RecovDB: accurate and efficient missing blocks recovery for large time series

Ines Arous, Mourad Khayati, Philippe Cudré-Mauroux
University of Fribourg
Fribourg, Switzerland
{ines.arous, mourad.khayati, pcm}@unifr.ch

Ying Zhang, Martin Kersten, Svetlin Stalinlov
MonetDB Solutions
Amsterdam, the Netherlands
{zhang, kersten, stalinlov}@monetdbsolutions.com

***Abstract*—With the emergence of the Internet of Things (IoT), time series data has become ubiquitous in our daily life. Making sense of time series is a topic of great interest in many domains. Existing time series analysis applications generally assume or even require perfect time series (i.e. regular time intervals without unknown values), but real-world time series are rarely so neat. They often contain "holes" of different sizes (i.e. single missing values, or blocks of consecutive missing values) due to some failures or irregular time intervals. Hence, missing value recovery is a prerequisite for many time series analysis applications.**

In this demo, we present RECOVDB, a relational database system enhanced with advanced matrix decomposition technology for missing blocks recovery. This demo will show the main features of RECOVDB that are important for today's time series analysis but are lacking in state-of-the-art technologies: i) recovering large missing blocks in *multiple time series* at once; ii) achieving high recovery accuracy by benefiting from different *correlations* across time series; iii) maintaining *recovery accuracy* under increasing size of missing blocks; iv) maintaining *recovery efficiency* with increasing time series' lengths and the number of time series; and iv) supporting all these features while being *parameter-free*. In this paper, we also compare the efficiency and accuracy of RECOVDB against state-of-the-art recovery systems.

I. INTRODUCTION

Our work is motivated by the properties of acquired data and the requirements for their subsequent analysis in many real-world use cases. Consider for instance meteorology in which sensors are used to record various weather conditions to perform weather forecast. For example, Figure 1 shows three time series containing temperature and humidity values measured in the city of Basel, Switzerland during 2017 – 2018[1]. When analyzing such meteorological time series, there are a number of important aspects to take into account. First, within the same (scientific) domain, different time series often exhibit some types of correlations. In Figure 1, the two temperature time series, "Temp. 2017" and "Temp. 2018", have a *positive correlation* with each other, while they both have a *negative correlation* with the humidity time series "Humidity 2018". Second, real-world time series often contain a large number of blocks of missing values due to sensor failures, power outages, transmission problems, etc. Some missing blocks can be rather big (shown as dashed lines in Figure 1), because, for instance, it can take minutes, hours or even days for a broken sensor to be replaced. Finally, many of the analysis tools and prediction

[1]Source: https://www.meteoblue.com

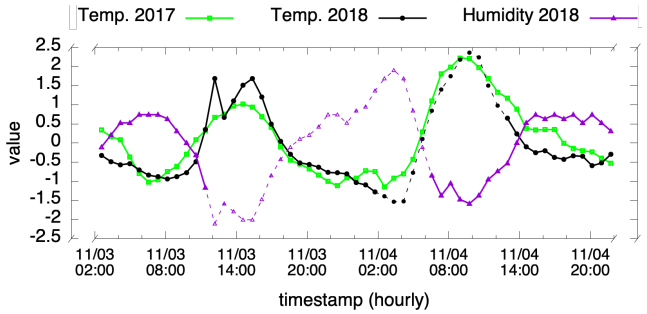


Fig. 1: Meteorological time series of temperature and humidity values measured in Basel, Switzerland during 2017 – 2018. The missing blocks are denoted by dashed lines.

models that meteorologists use to perform weather forecast require *complete time series* (i.e. the set of the input time series must have the same length, same time interval and all values are known).

In addition, even if a tool can work with incomplete time series, missing values are considered harmful. For instance, missing values often yield incorrect or ill-defined query results [1]. Also, missing values can unexpectedly introduce bias into the time series which might significantly alter their statistical properties, such as the correlations between time series. This in turn can affect further data analysis tasks, e.g. data sampling, exploration and prediction, rendering their results pointless.

Given the amount of data we nowadays need to deal with and the requirements of existing models/tools, *efficient and accurate recovery of large missing blocks in time series* has become a prerequisite to enable the work of many analytical applications.

Existing recovery techniques have several drawbacks. They either focus on repairing very small missing blocks (i.e., single missing values or only a handful of consecutive missing values) which generally can not yield a high accuracy when applied on big missing blocks [2], or they repair individual time series, while ignoring their correlations with other similar time series [3]. This limits the recovery accuracy, because in many systems today, multiple sensors are used to record the same/similar measurement, which makes using correlation beneficial in many applications such as error detection or missing values recovery.

Moreover, existing techniques are often stand-alone, as opposed to being integrated into a database system. As a result of this, the users of these tools have to conduct a number of time consuming and error-prune tasks themselves, such as either export/import time series from/into a database or do all data management work themselves, repeatedly load the data files and convert them to some internal format, and write code for every action that needs to be conducted on the time series (e.g. filtering and aggregations).

To overcome the aforementioned problems, we built RECOVDB, a relational database system (RDBMS) enhanced with advanced missing blocks recovery technology, which is based on our memory-efficient matrix decomposition technique, the Centroid Decomposition (CD) [4], to perform scalable and accurate recovery of missing values. The recovery algorithm has been *tightly* integrated into the open-source analytical RDBMS MonetDB [5] as native User Defined Functions (UDFs), which enables zero data conversion and transfer costs[2]. With this architecture, RECOVDB has a number of properties that are highly desirable for missing blocks recovery in large time series, which many of the existing recovery techniques fall short in providing (one or a combination of):

Parameter-free recovery Parametric recovery techniques are based on fine-tuning some input parameters which requires an expertise of the application field and the types of time series. RECOVDB avoids parameter tuning by performing recovery based on the *centroid value* of all time series. The centroid value is the only statistical property we use for the recovery.

Correlation-aware recovery The CD algorithm embeds the correlation across time series yielding a recovery with better performance *and* higher accuracy.

Large missing blocks in multiple time series By using advanced matrix decomposition technique, RECOVDB is capable of accurately recovering multiple time series with large missing blocks in one go, something that cannot be handled well by standard statistical methods such as interpolations.

Full-fledged DBMS support Due to the tight integration, RECOVDB can exploit MonetDB's full power as a highly optimized analytical RDBMS to handle the remaining data management and pre-/post-processing work.

In this demo, we will show all the above features using real-world datasets through an interactive GUI. The demo contains three incremental scenario's. Scenario 1 shows that RECOVDB can recover multiple time series in one go. Scenario 2 shows that RECOVDB maintains the recovery accuracy high when increasing sizes of missing blocks. Scenario 3 shows that even with more and/or longer time series, RECOVDB can still recover the missing blocks efficiently. For all three scenarios, a visitor merely needs to select the desired time series and press the "Recover" button. Finally the demo will be accompanied by a poster to elaborate on how the CD and recovery algorithms work.

[2]MonetDB is implemented in C and its internal data storage is a single C-array per column.

The remainder of this paper is organized as follows. Section II describes our recovery algorithm. Section III describes the implementation of RECOVDB and presents some evaluation results against two state-of-the-art systems on recovery efficiency and accuracy. Section IV elaborates the demo scenario's. Finally, Section V concludes.

II. MISSING VALUES RECOVERY

The recovery algorithm is based on our memory-efficient algorithm to compute the Centroid Decomposition (CD) for long time series [4]. This section first defines several basic concepts used in CD, before describing the recovery algorithm.

Centroid Decomposition (CD). Let $\mathbf{X}$ be an $n \times m$ matrix containing m time series each with n numerical values. CD decomposes $\mathbf{X}$ into an $n \times m$ matrix $\mathbf{L}$ and an $m \times m$ matrix $\mathbf{R}$ s.t. $\mathbf{X} = \mathbf{L} \cdot \mathbf{R}^T$, where $\mathbf{R}^T$ is the transpose of $\mathbf{R}$. Given a vector Z, the function $CD(\mathbf{X}, Z, k)$ returns the first k columns of $\mathbf{L}$ and $\mathbf{R}$ s.t. their product $\widetilde{\mathbf{X}}$ is an approximation of $\mathbf{X}$.

The most challenging part of computing the CD of $\mathbf{X}$ is to find the *maximizing sign vector* Z, which contains only 1s and -1s, that maximizes the *centroid value* $\|\mathbf{X}^T \cdot Z\|$, where $\mathbf{X}^T$ is the transpose of $\mathbf{X}$ and $\|\cdot\|$ denotes the norm of a vector. To efficiently compute Z, we use our Scalable Sign Vector (SSV) algorithm [4], which embeds the correlation across time series without constructing the correlation matrix. The SSV algorithm maintains linear complexity for both time and space with an increasing number of time series.

Recovery Algorithm. Algorithm 1 depicts our recovery algorithm `RecovM`. It uses *CD* to recover missing values in multiple time series in one go. `RecovM` takes as input a matrix $\mathbf{X}$ and a list T^- of pairs indicating the rows and columns of the missing values in $\mathbf{X}$. The recovery starts by initializing the missing values using linear interpolation (line 1). Then, we use SSV to efficiently compute Z (line 2), which is then used to compute the approximated matrix $\widetilde{\mathbf{X}}$ (lines 4-6). Finally, the values in $\mathbf{X}$ with positions in T^- are updated with their corresponding ones in $\widetilde{\mathbf{X}}$ (lines 6-8). The recovery process continues until the Frobenius difference $\|\mathbf{X} - \mathbf{X}'\|_F$ (defined as: $\sqrt{\sum_{i=1}^{n}(x_i - x'_i)^2}$, where $x_i \in \mathbf{X}$, $x'_i \in X'$) between $\mathbf{X}$ and $\mathbf{X}'$ falls below a small threshold ϵ (by default 10^{-5}) (lines 3-9).

Algorithm 1: RecovM($\mathbf{X}$, T^-)

Input : $n \times m$ matrix $\mathbf{X}$; List of missing time points T^-
Output: Matrix with recovered values $\widetilde{\mathbf{X}}$

1 Linearly interpolate all missing values in $\mathbf{X}$;
2 $Z \coloneqq SSV(\mathbf{X})$; $k \coloneqq 1$;
3 **repeat**
4 $\quad \mathbf{X}' \coloneqq \mathbf{X}$;
5 $\quad \widetilde{\mathbf{L}}, \widetilde{\mathbf{R}} \coloneqq CD(\mathbf{X}', Z, k)$;
6 $\quad \widetilde{\mathbf{X}} \coloneqq \widetilde{\mathbf{L}} \cdot \widetilde{\mathbf{R}}^T$;
$\quad$ `// Update missing values`
7 $\quad$ **foreach** $(i, j) \in T^-$ **do**
8 $\quad\quad x_{ij} \coloneqq \widetilde{x}_{ij}$;
9 **until** $\|\mathbf{X} - \mathbf{X}'\|_F < \epsilon$;
10 **return** $\widetilde{\mathbf{X}}$;

III. Implementation and Evaluation

Implementation. RecovDB is implemented using MonetDB, an open-source RDBMS optimized for in-memory processing of analytical workloads [5]. Internally, MonetDB stores the data of each SQL column as a single C array. This columnar storage model matches nicely with time series data.

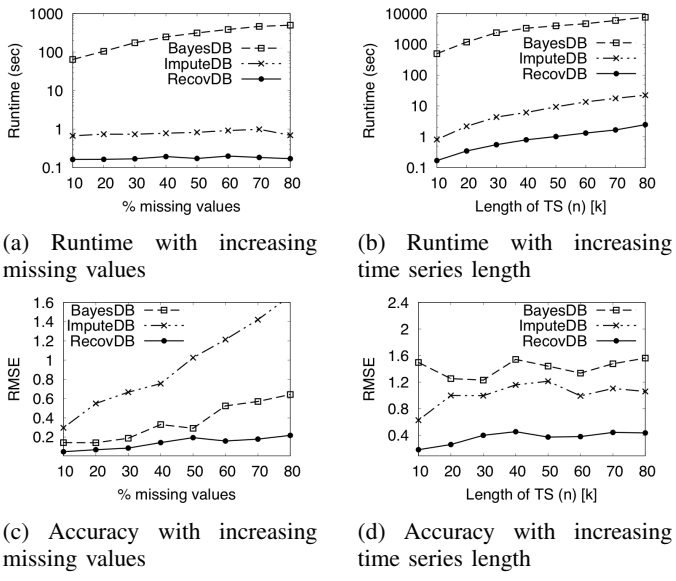
```
CREATE TABLE tss(ts timestamp, v1 float, .., vn float);
CREATE FUNCTION recov(ts timestamp, v1 float, .., vn float)
  RETURNS TABLE (ts timestamp, f1 float, .., fn float) ...;

SELECT * FROM recov((SELECT ts, v1, v3, v7 FROM tss
                     WHERE ts BETWEEN $TS_MIN AND $TS_MAX));
```

The SQL queries above show a skeleton of the implementation. First, we store all time series of one dataset in a single table[3] containing one timestamp column and a number of value columns to hold the values of the time series (after some preprocessing to align the timestamps). Then, we implemented `RecovM` (see Algorithm 1) and its auxiliary functions as native SQL UDFs in MonetDB. The table returning function `recov` is a wrapper for `RecovM`. It takes one column of timestamps and multiple columns of time series values containing `NULL`s as missing blocks as its inputs, and passes the value columns to `RecovM` for imputation. The recovered time series are returned together with the original timestamps in a single table[4]. Finally, we use `recov` in a `SELECT` query to recover the values of three time series within a given time period.

Evaluation. To evaluate the efficiency and accuracy of RecovDB, we have conducted extensive experiments against two state-of-the-art systems in missing value imputation: ImputeDB [1] and BayesDB [6]. We measure i) the runtime to perform the full recovery, and ii) the accuracy of the recovery using the Root Mean Square Error (RMSE) between the original block and the recovered one.

We used the BAFU dataset provided by the BundesAmt Für Umwelt (the Swiss Federal Office for the Environment)[5]. This dataset contains water discharge time series of 12 different Swiss rivers recorded every 30 min during 2010 – 2015 resulting in 80k records per time series. The Pearson Correlation Coefficients (PCC)[6] between these time series range from 0.03 (very low) to 0.89 (quite high), which allows us to evaluate RecovDB for different correlation "strengths". Each tuple in a time series contains a timestamp and the value of the measurement.

To evaluate the efficiency of RecovDB, we used two set-ups. First, we fixed the length of the 12 time series to $10k$, while incrementally dropping 10% of successive values from three different time series (the missing blocks are partially overlapping). Figure 2a shows that the runtime of RecovDB is barely sensitive to the percentage of missing values and is up to 5x faster than ImputeDB (0.17sec vs. 1sec) and up to 2900x faster than BayesDB (0.17sec vs. 500sec). Second, we increase the length of all 12 time series from $10k$ to $80k$ (Figure 2b). Our results show that RecovDB is up to 10x faster than ImputeDB (1.67sec vs 17.77sec), and up to four orders of magnitude faster than BayesDB (0.8sec vs 3343sec).



(a) Runtime with increasing missing values

(b) Runtime with increasing time series length

(c) Accuracy with increasing missing values

(d) Accuracy with increasing time series length

Fig. 2: RecovDB performance

To evaluate the accuracy of RecovDB, we used the same two set-ups, but measured the RMSEs instead. Figure 2c shows that the RMSE of RecovDB is barely affected with more missing values while the RMSE of ImputeDB and BayesDB tends to increase with the percentage of missing values. RecovDB is up to 7.9x more accurate than ImputeDB (0.18 vs. 1.42) and 3.4x more accurate than BayesDB (0.17 vs. 0.57). Figure 2d shows that when increasing the length of time series, RecovDB is up to 8.3x more accurate than ImputeDB (0.18 vs 1.49) and 4x more accurate than BayesDB (0.25 vs 0.99).

IV. RecovDB Demonstration

Demo GUI. Figure 3 shows a screenshot of the demo GUI[7]. Above the graph, users can load more data with the "load time series" button. Seven time series were already loaded and each was assigned a different color and is denoted by its name. Users can activate/deactivate a time series by clicking on its legend. The values of the three selected time series are visualized in the main graph below. The sliding bar under the main graph shows that we have zoomed in to only view their values in 1985. At the right-side of the GUI, some textual information about the loaded time series is given, followed by some simple options to change the default number of the time series to be repaired, the time series to be used for correlation, the recovery threshold (i.e. ϵ in Algorithm 1), and the percentage of additional missing values to introduce. By clicking the Recover button, users can start the recovery process with their configuration. Some statistics will be displayed in the text

[3]Assuming those time series contain related information, e.g., a weather dataset can contain time series of temperature and wind speed.

[4]Passing around (a pointer to) the `ts` column is merely an easy way to keep the repaired time series annotated with their timestamps. This does not incur additional space and computation.

[5]https://www.bafu.admin.ch/

[6]https://en.wikipedia.org/wiki/Pearson_correlation_coefficient

[7]http://revival.exascale.info

Fig. 3: GUI of the RecovDB demo

fields under this button. The recovered time series is visualized in the main window.

Demo Datasets. In addition to the BAFU dataset described in Section III, we will provide two more datasets. The MeteoSwiss dataset, provided by the Swiss Federal Office of Meteorology and Climatology (meteoswiss.admin.ch), has 20 weather time series each containing 200k records measured every 10 min in different Swiss cities. Their PCCs range from -0.12 to 0.9, which allows users to evaluate RecovDB with both positive and negative correlations. The Gas concentration dataset [7] contains the concentration level of 24 chemical substances each represented as a time series of 4k records measured every 6 hrs. These measurements were collected in a gas delivery platform facility situated at the University of California in San Diego. This dataset contains a larger range of negatively correlated time series, with PCCs in $[-0.75, 0.78]$.

Demo Scenario's. This demo will provide three scenarios to showcase the main properties of RecovDB.

S1: recover multiple time series at once. In this scenario, users can select multiple time series with different number of missing blocks of different lengths in the time series. The locations of missing blocks across different time series can be distinct or (partially/fully) overlapping. By clicking the Recover button, users can observe that all missing blocks in all time series are repaired in one go. If the missing blocks have been artificially introduced (see S2 below), the demo GUI will display both the original values and the repaired values so that users can easily compare the quality of the repair.

S2: accurate recovery with increasing missing values. In this scenario, users can specify a percentage of missing values in a text field. The GUI will remove this percentage of values from the selected range of the selected time series. By clicking the Recover button, all time series will be repaired at once. To compare the accuracy of the recovery, the GUI will show the original values in solid lines, and the recovered values in dotted lines. By specifying larger percentages, users shall be able to observe a fairly stable accurate recovery.

S3: efficient recovery with increasing data size. In this scenario, users can vary the numbers of selected time series and/or the range of the timestamps. For each configuration, the total recovery time will be displayed under the Recover button. By increasing the selection, users shall be able to observe that the execution time barely increases.

V. Conclusions

In this demo, we demonstrate through RecovDB the benefits of the CD-based recovery technique on top of a relational DBMS. This demo allows the audience to experience how to recover large and multiple time series without necessarily finding a trade-off between efficiency and accuracy.

Acknowledgments

This work is funded by the European Union's Horizon 2020 research and innovation programme under grant agreement No 732328 (FashionBrain).

References

[1] J. Cambronero *et al.*, "Query optimization for dynamic imputation," *PVLDB*, vol. 10, no. 11, pp. 1310–1321, 2017.
[2] J. Sun *et al.*, "Online latent variable detection in sensor networks," in *ICDE*, April 2005, pp. 1126–1127.
[3] K. Wellenzohn *et al.*, "Continuous imputation of missing values in streams of pattern-determining time series," in *EDBT*, March 2017.
[4] M. Khayati *et al.*, "Memory-efficient centroid decomposition for long time series," in *ICDE*, April 2014.
[5] P. A. Boncz *et al.*, "Breaking the memory wall in MonetDB," *Commun. ACM*, vol. 51, no. 12, pp. 77–85, 2008.
[6] F. Saad and V. K. Mansinghka, "A probabilistic programming approach to probabilistic data analysis," in *NIPS*, 2016, pp. 2011–2019.
[7] I. Rodriguez-Lujan *et al.*, "On the calibration of sensor arrays for pattern recognition using the minimal number of experiments," *Chemometrics and Intelligent Laboratory Systems*, vol. 130, pp. 123 – 134, 2014.