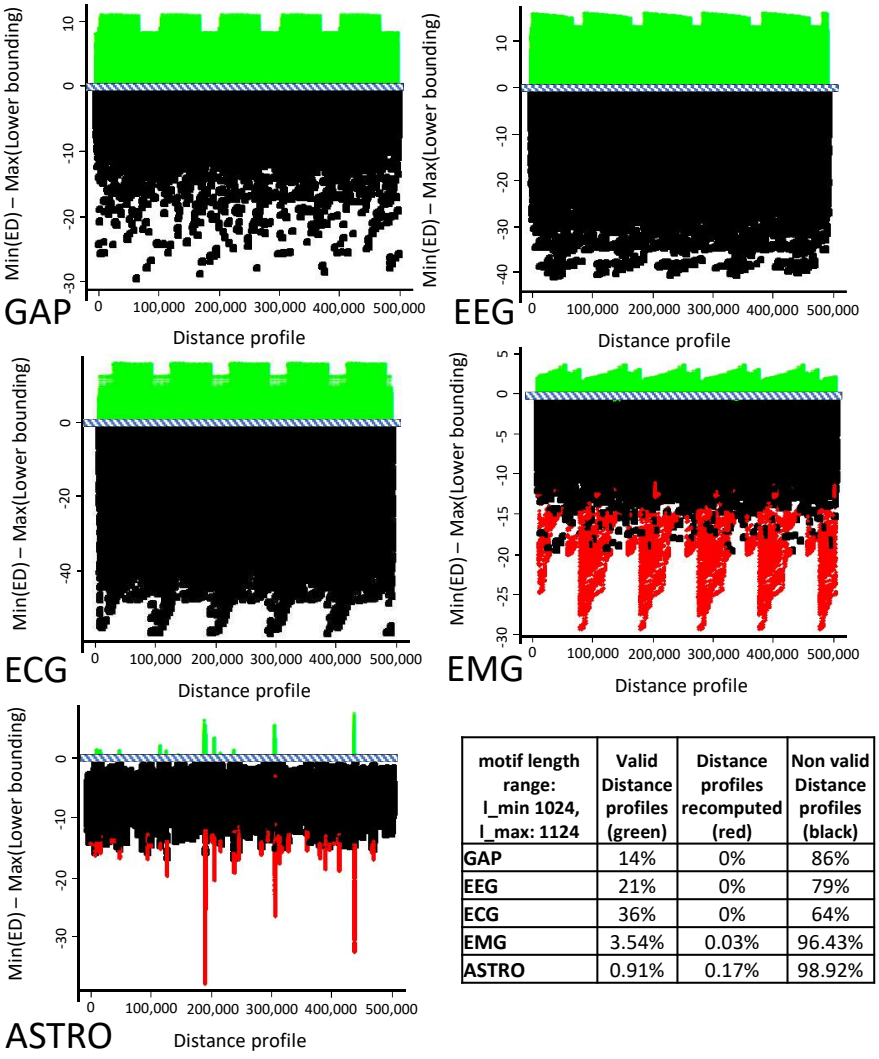


Figure 5.14: Scalability with increasing data series size.

eter p on VALMOD's performance. The p value determines how many distance profile entries we compute and keep in the memory. Increasing p leads to increased memory consumption, but could also translate to an overall speed-up, since having more distances may guarantee a larger margin between the greater lower bounding distance and the minimum true Euclidean distance in a distance profile. As we can see on the left side of the plot, increasing p does not provide any significant advantage in terms of time complexity. Moreover, the plots on the right-hand side of the figure demonstrate that the size of the Matrix profile subset ($subMP$), computed by the *computeSubMP* procedure, decreases in the same manner at each iteration (i.e., as we increase the length of the subsequences that the algorithm considers), regardless of the value of p.

It is important to note that irrespective of its size, $subMP$ *always* contains the smallest distances of the matrix profile, namely the distances of the motif pair. Having a larger $subMP$ does not represent an advantage w.r.t. motif discovery, but



motif length range: l_min 1024, l_max: 1124	Valid Distance profiles (green)	Distance profiles recomputed (red)	Non valid Distance profiles (black)
GAP	14%	0%	86%
EEG	21%	0%	79%
ECG	36%	0%	64%
EMG	3.54%	0.03%	96.43%
ASTRO	0.91%	0.17%	98.92%

Figure 5.15: Partial distance profile repartition (*valid*, *non-valid*, *recomputed*), in the motif discovery task on the five considered datasets.

rather an opportunity to view and analyze the subsequence pairs, whose distances are close to the motif.

5.7.3 Motif Sets

We now conduct an experiment to show the time performance of identifying the variable length motif sets. We use the default values of Table 5.2, varying K and

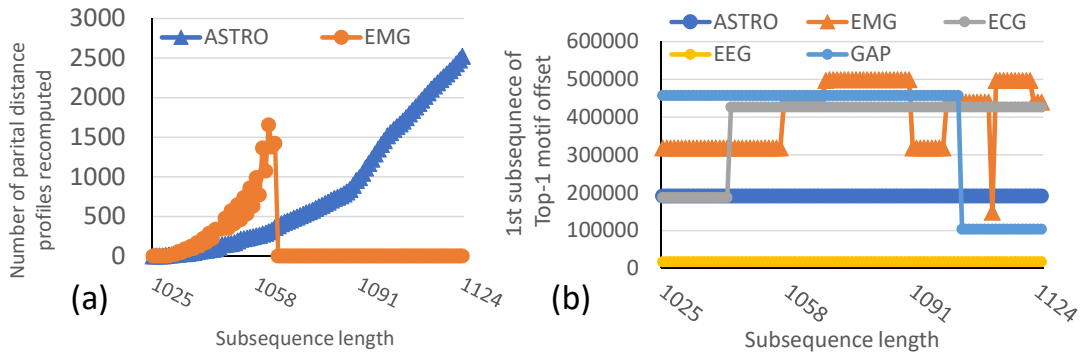


Figure 5.16: *(a)* Number of *recomputed* distance profiles in the EMG and ASTRO datasets. *(b)* Offset of the first subsequence for each motif in the EMG and ASTRO datasets.

the radius factor D for each dataset. In Figure 5.18 we report the results; we also show the time to compute $VALMP$ (the output of VALMOD). We note that once we build the pairs ranking of $VALMP$ ($heapBestKPairs$ in Algorithm 10), we can run the procedure that computes the motif sets (Algorithm 11). The results show that this operation is 3-6 orders of magnitude faster than the computation of $VALMP$. The advantage in time performance is pronounced for the ECG and EEG datasets, thanks to the pruning we perform with the partial distance profiles.

The fast performance of the proposed approach also allows for a fast exploratory analysis over the radius factor, which would otherwise (i.e., with previous approaches) be extremely time-consuming to set for each dataset.

5.7.4 Discord Discovery

In this last part, we conduct the experimental evaluation concerning discord discovery. In the following experiments, we use the same datasets as before.

We identify two state-of-the-art competitors to compare to our approach, the Motif And Discord (MAD) framework. The first one, DAD (Disk Aware Discord Discovery) [105], implements an algorithm suitable for enumerating the *fixed-length* m^{th} discords of a data series collection stored on a disk. We adapted this algorithm, as suggested by the authors, in order to extract discords from data series loaded in main memory. The second approach, GrammarViz [86], is the most recent technique, which discovers *Top-k* 1^{st} discords. It operates by means of grammar rules

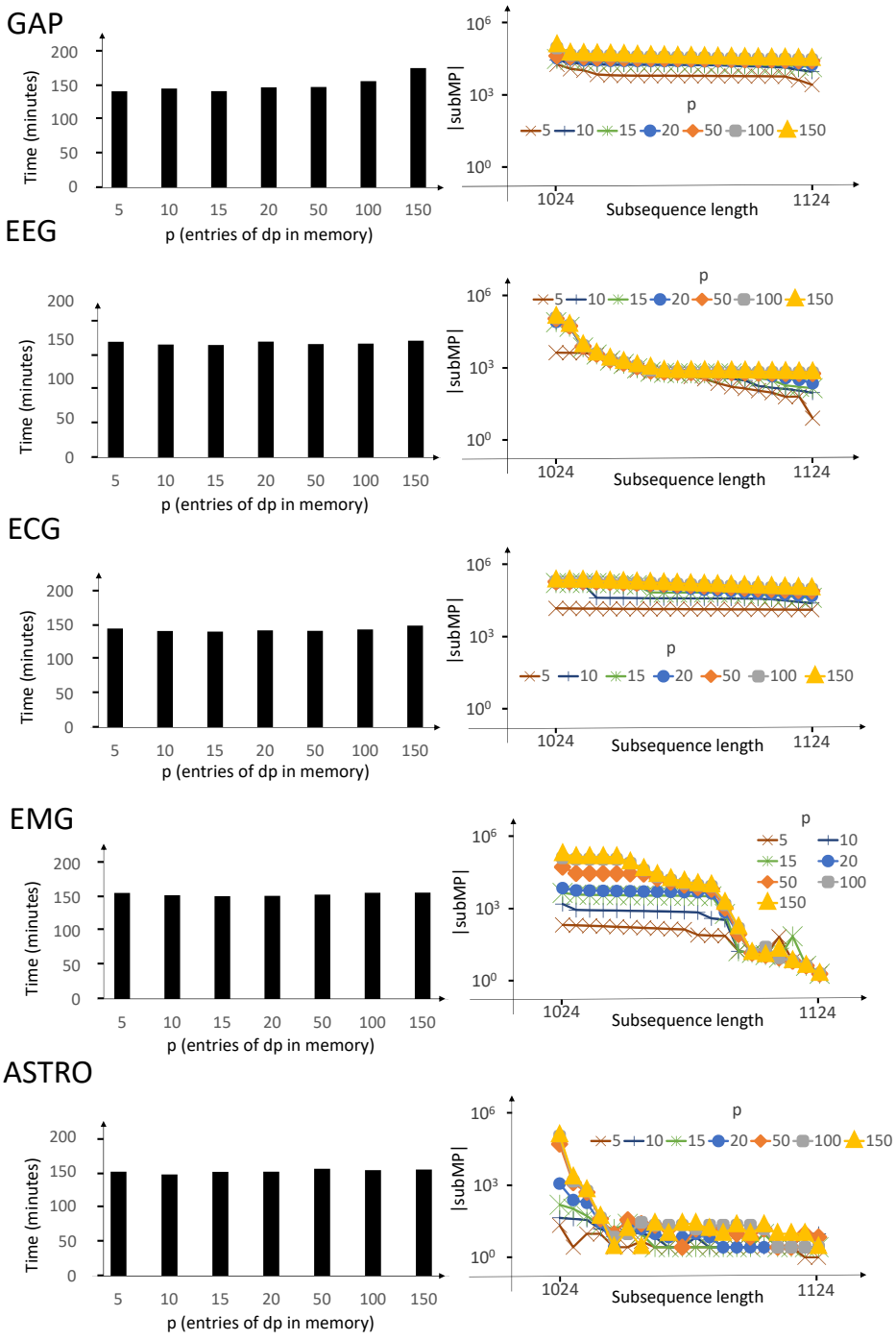


Figure 5.17: Scalability with increasing parameter p.

GAP		EEG		ECG		EMG		ASTRO	
VALMP time	9601 seconds	VALMP time	9608 seconds	VALMP time	9653 seconds	VALMP time	10294 seconds	VALMP time	9100 seconds

a)

K	Top K sets (seconds)	K	Top K sets (seconds)	K	Top K sets (seconds)	K	Top K sets (seconds)	K	Top K sets (seconds)
10	1.74	**10**	0.09	**10**	0.001	**10**	1.64	**10**	1.60
20	3.37	**20**	0.09	**20**	0.001	**20**	3.27	**20**	3.23
40	6.66	**40**	0.09	**40**	0.001	**40**	6.53	**40**	6.37
60	10	**60**	0.09	**60**	0.001	**60**	9.81	**60**	9.44
80	13.33	**80**	0.09	**80**	0.001	**80**	13.08	**80**	12.59

b)

D	Top K sets (seconds)	D	Top K sets (seconds)	D	Top K sets (seconds)	D	Top K sets (seconds)	D	Top K sets (seconds)
2	0.0015	**2**	0.001	**2**	0.001	**2**	6.52	**2**	4.45
3	0.0016	**3**	0.001	**3**	0.001	**3**	6.50	**3**	5.79
4	6.67	**4**	0.22	**4**	0.001	**4**	6.88	**4**	6.12
5	6.67	**5**	0.35	**5**	0.001	**5**	7.47	**5**	6.45
6	6.67	**6**	0.88	**6**	0.001	**6**	6.86	**6**	6.56

Figure 5.18: Time performance of variable length motif sets discovery. (a) Varying K (default D=4). (b) Varying radius factor D (default K=40).

compression, which further operate on a summarized data series representation, in order to find the rare segments of the data (discords) in a reduced search space. To the best of our knowledge, there exist no techniques capable of finding the *Top-k* m^{th} ranked variable-length discords as MAD, using a single execution of an algorithm.

M$^{\text{th}}$ Discord Discovery. In Figures 5.19(a)-(b), we present the performance comparison between MAD and DAD for finding the m^{th} discords, when we vary m, for all datasets. (All other parameters are set to their default values, as listed in Table 5.2.)

Since DAD discovers fixed-length m^{th} discords, we report its execution time *only* for the first length in the range, namely ℓ_{min}. We observe that MAD, which enumerates the m^{th} discords of *100* lengths ($\ell_{min} = 1024$, $\ell_{max} = 1124$) is still one order of magnitude faster than these DAD performance numbers, for all datasets, when m is larger or equal to *5*. Moreover, the performance trend of MAD remains stable over all datasets, whereas DAD has different execution times. We observe that the computational time of DAD depends on the subsequence length, since it computes Euclidean distances in their entirety (only applying early abandoning based on the best so far distance). How effective this early abandoning mechanism is, depends on the characteristics of the data. On the other hand, our algorithm computes all distances for the first subsequence length in constant time, and then

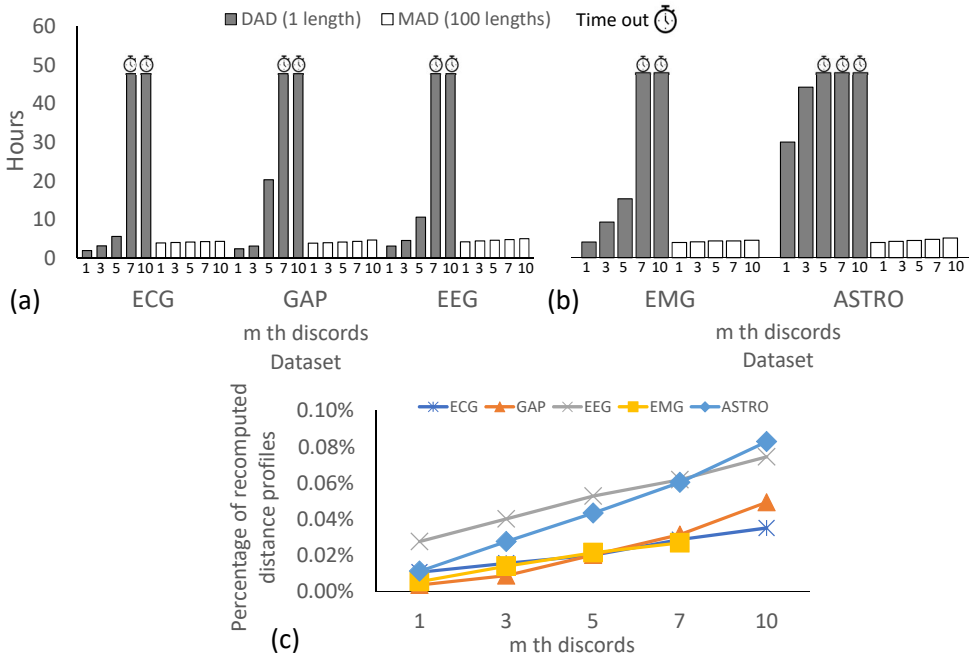


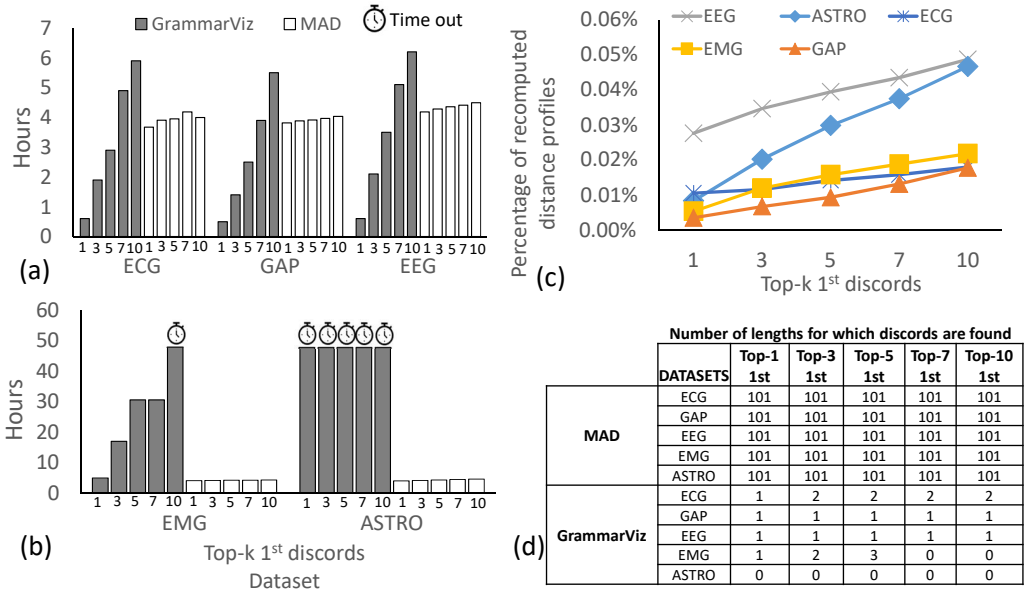
Figure 5.19: *(a),(b)* DAD (one length) and MAD (*100* lengths) *Top-k* m^{th} discords discovery time. (c) Percentage of *non-valid* partial distance profiles recomputed.

prunes entire distance computations for the larger lengths.

In Figure 5.19(c), we report the percentage of non-valid distance profiles that are recomputed, over the total number of distance profiles considered during the entire task of variable-length discord discovery. We note that the number of re-computations is limited to no more than *0.10%*, in the worst case. This demonstrates the high computation pruning rate achieved by our algorithm, justifying the considerable speed-up achieved.

***Top-k 1st* Discord Discovery.** In Figure 5.20, we depict the performance comparison between GrammarViz and MAD. Therefore, we consider *Top-k 1st* discords discovery, as previously introduced. (We maintain the same parameters setting in this experiment.)

First, we note that GrammarViz outperforms MAD in the first three datasets, for k smaller or equal to *5*, as depicted in Figure 5.20(a). Nevertheless, the experiment shows that MAD scales better over the number of discovered *Top-k 1st* discords, as its execution time increases only by a small constant factor. A different trend is observed for GrammarViz, whose performance significantly deteriorates as k



(d) Number of lengths for which discords are found

	DATASETS	Top-1 1st	Top-3 1st	Top-5 1st	Top-7 1st	Top-10 1st
MAD	ECG	101	101	101	101	101
	GAP	101	101	101	101	101
	EEG	101	101	101	101	101
	EMG	101	101	101	101	101
	ASTRO	101	101	101	101	101
GrammarViz	ECG	1	2	2	2	2
	GAP	1	1	1	1	1
	EEG	1	1	1	1	1
	EMG	1	2	3	0	0
	ASTRO	0	0	0	0	0

Figure 5.20: *(a),(b)* GrammarViz and MAD (*100* lengths) *Top-k 1st* discords discovery time. *(c)* Percentage of *non-valid* partial distance profiles recomputed.

increases from 1 to 6.

Moreover, this technique is highly sensitive to the dataset characteristics, as we observe in Figure 5.20(b), where the two noisy datasets, i.e., EMG and ASTRO, are considered. This is a direct consequence of the data summarization sensitivity to the data characteristics, which then influences the ability to prune distance computations.

In Figure 5.20(c), we report the percentage of non-valid distance profiles that MAD needed to recompute. In this case, too, this percentage is very low.

To conclude, since GrammarViz is a variable length approach that selects the most promising discord lengths according to the distribution of the data summarization (by picking the lengths of the series, whose discrete versions represent a rare occurrence), we report in Figure 5.20(d) the number of lengths, for which discords are found. We observe that our framework always enumerates and ranks discords of all lengths in the specified input range, based on the exact Euclidean distances of the subsequences. On the other hand, GrammarViz selects the most promising length based on the discrete version of the data, and only identifies the exact *Top-k 1st* discords for *3* (out of *100*) different lengths in the best case.

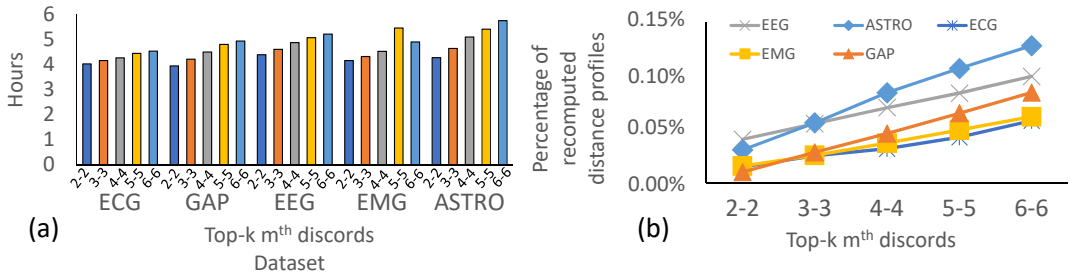


Figure 5.21: *(a)* MAD (*100* lengths) *Top-k* m^{th} discords discovery time on the five datasets. (b) Percentage of *non-valid* partial distance profiles recomputed.

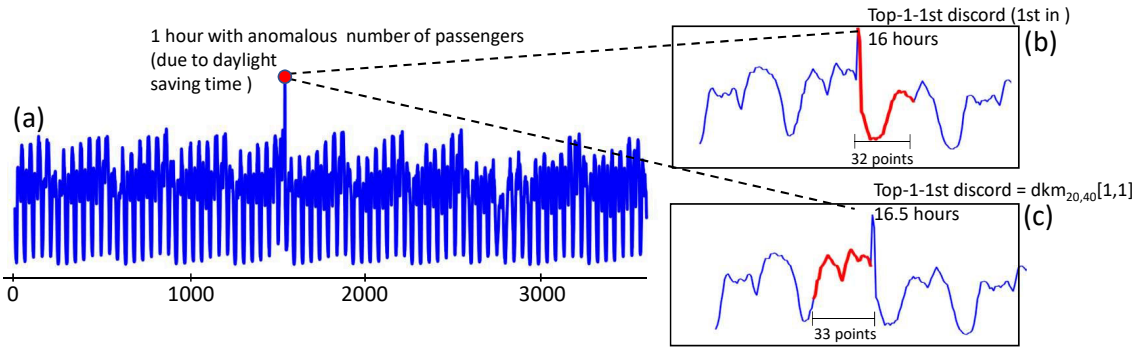


Figure 5.22: *(a)* Number of taxi passengers over 75 days in New York City. *(b)* $Top-1$ 1^{st} discord of length *32*. *(c)* $Top-1$ 1^{st} discord of length *33*.

***Top-k* m^{th} Discord Discovery.** Figure 5.21 depicts the execution time for the *Top-k* m^{th} discord discovery task, and the percentage of recomputed distance profiles for MAD, when varying k and m. We observe that the pruning power remains high: the percentage of distance profile re-computations averages around 0.05%.

Utility of Variable-Length Discord Discovery. We applied MAD on a real use case, a data series containing the average number of taxi passengers for each half hour over 75 days at the end of 2014 in New York City [80], depicted in Figure 5.22(a). We know that this dataset contains an anomaly that occurred during the daylight savings time end, which took place the 2^{nd} of November 2014 at 2am. At that time, the clock was set back at 1am. Since the recording was not adjusted, two samples (corresponding to a *1* hour recording) are summed up with the two subsequent ones.

We ran the variable-length discord discovery task using the length range $\ell_{min} = 20$ and $\ell_{max} = 48$, in order to cover subsequences that correspond to recordings

between 10− and 24 hours. Our algorithm correctly identifies the anomaly for subsequence length *32*, shown in Figure 5.22(b). Changing the window size does not allow the detection of the anomaly. For example, enlarging the window by *just* 1 point, the *Top-k* 1^{st} discord corresponds to a pattern *before* the abnormality (refer to Figure 5.22(c)).

These results showcase the importance of efficient variable-length discord discovery. It permits us to discover rare, or abnormal events with different durations, which can be easily missed in the fixed length discord discovery setting, where the analyst is constrained to examine a single length (or time permitting, a few fixed lengths).

5.7.5 Exploratory Analysis: Motif and Discord Length Selection

In this part, we present the results of an experiment we conducted to test the capability of MAD to suggest the most promising length/s for motifs and discords.

Given a data series, the user may have no clear idea about the motif/discord length. Therefore, we present use cases that examine the ability of MAD to perform a wide length-range search, providing the most promising results at the *correct* length.

We used MAD for finding motifs and discords in the length range: $\ell_{min} = 256$ and $\ell_{max} = 4096$. We conducted this experiment in the first $500K$ points of the datasets listed in Table 5.1. The considered motif/discord length range covers the user studies that have been presented so far in the literature (where knowledge of the exact length was always assumed).

Scalability. The MAD framework completed the motif/discord discovery task within *2 days* (on average), enumerating the motifs and the $Top-1$ discords of each length in the given range. Concerning the competitors, we estimated that STOMP, which is the state-of-the-art solution for fixed length motif/discord discovery would take 320 days for the same experiment (a little bit more than two hours for each of the lengths we tested). QUICK MOTIF, which has data dependent time performance, takes up to more than *6* days (projection) for all datasets but ECG (which completes in *38* hours). We note that the variable-length motif discovery competitor (MOEN) never terminates before *24* hours when searching motifs of 600 different lengths, while in this experiment, the length range is composed of 3841 different lengths. Considering discord discovery, we observed

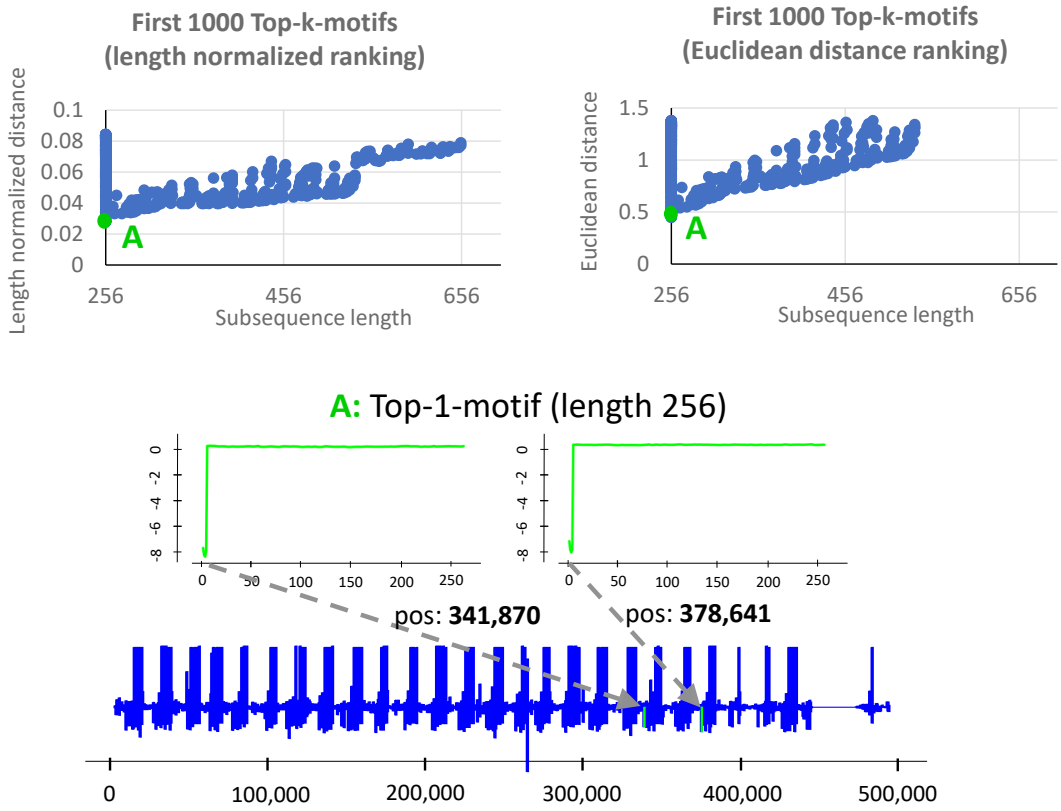


Figure 5.23: Top-1 motif (of length 256) in the EEG data set. The subsequences pairs composing this motif have the smallest distance in both the Euclidean distance and length normalized ranking.

that GrammarViz does not enumerate all the discords in the given length-range, since it selects the length according to the data summarizations. Thus, we are obliged to run this technique independently for each length, which would take at least 320 hours in the best case (projection based on results of Figure 5.20).

Select the most promising length in Motif Discovery. Once the search is completed, the MAD framework enumerates the motifs and discords ranking them in a second step, according to the proposed distance normalization strategy. In Figure 5.23, we show the results of motif discovery for the EEG dataset.

The objective of this experiment is to evaluate the proposed length-normalized correction strategy. In this regard, we compare the motifs sorted by using length-

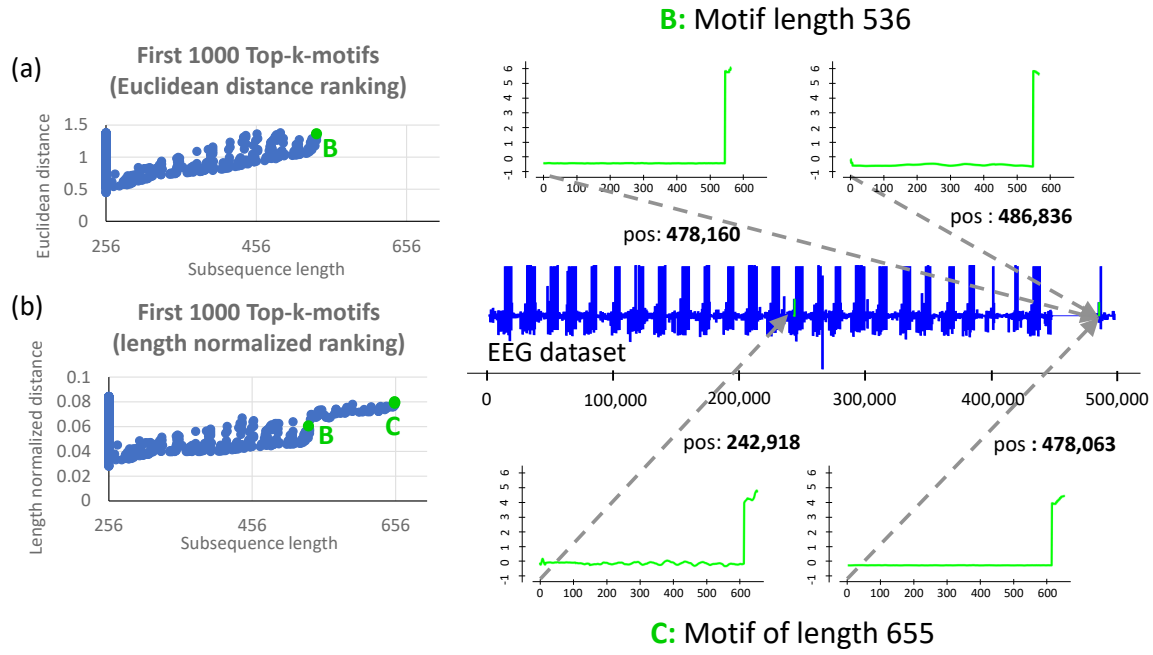


Figure 5.24: *(a)* Top-1000 motifs according the length normalized distance (top), and the Euclidean Distance (bottom). *(b)* Motif pair of the largest length (656) in the length normalized ranking (top) and motif pair of the largest length (536) in the Euclidean distance ranking (red/bottom).

normalization, and by Euclidean distances.

On the top part of Figure 5.23, we report the distance/length values of the $Top-1000$ motifs ranked by the length-normalized measure (left), which comprise a subset of the results we store in the VALMP structure (Algorithm 6). In the right part of the figure, we report the $Top-1000$ motifs ordered by their Euclidean distances.

We observe that the Top-1 motif, i.e., the subsequence pair with the smallest distance (marked by the letter A) is the same in both rankings. We report this motif in the bottom part of Figure 5.23, which is composed of two quasi-identical patterns in the EEG data series.

We now evaluate motifs of larger lengths in the same dataset, which may reveal other interesting and similar patterns at different resolutions (lengths). In Figure 5.24(a), we report again the distance/length values of the $Top-1000$ motifs ranked by their Euclidean distance, which reveal that the longest motif, marked

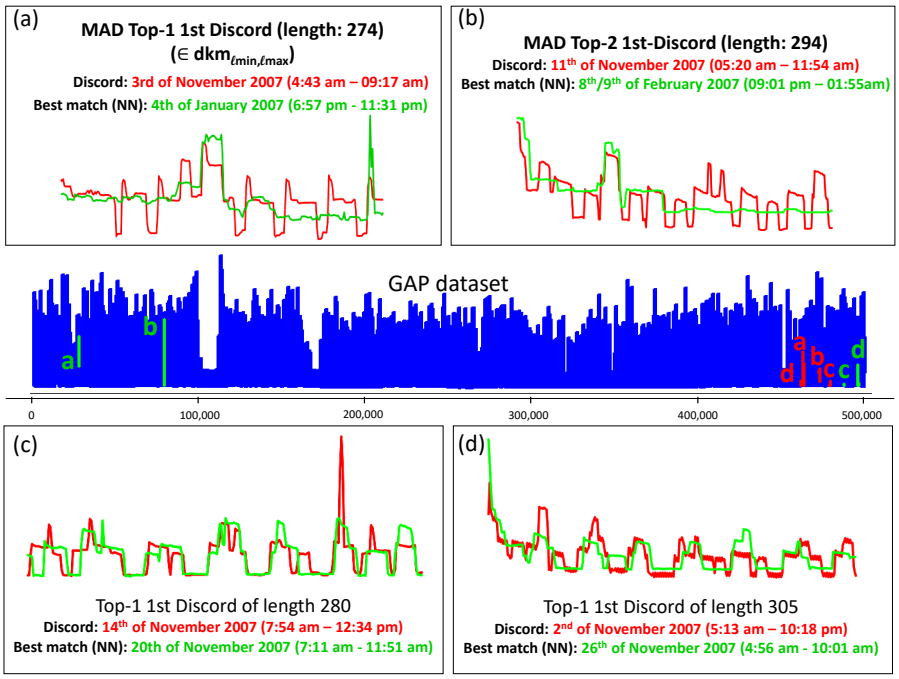


Figure 5.25: Four discords of different length in the GAP dataset. Each discord (red subsequence) is coupled with its nearest neighbor (green subsequence). *(a)* The discord, with the highest length-normalized distance to its nearest neighbor has length 274. *(b)* Discord with the second highest length-normalized distance. *(c)*,*(d)* discords with a smaller length-normalized distance to their nearest neighbor.

as B, has length 536. We observe that this subsequence pair substantially differs from the $Top-1$ motif of Figure 5.23.

Subsequently, in Figure 5.24(b), we report the longest motif (marked as C) of length 655 that we found in the $Top-1000$ motif ranking, based on length-normalized distances. We note that 6% of the length-normalized motifs are longer than those in the $Top-1000$ of the Euclidean ranking. The example of motif C, which is a longer version of B, shows that this pattern appears much earlier in the sequence than B. If we considered just the $Top-1000$ motifs ranked by their Euclidean distance, we would have missed this insight (motif C appears in the Euclidean distance ranking only in the $Top-4000$ motifs).

Select the most promising length in Discord Discovery. In this part, we show the results of discord discovery performed in the GAP dataset. We recall that

in this case, the discords ranking performed according to their length normalized distances aims to favor smaller discords, which have a high point to point distance.

In Figure 5.25, we report some of the discords we found in the length range $\ell_{min} = 256$ and $\ell_{max} = 4096$. The discord with the highest length-normalized distance, *best* Top-1 1^{st} discord, is the one depicted in the top-left part of the figure, and has length 274. We plot it in red (dark), whereas its nearest neighbor appears in green (light). We note that this discord drastically differs from its nearest neighbor: it represents a fluctuating cycle of global power activity, while its nearest neighbor exhibits the expected behavior of two major peaks, in the morning and around noon. In Figure 5.25(b) we report the Top-2 1^{st} discord in the length range 256-4096 identified by MAD, which corresponds to the subsequence in that length range with the second highest length-normalized distance to its nearest neighbor. Once again, we observe a high degree of dissimilarity between the pattern of this discord and its nearest neighbor. On the contrary, Figures 5.25(c) and (d) report the Top-1 1^{st} discords for two specific lengths (i.e., 280 and 305, respectively). These discords correspond to patterns that are not significantly different from their nearest neighbors. Therefore, they represent discords that are less interesting than the ones reported by MAD in Figures 5.25(a) and (b), which examines a large range of lengths.

This experiment demonstrates that MAD and the proposed discord ranking allows us to prioritize and select the correct discord length.

5.8 Conclusions

Motif and discord discovery are important problems in data series processing across several domains, and key operations necessary for several analysis tasks. Even though much effort has been dedicated to these problems, no solution had been proposed for discovering motifs and discords of different lengths.

In this chapter, we propose the first framework for variable-length motif and discord discovery. We describe a new distance normalization method, as well as a novel distance lower bounding technique, both of which are necessary for the solution to our problem. We experimentally evaluated our algorithm by using five real datasets from diverse domains. The results demonstrate the efficiency and scalability of our approach (up to 20x faster than the state of the art), as well as its usefulness.

Chapter 6

A Suite for Easy and Exact Detection of Variable Length Motifs in Data Series

Data series motif discovery represents one of the most useful primitives for data series mining, with applications to many domains, such as robotics, entomology, seismology, medicine, and climatology, and others. The state-of-the-art motif discovery tools still require the user to provide the motif length. Yet, in several cases, the choice of motif length is critical for their detection. Unfortunately, the obvious brute-force solution, which tests all lengths within a given range, is computationally untenable, and does not provide any support for ranking motifs at different resolutions (i.e., lengths).

In this chapter, we present a prototype system, which implements the scalable motif discovery algorithm (VALMOD) that efficiently finds all motifs in a given range of lengths, and outputs a length-invariant ranking of motifs. Furthermore, the prototype supports the analysis process by means of a newly proposed meta-data structure that helps the user to select the most promising pattern length. We illustrate in detail the steps of the proposed approach, showcasing how our algorithm and corresponding graphical interfaces enable users to efficiently identify the correct motifs.

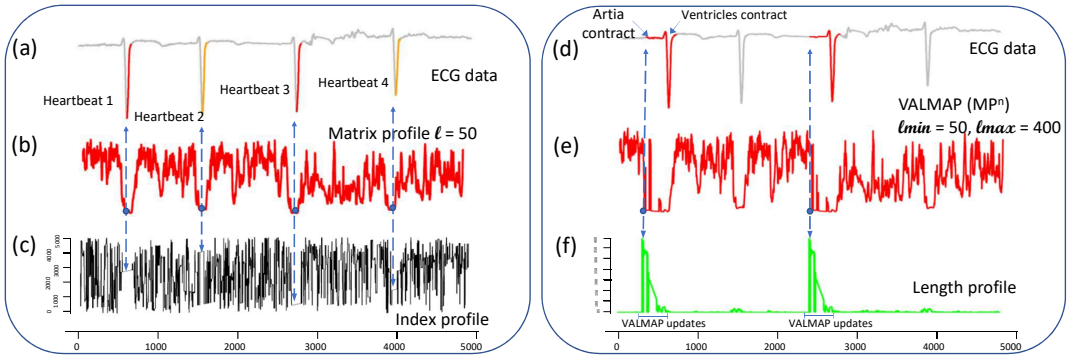


Figure 6.1: *Left*) *(a)* Snippet of ECG recording with highlighted motifs of length *50*, *(b)* Matrix profile computed with subsequence length *50*. *(c)* Index profile, reporting the offsets of the best match. *Right*) *(d)* Snippet of ECG recording with highlighted motifs of length *400*, *(e)* VALMAP MP^n, *(f)* VALMAP Length profile.

6.1 Motif discovery of different lengths.

Exact Motif discovery has merely become a single input parameter problem, namely the length of the patterns we want to mine. Unfortunately, this technique comes with an important lack. It does not provide an effective solution for trying several motif length in a range. If one has no cues about an effective fixed length, the simplest solution would be to run the algorithm over all lengths in the range and rank the various motifs discovered, picking eventually the patterns, which contain the desired insight. Clearly, this possibility is not optimal for at least two reasons; the scalability, since finding motif of one fixed length takes $O(n^2)$ time, and also because it does not provide an effective way to compare motifs of different lengths. In Chapter 5 we introduced *VALMOD* , the first approach for mining top-k motif pairs of variable length, which is up to orders of magnitude faster/more scalable than the alternatives that have been proposed in the literature.

Here, in order to show the superiority of variable-length motif discovery, we consider the following example. In Figure 6.1 (left) we depict a snippet of an Electrocardiogram (ECG) recording in (a), paired with its Matrix profile, computed with fixed subsequence length: $\ell = 50$ in (b). Note that each value in the Matrix profile corresponds to a point in the data, which is the representative starting point of a subsequence of length ℓ. Hence, given a data series D of length $|D|$, a Matrix profile records the $|D| - \ell + 1$ nearest neighbor distances, avoiding trivial matches. In Figure 6.1.(c) we plot the Index profile, which contains the offsets of the best matches.

Looking at the Matrix profile in this example, we note four deep valleys, which suggest the presence of very close matches, namely the motifs. Starting from the Matrix profile, it suffices to follow the dotted lines upwards, in order to detect the motifs, and downwards for finding the position of each subsequence best match. Despite the motifs (heartbeats) are easily detectable *to the naked eye*, since the snippet is relatively short, the highlighted motifs in Figure 6.1.(a) (red/orange subsequences), just report the second half of a ventricular contraction, giving thus a partial and unsatisfactory result.

6.2 VALMAP data structure.

In the previous chapter, we introduced a motif rank that weights the subsequences importance according to the ratio distance-length. Furthermore, we want to know, whether and how the motif pairs changes, helping the user to extract the desired insights at the correct length. To that extent, we introduce a new meta-data, called Variable Length Matrix Profile (*VALMAP*), maintaining the same logic and structure of the Matrix profile depicted in Figure 6.1 (top), with the difference that this new structure carries length normalized distances and it is coupled with a new vector called *Length profile*, which contains the lengths of the subsequences. More formally, given a data series D, and a range of subsequence lengths, whose extremes are denoted by ℓ_{min} and ℓ_{max}, we define *VALMAP* as a triple $\langle MP^n \in \mathbb{R}^{|D|-\ell_{min}+1}, IP \in \mathbb{N}^{|D|-\ell_{min}+1}, LP \in \mathbb{N}^{|D|-\ell_{min}+1} \rangle$, where MP^n is the Matrix profile containing length normalized distances, whereas IP and LP are the relative Index and Length Profile. If we consider just a fixed length, *VALMAP* will coincide with the length normalized version of the Matrix profile, with a flat Length profile. This is basically the structure that *VALMOD* builds, considering subsequences of length ℓ_{min}. In the second stage, we can update *VALMAP* using the top-k motif pairs, computed for each length until ℓ_{max}. We thus consider each $(D_{i,\ell_{min}+1}, D_{j,\ell_{min}+1}) \in$ top-k motif pairs, where i, j are the subsequences offsets, $\ell_{min}+1$ their lengths and $d^n_{i,j}$ their length normalized Euclidean distance. Note that in a motif pair the right subsequence is the one with the absolute shortest distance to the one at the left. Hence, *VALMAP*, $MP^n[i]$ is updated with $d^n_{i,j}$ if $d^n_{i,j} < MP^n[i]$, which was containing the distance between $D_{i,\ell_{min}}$ and its best match. If this update takes place, the Index and Length profile are respectively assigned with j, the offset of the new best match, and $\ell_{min}+1$ the new length. The update operation takes place for each top-k motif pair of any length between ℓ_{min} and ℓ_{max}. Once the algorithms ends, *VALMAP* contains a picture of the motif pairs showing, at which length the last update takes place. If a motif pair

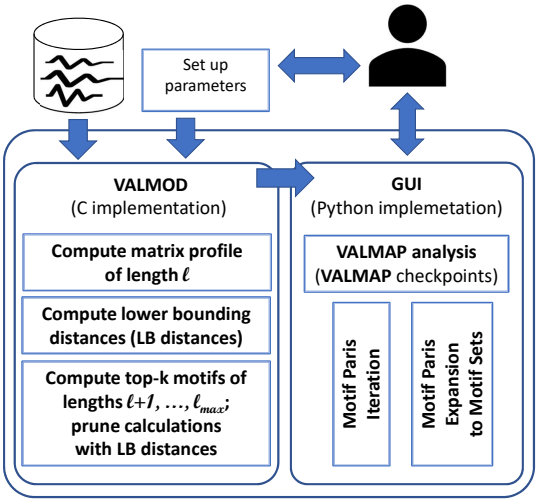


Figure 6.2: Architecture of VALMOD system.

is updated, this implies that a longer pattern represent a better match and thus it might reveal either a new event or the same event lasting longer.

Example of VALMAP Expressiveness. In order to show the expressiveness of *VALMAP*, we ran *VALMOD* on the ECG data snippet previously considered, showing the *VALMAP* structure in Figure 6.1 (right). We use the following input parameter: $\ell_{min} = 50$ and $\ell_{max} = 400$. We note that *VALMAP* reports the motif with the shortest length normalized distance of length 56, which is the same partial event detected by the Matrix profile in the fixed length case, at the top of the picture. If we look at the Length profile in Figure 6.1.(f), we observe that, at an earlier time than the discovered motifs pair, a sequence of contiguous updates took place, as we reported. The subsequences concerned have distances almost as short as the one of the best motifs in *VALMAP* , thus, remaining longer and possibly valid matches.

In Figure 6.1.(d) we depict and highlight the motif pair of length *400*. Immediately, we can note that, the subsequences in red, which compose this motif, are a better representation of a recurrent heartbeat. In fact, the two typical components (*Artia and Ventricles contract*) are correctly detected.

6.3 System Description

We now describe the architecture of the prototype we propose, depicted also in Figure 6.2. The input is represented by a data series of interest. As a starting point, the user has the possibility to inspect the data and also setting the desired parameter (lengths range $[\ell_{min},\ell_{max}]$). Afterwards, she can run the *VALMOD* algorithm, which is a part of the system back-end we implemented in C. Once terminated, *VALMOD* outputs the *VALMAP* meta-data. This latter is thus sent to the front-end, implemented in Python. Here, the user can interact with the system analyzing the showcased elements, such as:

- the checkpoints of the VALMAP, namely all the updates occurred from the length ℓmin till the desired length, selected with a dedicated slider.
- all the top-k motifs of variable length, which *VALMAP* reports.
- expand a selected motif pair to the relative Motif Set, containing all the similar subsequences of the pair in the data.

6.4 Prototype System

We now present the functionality of our prototype. We depict a screen-shot of our application in Figure 6.3, where the user imports (top of Figure 6.3) a dataset containing the recordings of the global active electric power in France for the period 2006-2008 (GAP) [53].

Traditional Motif discovery VS VALMOD. Once the user imports a dataset, she can opt to find motifs without having any knowledge of their lengths, just by inspecting the data themselves. Thereafter, the user can experience the VALMOD support in finding motif pairs that can be of variable length, understanding the quantity and quality of the insights that are not achievable with a simple raw data visual analysis. She can thus run our algorithm selecting the desired motif length range at the top of the interface (Figure 6.3). Our prototype disposes of two panels, which report the *VALMAP* structure and the Length Profile respectively, once the motif discovery terminates, as we depict in Figure 6.4. Here, we run motif discovery, with range $\ell_{min} = 256$ and $\ell_{min} = 1024$. We show the first *VALMAP* checkpoint, which permits to discover motifs of length 256; the $Top-1$ motif pair is reported in red.

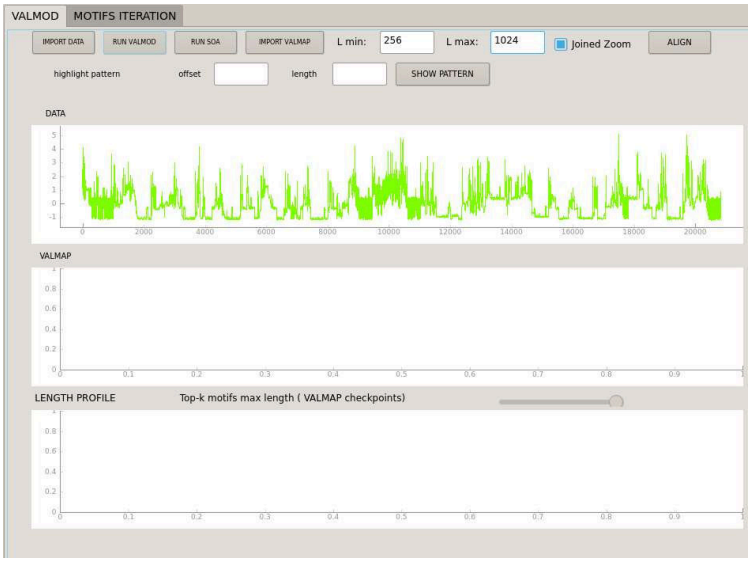


Figure 6.3: GUI interface of the prototype, which implements *VALMOD* .

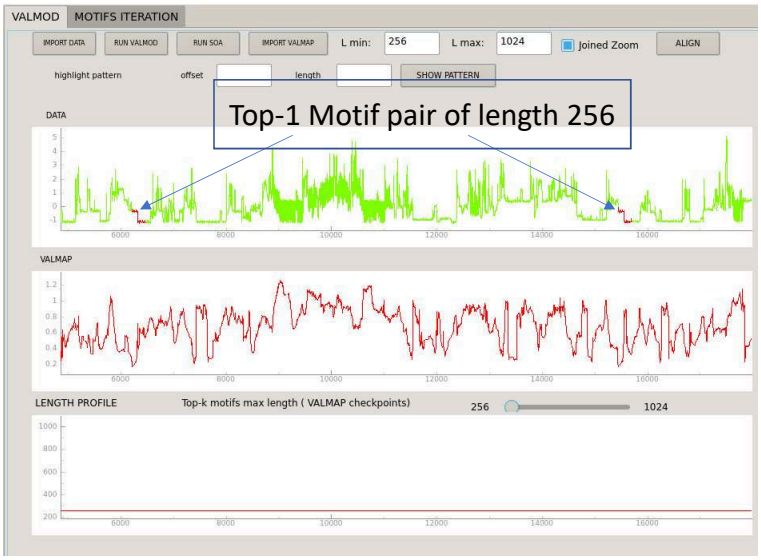


Figure 6.4: GUI interface of the prototype, *VALMAP* and Length profile structure panels.

Need for Variable Length Motifs. We tested our prototype, which implements *VALMOD* on different real datasets, including ECG and ASTRO (celestial objects data), GAP (global active power) as well as datasets coming from the domains of *Entomology* and *Seismology*. In these cases, but also in general the user can understand the importance of using variable length motif detection (with the support of *VALMAP*), in order to identify patterns of interesting behavior exhibiting them-

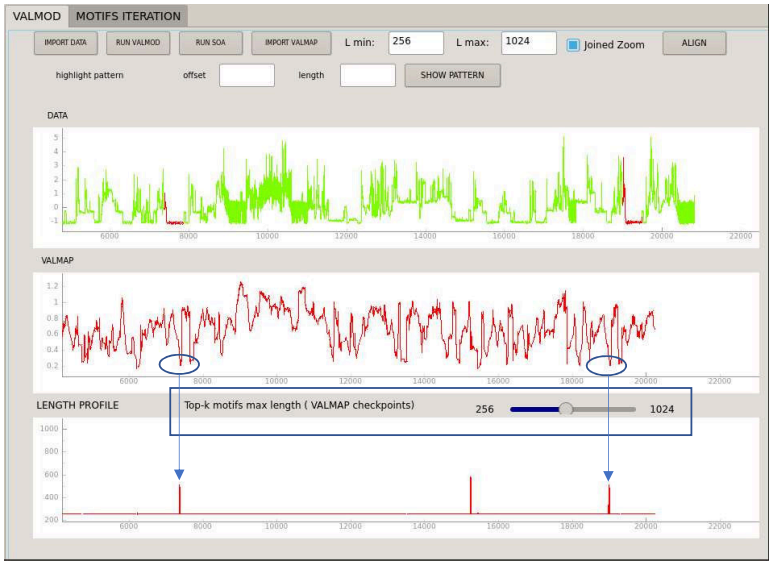


Figure 6.5: GUI interface of the prototype, mining motifs of variable length.

selves as sequences of different lengths. We can thus use the checkpoints slider, as depicted in Figure 6.5, to test the presence of motif pairs that are longer than ℓmin. We note that at length 490, the length profile contains several updates on different subsequences offset, which have (globally) small distances at the respective offsets in VALMAP (circles in Figure 6.5). This suggests the presence of a motif pair of length 490, which are reported in red. Beyond the *best* motif pairs (those with absolute smallest distances), the user can also iterate and visualize (in order) the rest of the motifs ranked by *VALMAP*, as we depict in Figure 6.6.

VALMOD VS Competitors. Using our prototype, the user has also the possibility to compare *VALMOD* to alternative approaches used for motif discovery. Specifically, she can note the *VALMOD* time performance improvement in variable length motif discovery. Beyond time performance, the user can also observe that the competitors do not provide an effective solution to compare motifs of different lengths. We face this limitation when running the state-of-the-art motif discovery algorithm. We depict the result in Figure 6.7, where the matrix profile is computed for one fixed length, namely 256 points, leading to discovery of motifs of only this length.

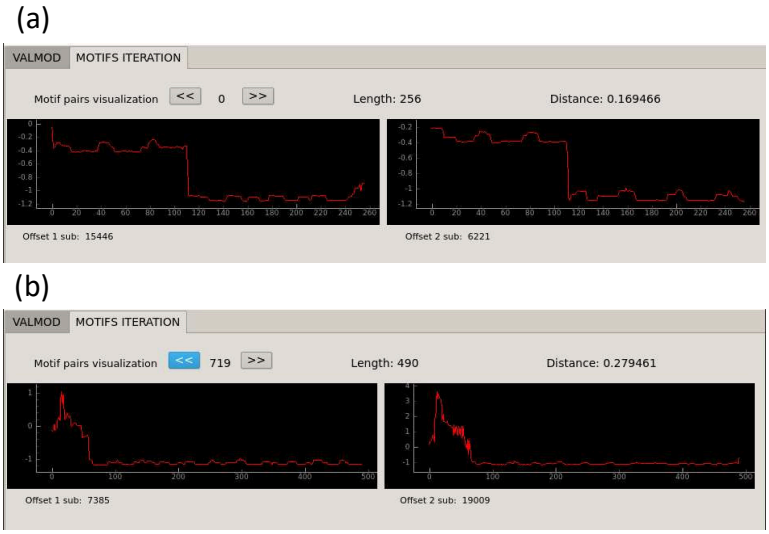


Figure 6.6: GUI interface of the prototype, $Top-k$ motifs iteration. *(a)* Top-1 motif of length 256. *(b)* Top-1 motif of length 490.

Figure 6.7: GUI interface of the prototype, state-of-the-art motif mining (Matrix profile).

6.5 Conclusions

In this work, we present a motif discovery prototype system based on the *VALMOD* algorithm, which efficiently finds data series motif of variable length. As opposed to the other approaches, our system provides a new meta data-series (*VALMAP*), which ranks motif pairs of variable length, using a new length normalized distance. Our solution provides enriched insights, which help to detect not only the correct resolution (length) of an interesting event, but also the occurrences of repeated patterns with different meanings, which are typical in numerous domains.

Chapter 7

Conclusions and Future Work

Similarity search is one of the fundamental operations for several data series analysis tasks. Even though much effort has been dedicated to the development of techniques that can speed up similarity search, all existing solutions are limited by the fact that they can only support fixed length results.

In this thesis we describe *exact* solutions for three problems that are based on similarity search, namely subsequence matching, motif and discord discovery. We extend the state-of-the-art by proposing algorithms that can operate with *variable-length* sequences, thus, removing this limitation that existed in all studies in the literature.

7.1 Subsequence Matching

To perform efficiently subsequence matching in large data collections, we propose ULISSE, the first index able to answer similarity search queries of variable-length, over both Z-normalized and non Z-normalized sequences, supporting the use of Euclidean and Dynamic Time Warping distances. The main ULISSE building block is new data series summarization (Envelope), which succinctly represents several contiguous overlapping sequences. We proposed a new search algorithm, which can answer $K-NN$ queries, and can be easily adapted to the ϵ-range search. We experimentally evaluated, our indexing and similarity search algorithms, on synthetic and real datasets, demonstrating that a compact and single structure

enables an efficient (in space and time cost) and scalable solution for subsequence matching.

7.1.1 Open Research Problems

This work has paved the road towards several extensions. First, we aim to improve the performance of the ULISSE indexing strategy for datasets that contain very long data series (where optimized serial scan techniques have an advantage). To that extent, we envision to further improve the space compression capability of ULISSE, finding a deterministic trade-off between the Envelopes size and space pruning capability.

Moreover, we want to explore the possibility to propose new similarity measures, which consider similar candidates according the subsequence features (possibly of variable length). A recent work [26] has shown that considering similar data series according to their subsequences can effectively improve clustering results, which are performed with Euclidean and DTW distances. We believe that ULISSE can inherently support this kind of measure, providing a scalable similarity search solution.

Finally, we also plan to study solutions built on top of ULISSE that can exploit multi-core and multi-socket architectures, which can significantly improve performance [75].

7.2 Motif and Discord Discovery

In the second part of the work, we proposed the first framework (MAD) for variable-length motif and discord discovery. We described a new distance normalization method, as well as a novel distance lower bounding technique, both of which are necessary for the solution to our problems. We experimentally evaluated our algorithms by using five real datasets from diverse domains. The results demonstrate the efficiency and scalability of our approach (up to 20x faster than the state of the art), as well as its usefulness.

7.2.1 Open Research Problems

In terms of future work, we would like to further improve the scalability of the MAD framework, as well as to extend the support to other data mining primitives. Our goal would be to support efficiently the computation of the complete matrix profile for each subsequence length in the input range. This would enable us to support more diverse applications, such as discovery of *shapelets* [108] in data series classification.

Moreover, we observe that motif discovery tools typically work by discovering motif pairs, which are then expanded to motif sets, just considering a fixed neighborhood space. Similarly, this is also true in the case of discord (anomalous patterns) discovery. We note that contrary to the case of motif pairs, there exists no measure for evaluating the quality of a pattern (motif or discord) set. In this direction, it seems promising to study new primitives that may describe and rank a set of interesting patterns. Following this idea, we would like to extend our framework by proposing a scalable solution for mining and ranking sets of patterns, irrespective of their cardinality.

In general, we showed that relaxing the fixed length (search) constraint in motif and discord discovery is a useful feature. In this sense, proposing an effective solution required us to tackle several non-trivial challenges. We are convinced that despite the hard nature of this problem, our results will allow us to push the limits (and our ambitions) even further. Therefore, our research directions point towards parameter-free solutions for data series analysis.

Bibliography

[1] Machine Learning in Time Series Databases (and Everything Is a Time Series !) Outline of Tutorial II. *Update*, pages 1–31.

[2] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient similarity search in sequence databases. In *Foundations of Data Organization and Algorithms, 4th International Conference, FODO'93, Chicago, Illinois, USA, October 13-15, 1993, Proceedings*, 1993.

[3] Ira Assent, Ralph Krieger, Farzad Afschari, and Thomas Seidl. The ts-tree: Efficient time series search and retrieval. In *EDBT*, 2008.

[4] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn J. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.*, 2017.

[5] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: An efficient and robust access method for points and rectangles. SIGMOD, 1990.

[6] Yingyi Bu, Tat wing Leung, Ada Wai chee Fu, Eamonn Keogh, Jian Pei, and Sam Meshkin. Wat: Finding top-k discords in time series database. In *SDM*, 2007.

[7] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn J. Keogh. isax 2.0: Indexing and mining one billion time series. In *ICDM 2010*.

[8] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn J. Keogh. isax 2.0: Indexing and mining one billion time series. In *ICDM 2010*, 2010.

[9] Edwin Cartlidge. Seven-year legal saga ends as italian official is cleared of manslaughter in earthquake trial. *Science*, Oct. 3, 2016.

[10] Kaushik Chakrabarti, Eamonn Keogh, and Sharad Mehrotra. Locally adaptive dimensionality reduction for indexing large time series databases. 2002.

[11] Qiuxia Chen, Lei Chen, Xiang Lian, Yunhao Liu, and Jeffrey Xu Yu. Indexable pla for similarity search. VLDB, 2007.

[12] Bill Yuan Chiu, Eamonn J. Keogh, and Stefano Lonardi. Probabilistic discovery of time series motifs. In *SIGKDD 2003*, pages 493–498, 2003.

[13] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, 1997.

[14] Michele Dallachiesa, Themis Palpanas, and Ihab F. Ilyas. Top-k nearest neighbor search in uncertain data series. *PVLDB*, 8(1):13–24, 2014.

[15] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art. *PVLDB*, 12(2), 2018.

[16] Alessandro Camerra et al. Beyond one billion time series: indexing and mining very large time series collections with isax2+. *KAIS*, 2014.

[17] Goldberger et al. Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals., 2000 June 13.

[18] Jiaye Wu et al. Kv-match: A subsequence matching approach supporting normalization and time warping. *ICDE*, 2019.

[19] S. Soldi et al. Long-term variability of agn at hard x-rays. *Astronomy & Astrophysics*, 2014.

[20] Thanawin Rakthanmanon et al. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*, 2012.

[21] Yan Zhu et al. Matrix profile II: exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In *ICDM 2016*.

[22] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, 1994.

[23] Ada Wai-Chee Fu, Oscar Tat-Wing Leung, Eamonn J. Keogh, and Jessica Lin. Finding time series discords based on haar transform. In *ADMA*, 2006.

[24] Yifeng Gao and Jessica Lin. Exploring variable-length time series motifs in one hundred million length scale. *Data Min. Knowl. Discov.*, 32(5):1200–1228, 2018.

[25] Yifeng Gao, Jessica Lin, and Huzefa Rangwala. Iterative grammar-based framework for discovering variable-length time series motifs. In *ICMLA, 2016*.

[26] Shaghayegh Gharghabi, Shima Imani, Anthony J. Bagnall, Amirali Darvishzadeh, and Eamonn J. Keogh. Matrix profile XII: mpdist: A novel time series distance measure to allow data mining in more challenging scenarios. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, 2018.

[27] C. Gisler, A. Ridi, D. Zufferey, O. A. Khaled, and J. Hennebert. Appliance consumption signature database and recognition test protocols. In *2013 WoSSPA)*, pages 336–341, 2013.

[28] Anna Gogolou, Theophanis Tsandilas, Themis Palpanas, and Anastasia Bezerianos. Progressive similarity search on time series data. In *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019.*

[29] Anna Gogolou, Theophanis Tsandilas, Themis Palpanas, and Anastasia Bezerianos. Comparing similarity perception in time series visualizations. *IEEE Trans. Vis. Comput. Graph.*, 25(1):523–533, 2019.

[30] Josif Grabocka, Nicolas Schilling, and Lars Schmidt-Thieme. Latent time-series motifs. *TKDD*, 11(1):6:1–6:20, 2016.

[31] Picard RW. Healey JA. Detecting stress during real-world driving tasks using physiological sensors. *IEEE Transactions in Intelligent Transportation Systems 6(2):156-166*, June 2016.

[32] Pablo Huijse, Pablo A. Estévez, Pavlos Protopapas, Jose C. Principe, and Pablo Zegers. Computational intelligence challenges and applications on large-scale astronomical time series databases. 2014.

[33] IRIS. Seismic Data Access 2016.

[34] F. Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1975.

[35] Srividya Kadiyala and Nematollaah Shiri. A compact multi-resolution index for variable length queries in time series databases. *KAIS*, 2008.

[36] Srividya Kadiyala and Nematollaah Shiri. A compact multi-resolution index for variable length queries in time series databases. *KAIS*, 2008.

[37] T. Kahveci and A. Singh. Variable length queries for time series data. In *ICDEF*, 2001.

[38] T. Kahveci and A. Singh. Variable length queries for time series data. In *Proceedings 17th International Conference on Data Engineering*, 2001.

[39] Kunio Kashino, Gavin Smith, and Hiroshi Murase. Time-series active search for quick retrieval of audio and video. In *ICASSP*, 1999.

[40] E. Keogh, J. Lin, and a. Fu. HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence. *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 226–233, 2005.

[41] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 3, 2000.

[42] Eamonn Keogh, Stefano Lonardi, Chotirat Ann Ratanamahatana, Li Wei, Sang-Hee Lee, and John Handley. Compression-based data mining of sequential data. *Data Mining and Knowledge Discovery*, 2007.

[43] Eamonn J. Keogh and Shruti Kasetty. On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Min. Knowl. Discov.*, 2003.

[44] Eamonn J. Keogh, Themis Palpanas, Victor B. Zordan, Dimitrios Gunopulos, and Marc Cardle. Indexing large human-motion databases. In *VLDB*, 2004.

[45] Eamonn J. Keogh, Themis Palpanas, Victor B. Zordan, Dimitrios Gunopulos, and Marc Cardle. Indexing large human-motion databases. In *VLDB*, 2004.

[46] Eamonn J. Keogh and Chotirat (Ann) Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 2005.

[47] Kin-Pong Chan and Ada Wai-Chee Fu. Efficient time series matching by wavelets. In *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, pages 126–133, March 1999.

[48] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. Coconut: A scalable bottom-up approach for building data series indexes. In *PVLDB*, 2018.

[49] Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. Coconut: A scalable bottom-up approach for building data series indexes. *PVLDB (11)6:677-690*, 2018.

[50] JB Kruskal and Mark Liberman. The symmetric time-warping problem: From continuous to discrete. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, 01 1983.

[51] Yuhong Li, Leong Hou U, Man Lung Yiu, and Zhiguo Gong. Quick-motif: An efficient and scalable framework for exact motif discovery, 2015.

[52] Yuhong Li, Leong Hou U, Man Lung Yiu, and Zhiguo Gong. Quick-motif: An efficient and scalable framework for exact motif discovery ICDE. 2015.

[53] M. Lichman. UCI machine learning repository, 2013.

[54] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 2007.

[55] Michele Linardi. Valmod support web page, 2017.

[56] Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn J. Keogh. VALMOD: A suite for easy and exact detection of variable length motifs in data series. In *SIGMOD Conference 2018*.

[57] Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn J. Keogh. VALMOD: A suite for easy and exact detection of variable length motifs in data series. In *SIGMOD Conference 2018*.

[58] Yubao Liu, Xiuwei Chen, and Fei Wang. Efficient Detection of Discords for Time Series Stream. *Advances in Data and Web Management*, pages 629–634, 2009.

[59] Wei Luo and Marcus Gallagher. Faster and parameter-free discord search in quasi-periodic time series. In Joshua Zhexue Huang, Longbing Cao, and Jaideep Srivastava, editors, *Advances in Knowledge Discovery and Data Mining*, 2011.

[60] Wei Luo, Marcus Gallagher, and Janet Wiles. Parameter-free search of time-series discord. *Journal of Computer Science and Technology*, 2013.

[61] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9), September 1993.

[62] David Minnen, Charles Lee Isbell Jr., Irfan A. Essa, and Thad Starner. Discovering multivariate motifs using subsequence density estimation and greedy mixture learning. In *AAAI Conference on Artificial Intelligence, 2007*.

[63] Katsiaryna Mirylenka, Michele Dallachiesa, and Themis Palpanas. Data series similarity using correlation-aware measures. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*, 2017.

[64] Y. Mohammad and T. Nishida. Unsupervised discovery of basic human actions from activity recording datasets. In *2012 IEEE/SICE International Symposium on System Integration (SII)*, 2012.

[65] Yasser F. O. Mohammad and Toyoaki Nishida. Exact discovery of length-range motifs. In *Intelligent Information and Database Systems - 6th Asian Conference, ACIIDS 2014*.

[66] Y Morinaka, Masatoshi Yoshikawa, Toshiyuki Amagasa, and Shunsuke Uemura. The l - index: An indexing structure for efficient subsequence matching in time sequence databases. 01 2001.

[67] Abdullah Mueen and Nikan Chavoshi. Enumeration of time series motifs of all lengths. *Knowl. Inf. Syst.*, 2015.

[68] Abdullah Mueen, Hossein Hamooni, and Trilce Estrada. Time series join on subsequence correlation. In *ICDM 2014*, 2014.

[69] Abdullah Mueen, Eamonn J. Keogh, Qiang Zhu, Sydney Cash, and M. Brandon Westover. Exact discovery of time series motifs. In *SDM 2009*.

[70] Moss C. B. Neupane, D. and A. H. 2016. van Bruggen. Estimating citrus production loss due to citrus huanglongbing in florida. *Annual Meeting, Southern Agricultural Economics Association, San Antonio, TX.*, 2016.

[71] Alleon Guillaume. Head of Operational Intelligence Department Airbus. Personal communication., 2017.

[72] Themis Palpanas. Data series management: The road to big sequence analytics. *SIGMOD Rec.*, 2015.

[73] Themis Palpanas. Big sequence management: A glimpse of the past, the present, and the future. In *SOFSEM*, 2016.

[74] Themis Palpanas. The parallel and distributed future of data series mining. In *High Performance Computing & Simulation (HPCS)*, 2017.

[75] Botao Peng, Panagiota Fatourou, and Themis Palpanas. Paris: The next destination for fast data series indexing and query answering. In *IEEE Big Data*, 2018.

[76] Ivan Popivanov and Renée J. Miller. Similarity search over time-series data using wavelets. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, 2002.

[77] Davood Rafiei and Alberto Mendelzon. Efficient retrieval of similar time sequences using dft. In *ICDE*, 1998.

[78] Thanawin Rakthanmanon, Bilson J. L. Campana, Abdullah Mueen, Gustavo E. A. P. A. Batista, M. Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *SIGKDD*, 2012.

[79] Usman Raza, Alessandro Camerra, Amy L. Murphy, Themis Palpanas, and Gian Pietro Picco. Practical data prediction for real-world wireless sensor networks. *IEEE Trans. Knowl. Data Eng.*, 2015.

[80] Kexin Rong and Peter Bailis. ASAP: prioritizing attention via time series smoothing. *PVLDB*, 10(11):1358–1369, 2017.

[81] A.C. Rosa, L. Parrino, and M.G. Terzano. Automatic detection of cyclic alternating pattern (cap) sequences in sleep: preliminary results. *Clinical Neurophysiology*, 1999.

[82] D. Roverso. Multivariate temporal classification by windowed wavelet decomposition and recurrent networks. In *ANS International Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface*, 2000.

[83] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1978.

[84] Suchi Saria, Andrew Duchi, and Daphne Koller. Discovering deformable motifs in continuous time series data. In *IJCAI 2011*.

[85] Patrick Schäfer and Mikael Högqvist. Sfa: A symbolic fourier approximation and index for similarity search in high dimensional datasets. In *EDBT*, 2012.

[86] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, Crystal Chen, and Susan Frankenstein. Time series anomaly discovery with grammar-based compression. In *EDBT*, 2015.

[87] Dennis Shasha. Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Eng. Bull.*, 1999.

[88] Jin Shieh and Eamonn Keogh. iSAX: Indexing and Mining Terabyte Sized Time Series. In *SIGKDD*, pages 623–631, 2008.

[89] Jin Shieh and Eamonn Keogh. iSAX: disk-aware mining and indexing of massive time series datasets. *DMKD*, 19(1):24–57, 2009.

[90] Jin Shieh and Eamonn J. Keogh. *i*sax: indexing and mining terabyte sized time series. In *KDD*, 2008.

[91] Saurabh Sinha. Discriminative motifs. In *Proceedings of the Sixth Annual International Conference on Computational Biology, RECOMB 2002*, pages 291–298, 2002.

[92] Zeeshan Syed, Collin M. Stultz, Manolis Kellis, Piotr Indyk, and John V. Guttag. Motif discovery in physiological datasets: A methodology for inferring predictive elements. *TKDD*, 4(1):2:1–2:23, 2010.

[93] et al. Terzano. Atlas, rules, and recording techniques for the scoring of cyclic alternating pattern (cap) in human sleep. Sleep Med 2001 Nov, 2(6):537-553.

[94] `www.mi.parisdescartes.fr/~mlinardi/ULISSE.html`.

[95] J. Wang, A. Balasubramanian, L. Mojica de la Vega, J. Green, A. Samal, and B. Prabhakaran. Word recognition from continuous articulatory movement time-series data using symbolic representations. In *Workshop on Speech and Language Processing for Assistive Technologies. (SLPAT)*.

[96] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Min. Knowl. Discov.*, 2013.

[97] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *PVLDB*, 2013.

[98] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *PVLDB*, 6(10):793–804, 2013.

[99] Yang Wang, Peng Wang, Jian Pei, Wei Wang, and Sheng Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *Proceedings of the VLDB Endowment*, 6:793–804, 08 2013.

[100] CW Whitney, DJ Gottlieb, S Redline, RG Norman, RR Dodge, E Shahar, S Surovec, and FJ Nieto. Reliability of scoring respiratory disturbance indices and sleep staging. *Sleep*, November 1998.

[101] Daniel Wu, Divyakant Agrawal, Amr El Abbadi, Ambuj K. Singh, and Terence R. Smith. Efficient retrieval for browsing large image databases. In *CIKM*, 1996.

[102] Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. Dpisax: Massively distributed partitioned isax. In *ICDM*, 2017.

[103] Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. Dpisax: Massively distributed partitioned isax. In *ICDM*, pages 1135–1140, 2017.

[104] Dragomir Yankov, Eamonn J. Keogh, Jose Medina, Bill Yuan-chi Chiu, and Victor B. Zordan. Detecting time series motifs under uniform scaling. In *ACM*.

[105] Dragomir Yankov, Eamonn J. Keogh, and Umaa Rebbapragada. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. In *IEEE (ICDM 2007)*, 2007.

[106] Dragomir Yankov, Eamonn J. Keogh, and Umaa Rebbapragada. Disk aware discord discovery: finding unusual time series in terabyte sized datasets. *Knowl. Inf. Syst.*, 2008.

[107] Dragomir Yankov, Eamonn J. Keogh, and Umaa Rebbapragada. Disk aware discord discovery: finding unusual time series in terabyte sized datasets. *Knowl. Inf. Syst.*, 2008.

[108] Lexiang Ye and Eamonn J. Keogh. Time series shapelets: a new primitive for data mining. In *KDD*, 2009.

[109] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn J. Keogh. Matrix profile I: all pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In *IEEE 16th International Conference on Data Mining, ICDM 2016*.

[110] Sorrachai Yingchareonthawornchai, Haemwaan Sivaraks, Thanawin Rakthanmanon, and Chotirat Ann Ratanamahatana. Efficient proper length time series motif discovery. In *2013 IEEE ICDM*.

[111] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. T-store: Tunable storage for large sequential data. In *North East Database Day (NEDB), Boston, MA, USA, January 2019*.

[112] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. Indexing for interactive exploration of big data series. In *SIGMOD*, 2014.

[113] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. RINSE: interactive data series exploration with ADS+. *PVLDB*, 2015.

[114] Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. ADS: the adaptive data series index. *VLDB J. 25(6): 843-866*, 2016.

[115] Kostas Zoumpatianos, Yin Lou, Themis Palpanas, and Johannes Gehrke. Query workloads for data series indexes. In *KDD*, 2015.

[116] Kostas Zoumpatianos and Themis Palpanas. Data series management: Fulfilling the need for big sequence analytics. In *ICDE*, 2018.