The Parallel and Distributed Future of Data Series Mining

Themis Palpanas
Paris Descartes University
themis@mi.parisdescartes.fr

***Abstract*—There is an increasingly pressing need, by several applications in diverse domains, for developing techniques able to index and mine very large collections of sequences, or data series. Examples of such applications come from biology, astronomy, entomology, the web, and other domains. It is not unusual for these applications to involve numbers of data series in the order of hundreds of millions to billions, which are often times not analyzed in their full detail due to their sheer size. In this work, we describe past efforts in designing techniques for indexing and mining truly massive collections of data series, based on indexing techniques for fast similarity search, an operation that lies at the core of many mining algorithms. We show that there are two bottlenecks in mining such massive datasets, namely, the time taken to build the index, and the time required to answer exactly similarity queries. In response to these challenges, we discuss novel techniques that adaptively create data series indexes, allowing users to correctly answer queries before the indexing task is finished. We also show how our methods allow mining on datasets that would otherwise be completely untenable, including the first published experiments using one billion data series. Moreover, we present our vision for the future in big sequence management and mining research: we argue that more efforts should concentrate on parallel (including modern hardware optimization opportunities) and distributed solutions, which have until now been largely unexploited.**

I. Introduction

[Motivation.] Data series have gathered the attention of the data management community for almost two decades [40], [9], [23]. Data series are one of the most common types of data, and are present in virtually every scientific and social domain: they appear as audio sequences [16], shape and image data [45], financial [39], telecommunications [32], [25], environmental monitoring [37] and scientific data [15], [1], and they have many diverse applications, such as in health care, astronomy, biology, economics, and others.

Recent advances in sensing, networking, data processing and storage technologies have significantly eased the process of generating and collecting tremendous amounts of data series at extremely high rates and volumes. It is not unusual for applications to involve numbers of sequences in the order of hundreds of millions to billions [1], [2].

[Data Series.] A *data series*, or *data sequence*, is an ordered sequence of data points[1]. Formally, a data series $T = (p_1, ...p_n)$ is defined as a sequence of points $p_i = (v_i, t_i)$, where each point is associated with a value v_i and a time t_i in which this recording was made, and n is the size (or length) of the series. If the dimension that imposes the ordering of the sequence 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, position in genome sequences, etc.).

A key observation is that analysts need to process and analyze a sequence (or subsequence) of values as a single object, rather than the individual points independently, which is what makes the management and analysis of data sequences a hard problem. Note that even though a sequence can be regarded as a point in n-dimensional space, traditional multi-dimensional approaches fail in this case, mainly due to the combination of the following two reasons: (a) the length (or dimensionality) is typically very high, i.e., in the order of several hundreds to several thousands, and (b) dimensions are strictly ordered (imposed by the sequence itself) and neighboring values are correlated.

[Need for Data Series Indexing.] There are two main types of data series queries that analysts need to perform: (a) simple Selection-Projection-Transformation (SPT) queries, and (b) more complex Data-Mining (DM) queries. Simple SPT queries are those that select sequences and project points based on thresholds, point positions, or specific sequence properties (e.g., above, first 10 points, peaks), or queries that transform sequences using mathematical formulas (e.g., average). DM queries on the other hand, treat an entire sequence as a single object, and are therefore much more complex to process. Examples under this category are: queries by content (range and similarity queries, nearest neighbors), clustering, classification, outlier patterns, frequent sub-sequences, and others. These queries cannot be supported by current data management systems, since they require specialized data structures, algorithms and storage methods in order to be performed efficiently.

In this context, the nearest neighbor, or similarity search operation is of paramount importance, since it forms the basis of virtually every DM query. However, similarity search queries across a large collection of data series are challenging, because data series collections grow very large in practice [10], [35], and existing data management solutions [6], [41], [42] cannot efficiently support them. Thus, methods for answering similarity search queries rely on two main techniques: data summarization and indexing. Data series summarization is used to reduce the dimensionality of the data series [18], [34], [20], [4], [17], [11], [22], and then indexes are built on top of these summarizations [34], [40], [5], [38], [44].

[Outlook.] We argue that we are now getting close to the limit of the performance that a single computer node (without use of the modern hardware optimization opportunities) can deliver, though the requirements of applications for managing and mining data series collections continue to increase along with the size of these collections. Therefore, it is now necessary to turn our efforts towards developing parallel and distributed algorithms, in order to take advantage

[1]For the rest of this paper, we are going to use the terms *data series* and *sequence* interchangeably.

of the scalability opportunities that these technologies have to offer [27], [28], [29]. In particular, these new algorithms should be able to exploit the parallelization opportunities of modern hardware, namely, Single Instruction Multiple Data (SIMD) operations, as well as multi-core and multi-threaded functionality. The Graphics Processing Unit (GPU) cards provide an additional opportunity for massive parallelization of certain data series operations. Moreover, these algorithms should also be able to scale out, using large clusters of machines in order to distribute the computation effort. The algorithms and techniques we propose here will make it feasible for analysts to tap in the goldmine of the massive and ever-growing data series collections they (already) have.

II. The State of the Art

In this section, we briefly describe and comment on the some of the most prominent efforts in the areas of managing and indexing data series collections[2].

[Using Existing Data Management Systems.] Existing approaches based on DBMSs [6], Column Stores [41], or Array Databases [42] do not provide a viable solution, since they have not been designed for managing and processing sequence data, and do not treat sequences as first class citizens. Note that neither the relational model nor the array model can adequately capture the characteristics of sequences. In the case of relational data, there are various options available for translating sequences into relations and each one of them has significant limitations. On the other hand, in Array Databases we lack the expressive power to define collections of sequences, and are restricted to defining large multi-dimensional matrices that encode both sequence and meta-data on an equal basis, which hinders efficiency.

These systems do not offer a suitable declarative query language, storage model, auxiliary data structures (such as indexes), and optimization mechanism that can support a variety of sequence query workloads in an efficient manner. Therefore, any solution built on top of them will suffer in terms of expressive power, usability, and performance.

[Scaling Up.] Even though recent studies have shown that in certain cases sequential scans can be performed very efficiently [35], [26], such techniques do not bring benefit to the general case of querying a mixed database of several data series (rather than a single, long series). Therefore, indexing is required in order to efficiently support data exploration tasks, which involve ad-hoc queries, i.e., the query workload is not known in advance. Figure 1 depicts a data series index, and how this can help prune the search space during query answering. A large set of indexing methods have been proposed for the different data series summarization methods, including traditional multidimensional [14], [34], [7], [19] and specialized [40], [5], [17], [38], [44] indexes.

Indexing can significantly reduce the time to answer DM queries. Yet, as the data series collections grow in size, the operation of indexing these collections can itself become the bottleneck in the entire process [9], [10], [47]. As an answer

[2]We do not discuss here problems related to data mining and analysis. Nevertheless, we argue that in most cases, the correct data management techniques can lead to significant time efficiency benefits for the mining and analysis algorithms.

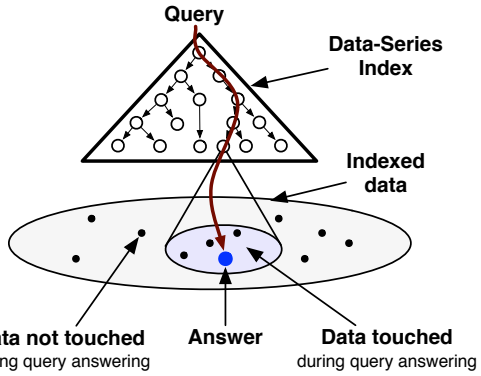


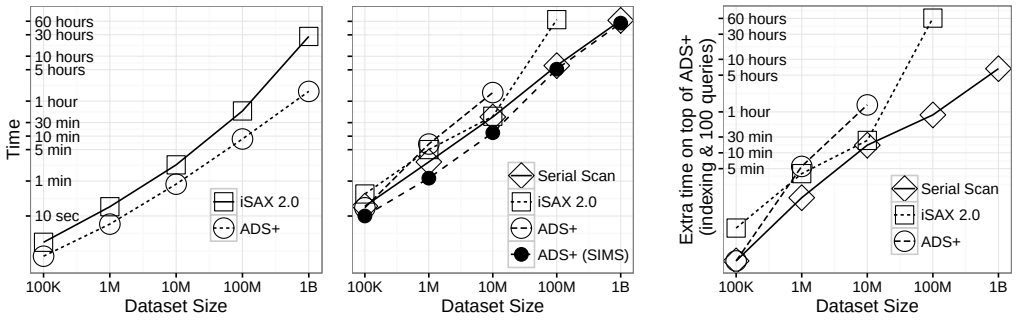
Fig. 1. An index structure built above a set of data series, pruning the search space for a query [50].

to this problem, iSAX2.0 [9] and iSAX2+ [10] were proposed, which are the first data series indexes that inherently support bulk loading, and thus aimed to minimize the index building time. More recently, the ADS+ index [47], [48], [49] was developed, which is the first data series index than can start answering queries correctly before the entire index has been built, by adaptively building and growing only the parts of the index that are needed for answering the queries. These techniques considerably shrink the data-to-query gap, allowing users to start answering queries much faster than any previous approach.

Nevertheless, many interesting problems are still open. For example, how we can efficiently support exact queries with ADS+, how we can reduce the large variance in exact query answering times, and how the iSAX2+ and ADS+ indexes are best parallelized. In Figure 2, we report the time performance of four existing exact similarity search algorithms: (i) Serial Scan, a baseline approach, which performs an optimized full sequential scan of the raw data file; (ii) iSAX 2.0, the exact search algorithm of the iSAX 2.0 index [9]; (iii) ADS+, the exact search algorithm of iSAX 2.0 running on top of the ADS+ index [47]; and (iv) ADS+ (SIMS), the SIMS exact search algorithm of ADS+, which uses a combination of index search and skip sequential scan [49]. In Figure 2(a), we plot the indexing time for iSAX 2.0 and ADS+ (Serial Scan has no initialization cost). ADS+ outperforms iSAX 2.0 by more than an order of magnitude in terms of data-to-query time. In Figure 2(b), we plot the query answering time for all algorithms, and in Figure 2(c), we plot the amount of additional time that all methods need in order to (index the dataset and) answer all the queries in the workload, when compared to ADS+ (SIMS). We observe that ADS+ (SIMS) is the fastest method across the board, and its benefit increases with the dataset size. Nevertheless, this experiment also shows that answering 100 exact similarity search queries is already almost one order of magnitude more expensive than building the index for the same dataset.

Evidently, it is necessary to significantly improve the performance of the state of the art algorithms, in order to address the increasing scalability needs of modern applications.

[Scaling Out.] During the last years there has been a lot of research on MapReduce systems, where various methods have been proposed to support the indexing of large multidi-



(a) Indexing times. (b) Querying times. (c) Extra time on top of ADS+ (with SIMS) for all other methods.

Fig. 2. Indexing up to 1 billion data series (random walk datasets) and issuing 100 exact queries [49].

mensional data [21], [24], where an index is distributed among several compute nodes. Nevertheless, up to this point work on sequential data query processing using MapReduce has mainly concentrated on efficiently performing parallel scans of the complete dataset [3].

Gorilla is a recent effort on building a distributed, in-memory sequence database, coupled with a long-term storage solution using HBase [33]. This system has several desirable properties: dedicated storage manager, compression support, efficient update mechanism, high availability, and very good scalability characteristics. The focus of the system though, is on answering simple SPT queries, and extending it to cover DM queries as well, is an open problem.

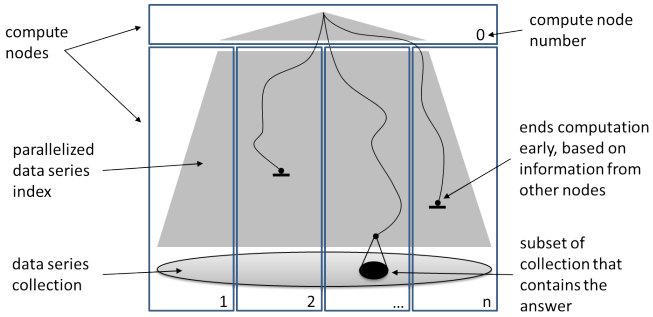


Fig. 3. Illustration of a distributed index solution using $n+1$ compute nodes.

III. Research Directions

In the following, we present open research problems related to parallel and distributed data series management and mining, along with the corresponding challenges.

A large collection of access methods has been proposed in the literature, able to evaluate different queries under various settings, including both indexes and scan-based methods. Methods that rely on fast scans of the data [17], [35] should be explored in more detail, especially, given the data management trend on large-scale parallelization, the usage of compression, multi-cores, SIMD architectures and the exploitation of available GPUs [36]. These directions can provide viable alternatives to the indexes discussed above, and in several situations can be the access method of choice. Data are compressed and indexed in a very light weight way in order to minimize the cost of transferring them to the CPU, and queries are able to operate directly on compressed data, thus removing the burden of compressing and de-compressing while answering a query. Performing efficient serial scans in compressed data sequences is still an open research problem. Apart from the need of developing such techniques, current Multi-core CPUs with their support for SIMD instructions, as well as new GPU processors, will allow for several different execution strategies that will vary given the features of each different platform. Some studies have already started to explore this space [43], [46], but there is a lot more yet to be done.

Up to this point, work on sequential data query processing using massively distributed platforms (such as MapReduce and Spark) has mainly concentrated on efficiently performing parallel scans of the complete dataset, while all indexing-related studies only consider read-only operations. Even though various approaches have been proposed for speeding up iterative algorithms, none of the proposed models is a suitable match for the algorithms and techniques we need, where timely communications among workers play a crucial role in reducing the amount of *total work* done. Figure 3 depicts this situation, where a data series index (such as iSAX 2.0) is split among several compute nodes. The challenge is that some operations, e.g., exact similarity search, require that different compute nodes coordinate their processing in order to reduce the total number of executed instructions. Therefore, there is need for more work in this area, taking into consideration new paradigms as well [8].

As we discussed above, there can be multiple different execution strategies for answering the same query, including the various choices of serial scans, indexes, and processing methods (e.g., parallelization, GPU, etc.). The challenge in choosing the right execution strategy is to estimate the amount of data that such a query will need to access before executing it. For example, a fast parallel SIMD-enabled scan on compressed data might be a better option than the use of a non-optimized index when SIMD instructions are available, but not a better choice when such instructions are not available. All these characteristics have to be considered when choosing the best method for answering a particular query.

Finally, we argue for the need of benchmarks that can stress-test sequence processing techniques in a controlled

way and to pre-defined levels of query hardness. A recent work takes the first step in this direction: it shows that the amount of effort employed by data series indexes can be consistently captured across different indexing approaches, using implementation-invariant measures, and proposes a corresponding data series query workload benchmark [50].

IV. Conclusions

In this study, we focused on the problem of data series management[3]. We discussed the state-of-the-art approaches, and observed that they cannot cope with the increasing scalability requirements of modern applications and their massive data series collections. Therefore, we articulate the necessity for rigorous work on parallel and distributed data series techniques, which involves several challenging and exciting research directions.

Acknowledgements
I would like to thank my collaborators (in alphabetical order): Alessandro Camerra, Michele Dallachiesa, Johannes Gehrke, Stratos Idreos, Eamonn Keogh, Michele Linardi, and Yin Lou. Special thanks go to Kostas Zoumpatianos, who has been the driving force behind several of the ideas discussed in this paper.

References

[1] Adhd-200. http://fcon_1000.projects.nitrc.org/indi/adhd200/, 2011.

[2] Sloan digital sky survey. https://www.sdss3.org/dr10/data_access/volume.php, 2015.

[3] T. G. Addair, D. A. Dodge, W. R. Walter, and S. D. Ruppert. Large-scale seismic signal analysis with hadoop. *Computers & Geosciences*, 66:145–154, 2014.

[4] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *FODO*, 1993.

[5] I. Assent, R. Krieger, F. Afschari, and T. Seidl. The ts-tree: Efficient time series search and retrieval. In *EDBT*, 2008.

[6] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. Gray, P. P. Griffiths, W. F. K. III, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. System R: relational approach to database management. *TODS*, 1(2):97–137, 1976.

[7] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *VLDB*, pages 28–39, 1996.

[8] P. Bernstein, S. Bykov, A. Geller, G. Kliot, and J. Thelin. Orleans: Distributed virtual actors for programmability and scalability. MSR-TR-2014-41, 2014.

[9] A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *ICDM*, 2010.

[10] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. J. Keogh. Beyond one billion time series: indexing and mining very large time series collections with isax2+. *KAIS*, 39(1):123–151, 2014.

[11] K.-P. Chan and A.-C. Fu. Efficient time series matching by wavelets. In *ICDE*, 1999.

[12] M. Dallachiesa, B. Nushi, K. Mirylenka, and T. Palpanas. Uncertain time-series similarity: Return to the basics. *PVLDB*, 5(11):1662–1673, 2012.

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

[14] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[3] A more detailed analysis of the topics discussed in this paper can be found in our previous works [31], [19], [30], [9], [12], [10], [47], [13], [50], [48], [27], [28], [29].

[15] P. Huijse, P. A. Estévez, P. Protopapas, J. C. Principe, and P. Zegers. Computational intelligence challenges and applications on large-scale astronomical time series databases. *IEEE CIM*, 9(3):27–39, 2014.

[16] K. Kashino, G. Smith, and H. Murase. Time-series active search for quick retrieval of audio and video. In *ICASSP*, 1999.

[17] S. Kashyap and P. Karras. Scalable knn search on vertically stored time series. In *KDD*, 2011.

[18] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 3(3):263–286, 2000.

[19] E. J. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *VLDB*, 2004.

[20] C.-S. Li, P. Yu, and V. Castelli. Hierarchyscan: a hierarchical similarity search algorithm for databases of long sequences. In *ICDE*, 1996.

[21] H. Liao, J. Han, and J. Fang. Multi-dimensional index on hadoop distributed file system. In *NAS*, 2010.

[22] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, 2003.

[23] J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *J. Intell. Inf. Syst.*, 39(2), 2012.

[24] P. Lu, G. Chen, B. C. Ooi, H. T. Vo, and S. Wu. Scalagist: Scalable generalized search trees for mapreduce systems [innovative systems paper]. *PVLDB*, 7(14):1797–1808, 2014.

[25] K. Mirylenka, V. Christophides, T. Palpanas, I. Pefkianakis, and M. May. Characterizing home device usage from wireless traffic time series. In *EDBT*, pages 551–562, 2016.

[26] A. Mueen, K. Viswanathan, C. Gupta, and E. Keogh. The fastest similarity search algorithm for time series subsequences under euclidean distance, August 2015. http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html.

[27] T. Palpanas. Data series management: The road to big sequence analytics. *SIGMOD Rec.*, 44(2):47–52, 2015.

[28] T. Palpanas. Big sequence management: A glimpse of the past, the present, and the future. In *SOFSEM*, pages 63–80, 2016.

[29] T. Palpanas. Data series management: The next challenge. In *ICDE Workshops*, 2016.

[30] T. Palpanas, M. Vlachos, E. J. Keogh, and D. Gunopulos. Streaming time series summarization using user-defined amnesic functions. *IEEE Trans. Knowl. Data Eng.*, 20(7):992–1006, 2008.

[31] T. Palpanas, M. Vlachos, E. J. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *ICDE*, 2004.

[32] P. Paraskevopoulos, T.-C. Dinh, Z. Dashdorj, T. Palpanas, and L. Serafini. Identification and characterization of human behavior patterns from mobile phone data. In *D4D Challenge session, NetMob*, 2013.

[33] T. Pelkonen, S. Franklin, P. Cavallaro, Q. Huang, J. Meza, J. Teller, and K. Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *PVLDB*, 8(12):1816–1827, 2015.

[34] D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *SIGMOD*, 1997.

[35] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, 2012.

[36] V. Raman, G. K. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Müller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. J. Storm, and L. Zhang. DB2 with BLU acceleration: So much more than just a column store. *PVLDB*, 6(11):1080–1091, 2013.

[37] U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco. Practical data prediction for real-world wireless sensor networks. *IEEE Trans. Knowl. Data Eng.*, accepted for publication, 2015.

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

[39] D. Shasha. Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Eng. Bull.*, 22(2):40–46, 1999.

[40] J. Shieh and E. J. Keogh. *i*sax: indexing and mining terabyte sized time series. In *KDD*, pages 623–631, 2008.

[41] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-store: A column-oriented DBMS. In *VLDB*, 2005.

[42] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman. The architecture of scidb. In *SSDBM*, 2011.

[43] B. Tang, M. L. Yiu, Y. Li, and L. H. U. Exploit every cycle: Vectorized time series algorithms on modern commodity cpus. In *ADMS*, 2016.

[44] Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *PVLDB*, 6(10):793–804, 2013.

[45] L. Ye and E. J. Keogh. Time series shapelets: a new primitive for data mining. In *KDD*, 2009.

[46] Y. Zhu, Z. Zimmerman, N. S. Senobari, C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. J. Keogh. Matrix profile II: exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. In *ICDM*, pages 739–748, 2016.

[47] K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *SIGMOD*, 2014.

[48] K. Zoumpatianos, S. Idreos, and T. Palpanas. RINSE: interactive data series exploration with ADS+. *PVLDB*, 8(12):1912–1923, 2015.

[49] K. Zoumpatianos, S. Idreos, and T. Palpanas. ADS: the adaptive data series index. *VLDB J.*, 25(6):843–866, 2016.

[50] K. Zoumpatianos, Y. Lou, T. Palpanas, and J. Gehrke. Query workloads for data series indexes. In *KDD*, 2015.