Missing Value Imputation in Time Series using Top-k Case Matching

Kevin Wellenzohn
Free University of Bozen-Bolzano
kevin.wellenzohn@unibz.it

Hannes Mitterer
Free University of Bozen-Bolzano
hannes.mitterer@unibz.it

Johann Gamper
Free University of Bozen-Bolzano
gamper@inf.unibz.it

M. H. Böhlen
University of Zurich
boehlen@ifi.uzh.ch

Mourad Khayati
University of Zurich
mkhayati@ifi.uzh.ch

ABSTRACT

In this paper, we present a simple yet effective algorithm, called the Top-k Case Matching algorithm, for the imputation of missing values in streams of time series data that are similar to each other. The key idea of the algorithm is to look for the k situations in the historical data that are most similar to the current situation and to derive the missing value from the measured values at these k time points. To efficiently identify the top-k most similar historical situations, we adopt Fagin's Threshold Algorithm, yielding an algorithm with sub-linear runtime complexity with high probability, and linear complexity in the worst case (excluding the initial sorting of the data, which is done only once). We provide the results of a first experimental evaluation using real-world meteorological data. Our algorithm achieves a high accuracy and is more accurate and efficient than two more complex state of the art solutions.

Keywords

Time series, imputation of missing values, Threshold Algorithm

1. INTRODUCTION

Time series data is ubiquitous, e.g., in the financial stock market or in meteorology. In many applications time series data is incomplete, that is some values are missing for various reasons, e.g., sensor failures or transmission errors. However, many applications assume complete data, hence need to recover missing values before further data processing is possible.

In this paper, we focus on the imputation of missing values in long streams of meteorological time series data. As a case study, we use real-world meteorological data collected by the *Südtiroler Beratungsring*[1] (*SBR*), which is an organization that provides professional and independent consultancy to the local wine and apple farmers, e.g., to determine the optimal harvesting time or to warn about potential threats, such as apple scab, fire blight, or frost. Especially frost is dangerous as it can destroy the harvest within a few minutes unless the farmers react immediately. The *Südtiroler Beratungsring* operates more than 120 weather stations spread all over South Tyrol, where each of them collects every five minutes up to 20 measurements including temperature, humidity etc. The weather stations frequently suffer outages due to sensor failures or errors in the transmission of the data. However, the *continuous monitoring* of the current weather condition is crucial to immediately warn about imminent threats such as frost and therefore the need arises to recover those missing values as soon as they are detected.

In this paper, we propose an accurate and efficient method to automatically recover missing values. The need for a continuous monitoring of the weather condition at the *SBR* has two important implications for our solution. Firstly, the proposed algorithm has to be efficient enough to complete the imputation before the next set of measurements arrive in a few minutes time. Secondly, the algorithm cannot use future measurements which would facilitate the imputation, since they are not yet available.

The key idea of our Top-k Case Matching algorithm is to seek for the k time points in the historical data when the measured values at a set of reference stations were most similar to the measured values at the current time point (i.e., the time point when a value is missing). The missing value is then derived from the values at the k past time points. While a naïve solution to identify the top-k most similar historical situations would have to scan the entire data set, we adopt Fagin's Threshold Algorithm, which efficiently answers top-k queries by scanning, on average, only a small portion of the data. The runtime complexity of our solution is derived from the Threshold Algorithm and is sub-linear with high probability and linear in the worst case, when all data need to be scanned. We provide the results of a first experimental evaluation using real-world meteorological data from the *SBR*. The results are promising both in terms of efficiency and accuracy. Our algorithm achieves a high accuracy and is more accurate than two state of the art solutions.

The rest of the paper is organized as follows. In Section 2, we review the existing literature about imputation methods for missing values. In Section 3, we introduce the basic notation and a running example. In Section 4, we present our Top-k Case Matching algorithm for the imputation of missing values, followed by the results of an experimental evaluation in Section 5. Section 6 concludes the paper and outlines ideas for future work.

2. RELATED WORK

Khayati et al. [4] present an algorithm, called REBOM, which

[1] http://www.beratungsring.org/

In: G. Specht, H. Gamper, F. Klan (eds.): Proceedings of the 26th GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 21.10.2014 - 24.10.2014, Bozen, Italy, published at http://ceur-ws.org.

recovers blocks of missing values in irregular (with non repeating trends) time series data. The algorithm is based on an iterated truncated matrix decomposition technique. It builds a matrix which stores the time series containing the missing values and its k most correlated time series according to the Pearson correlation coefficient [7]. The missing values are first initialized using a simple interpolation technique, e.g., linear interpolation. Then, the matrix is iteratively decomposed using the truncated *Singular Value Decomposition* (SVD). By multiplying the three matrices obtained from the decomposition, the algorithm is able to accurately approximate the missing values. Due to its quadratic runtime complexity, REBOM is not scalable for long time series data.

Khayati et al. [5] further investigate the use of matrix decomposition techniques for the imputation of missing values. They propose an algorithm with linear space complexity based on the *Centroid Decomposition*, which is an approximation of SVD. Due to the memory-efficient implementation, the algorithm scales to long time series. The imputation follows a similar strategy as the one used in REBOM.

The above techniques are designed to handle missing values in static time series. Therefore, they are not applicable in our scenario, as we have to continuously impute missing values as soon as they appear. A naïve approach to run the algorithms each time a missing value occurs is not feasible due to their relatively high runtime complexity.

There are numerous statistical approaches for the imputation of missing values, including easy ones such as linear or spline interpolation, all the way up to more complex models such as the ARIMA model. The ARIMA model [1] is frequently used for forecasting future values, but can be used for *backcasting* missing values as well, although this is a less common use case. A recent comparison of statistical imputation techniques for meteorological data is presented in [9]. The paper comprises several simple techniques, such as the (weighted) average of concurrent measurements at nearby reference stations, but also computationally more intensive algorithms, such as neural networks.

3. BACKGROUND

Let $\mathbf{S} = \{s_1, \ldots, s_n\}$ be a set of time series. Each time series, $s \in \mathbf{S}$, has associated a set of *reference time series* $\mathbf{R}_s$, $\mathbf{R}_s \subseteq \mathbf{S} \setminus \{s\}$. The value of a time series $s \in \mathbf{S}$ at time t is denoted as $s(t)$. A *sliding window* of a time series s is denoted as $s([t_1, t_2])$ and represents all values between t_1 and t_2.

EXAMPLE 1. *Table 1 shows four temperature time series in a time window $w = [1, 7]$, which in our application corresponds to seven timestamps in a range of 30 minutes. s is the base time series from the weather station in* Schlanders, *and $\mathbf{R}_s = \{r_1, r_2, r_3\}$ is the associated set of reference time series containing the stations of Kortsch, Göflan, and Laas, respectively. The temperature value $s(7)$ is missing. Figure 1 visualizes this example graphically.*

The Top-k Case Matching algorithm we propose assumes that the time series data is aligned, which generally is not the case for our data. Each weather station collects roughly every 5 minutes new measurements and transmits them to a central server. Since the stations are not perfectly synchronized, the timestamps of the measurements typically differ, e.g., one station collects measurements at 09:02, 09:07, …, while another station collects them at 09:04, 09:09, …. Therefore, in a pre-processing step we align the time series data using linear interpolation, which yields measurement values every 5 minutes (e.g., 00:00, 00:05, 00:10, …). If we observe a gap of more than 10 minutes in the measurements, we assume that the value is missing.

$t \in w$	$s(t)$	$r_1(t)$	$r_2(t)$	$r_3(t)$
1	16.1°	15.0°	15.9°	14.1°
2	15.8°	15.2°	15.7°	13.9°
3	15.9°	15.2°	15.8°	14.1°
4	16.2°	15.0°	15.9°	14.2°
5	16.5°	15.3°	15.7°	14.5°
6	16.1°	15.2°	16.0°	14.1°
7	?	15.0°	16.0°	14.3°

Table 1: Four time series in a window $w = [1, 7]$.

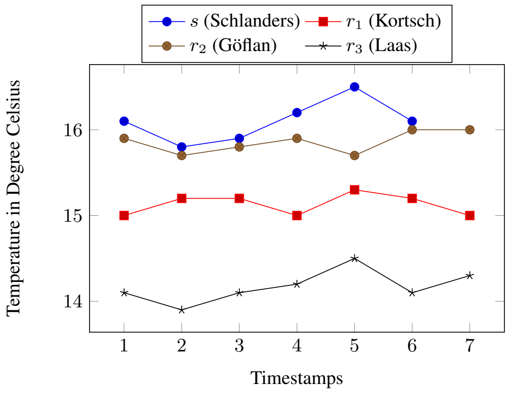

Figure 1: Visualization of the time series data.

For the imputation of missing values we assign to each time series s a set $\mathbf{R}_s$ of reference time series, which are similar to s. The notion of *similarity* between two time series is tricky, though. Intuitively, we want time series to be similar when they have similar values and behave similarly, i.e., values increase and decrease roughly at the same time and by the same amount.

As a simple heuristic for time series similarity, we use the spatial proximity between the stations that record the respective time series. The underlying assumption is that, if the weather stations are nearby (say within a radius of 5 kilometers), the measured values should be similar, too. Based on this assumption, we manually compiled a list of 3–5 reference time series for each time series. This heuristic turned out to work well in most cases, though there are situations where the assumption simply does not hold. One reason for the generally good results is most likely that in our data set the over 100 weather stations cover a relatively small area, and hence the stations are very close to each other.

4. TOP-K CASE MATCHING

Weather phenomena are often repeating, meaning that for example during a hot summer day in 2014 the temperature measured at the various weather stations are about the same as those measured during an equally hot summer day in 2011. We use this observation for the imputation of missing values. Let s be a time series where the current measurement at time θ, $s(\theta)$, is missing. Our assumption on which we base the imputation is as follows: if we find historical situations in the reference time series $\mathbf{R}_s$ such that the past values are very close to the current values at time θ, then also the past measurements in s should be very similar to the missing value $s(\theta)$. Based on this assumption, the algorithm searches for similar climatic situations in historical measurements, thereby leveraging the vast history of weather records collected by the *SBR*.

More formally, given a base time series s with reference time series $\mathbf{R}_s$, we are looking for the k timestamps (i.e., historical situations), $\mathbf{D} = \{t_1, \ldots, t_k\}$, $t_i < \theta$, which minimize the error function

$$\delta(t) = \sum_{r \in \mathbf{R}_s} |r(\theta) - r(t)|.$$

That is, $\delta(t) \leq \delta(t')$ for all $t \in \mathbf{D}$ and $t' \notin \mathbf{D} \cup \{\theta\}$. The error function $\delta(t)$ is the accumulated absolute difference between the current temperature $r(\theta)$ and the temperature at time t, $r(t)$, over all reference time series $r \in \mathbf{R}_s$. Once $\mathbf{D}$ is determined, the missing value is recovered using some aggregation function $g\left(\{s(t)|\forall t \in \mathbf{D}\}\right)$ over the measured values of the time series s at the timestamps in $\mathbf{D}$. In our experiments we tested the average and the median as aggregation function (cf. Section 5).

EXAMPLE 2. *We show the imputation of the missing value $s(7)$ in Table 1 using as aggregation function g the average. For the imputation, we seek the $k = 2$ most similar historical situations. The two timestamps $\mathbf{D} = \{4, 1\}$ minimize $\delta(t)$ with $\delta(4) = |15.0° - 15.0°| + |16.0° - 15.9°| + |14.3° - 14.2°| = 0.2°$ and $\delta(1) = 0.3°$. The imputation is then simply the average of the base station measurements at time $t = 4$ and $t = 1$, i.e.,$s(7) = avg(16.2°, 16.1°) = \frac{1}{2}(16.2° + 16.1°) = 16.15°$.*

A naïve implementation of this algorithm would have to scan the entire database of historical data to find the k timestamps that minimize $\delta(t)$. This is, however, not scalable for huge time series data, hence a more efficient technique is needed.

4.1 Fagin's Threshold Algorithm

What we are actually trying to do is to answer a top-k query for the k timestamps which minimize $\delta(t)$. There exist efficient algorithms for top-k queries. For example, Fagin's algorithm [2] solves this problem by looking only at a small fraction of the data. Since the first presentation of Fagin's algorithm there were two noteworthy improvements, namely the *Threshold Algorithm* (TA) by Fagin et al. [3] and a probabilistic extension by Theobald et al. [8]. The latter approach speeds up TA by relaxing the requirement to find the exact top-k answers and providing approximations with probabilistic guarantees.

Our Top-k Case Matching algorithm is a variation of TA with slightly different settings. Fagin et al. assume objects with m attributes, a *grade* for each attribute and a *monotone aggregation function* $f : \mathbb{R}^m \mapsto \mathbb{R}$, which aggregates the m grades of an object into an overall grade. The *monotonicity* property is defined as follows.

DEFINITION 1. (Monotonicity) *Let $x_1, \ldots, x_m$ and $x'_1, \ldots, x'_m$ be the m grades for objects X and X', respectively. The aggregation function f is monotone if $f(x_1, \ldots, x_m) \leq f(x'_1, \ldots, x'_m)$ given that $x_i \leq x'_i$ for each $1 \leq i \leq m$.*

The TA finds the k objects that *maximize* the function f. To do so it requires two modes of accessing the data, one being *sorted* and the other *random* access. The sorted access is ensured by maintaining a sorted list $\mathbf{L}^i$ for each attribute m_i, ordered by the grade in descending order. TA keeps a bounded buffer of size k and scans each list $\mathbf{L}^i$ in parallel until the buffer contains k objects and the lowest ranked object in the buffer has an aggregated grade that is greater than or equal to some threshold τ. The threshold τ is computed using the aggregation function f over the grades last seen under the sorted access for each list $\mathbf{L}^i$.

EXAMPLE 3. *Table 2 shows four objects $\{A, B, C, D\}$ and their grade for the two attributes* `interestingness` *and* `popularity`*. Let us assume that $k = 2$ and the aggregation function $f(x_1, x_2) = x_1 + x_2$. Further, assume that the bounded buffer currently contains $\{(C, 18), (A, 16)\}$ and the algorithm has read the data up to the boxes shown in gray. At this point the algorithm computes the threshold using the* `interestingness` *grade for object B and the* `popularity` *grade of object C, yielding $\tau = f(5, 9) = 5 + 9 = 14$. Since the lowest ranked object in the buffer, object A, has an aggregated grade that is greater than τ, we can conclude that C and A are the top-2 objects. Note that the algorithm never read object D, yet it can conclude that D cannot be part of the top-k list.*

interestingness

Object	grade
A	10
C	9
B	5
D	4

popularity

Object	grade
B	10
C	9
D	8
A	6

Table 2: Threshold Algorithm example.

4.2 Adapting the Threshold Algorithm

In order to use the Threshold Algorithm for the imputation of missing values in time series data, we have to adapt it. Instead of looking for the top-k objects that maximize the aggregation function f, we want to find the top-k timestamps that minimize the error function $\delta(t)$ over the reference time series $\mathbf{R}_s$. Similar to TA, we need sorted access to the data. Therefore, for each time series $r \in \mathbf{R}_s$ we define $\mathbf{L}^r$ to be the time series r ordered first by value and then by timestamp in ascending order. Table 3 shows the sorted data for the three reference time series of our running example (ignore the gray boxes and small subscript numbers for the moment).

$\mathbf{L}^{r_1}$

t	$r_1(t)$
1	15.0° $_4$
4	15.0° $_1$
7	15.0°
2	15.2°
3	15.2°
6	15.2°
5	15.3°

$\mathbf{L}^{r_2}$

t	$r_2(t)$
2	15.7°
5	15.7°
3	15.8°
1	15.9°
4	15.9° $_5$
6	16.0° $_2$
7	16.0°

$\mathbf{L}^{r_3}$

t	$r_3(t)$
2	13.9°
1	14.1°
3	14.1°
6	14.1°
4	14.2° $_3$
7	14.3°
5	14.5° $_6$

Table 3: Time series sorted by temperature.

The general idea of our modified TA algorithm is the following. The scan of each sorted lists starts at the current element, i.e., the element with the timestamp $t = \theta$. Instead of scanning the lists $\mathbf{L}^{r_i}$ only in one direction as TA does, we scan each list sequentially in two directions. Hence, as an initialization step, the algorithm places two pointers, pos_r^+ and pos_r^-, at the current value $r(\theta)$ of time series r (the gray boxes in Table 3). During the execution of the algorithm, pointer pos_r^+ is only incremented (i.e., moved down the list), whereas pos_r^- is only decremented (i.e., moved up the list). To maintain the k highest ranking timestamps, the algorithm uses a bounded buffer of size k. A new timestamp t' is added only if the buffer is either not yet full or $\delta(t') < \delta(\underline{t})$, where $\underline{t}$ is the last (i.e., lowest ranking) timestamp in the buffer. In the latter case the timestamp $\underline{t}$ is removed from the buffer.

After this initialization, the algorithm iterates over the lists $\mathbf{L}^r$ in round robin fashion, i.e., once the last list is reached, the algorithm wraps around and continues again with the first list. In each iteration, exactly one list $\mathbf{L}^r$ is processed, and either pointer pos_r^+ or pos_r^- is advanced, depending on which value the two pointers point to has a smaller absolute difference to the current value at time θ, $r(\theta)$. This process grows a neighborhood around the element $r(\theta)$ in each list. Whenever a pointer is advanced by one position, the timestamp t at the new position is processed. At this point, the algorithm needs random access to the values $r(t)$ in each list to compute the error function $\delta(t)$. Time t is added to the bounded buffer using the semantics described above.

The algorithm terminates once the error at the lowest ranking timestamp, $\underline{t}$, among the k timestamps in the buffer is less or equal to the threshold, i.e., $\delta(\underline{t}) \leq \tau$. The threshold τ is defined as $\tau = \sum_{r \in \mathbf{R}_s} |r(\theta) - r(pos_r)|$, where pos_r is either pos_r^+ or pos_r^-, depending on which pointer was advanced last. That is, τ is the sum over all lists $\mathbf{L}^r$ of the absolute differences between $r(\theta)$ and the value under pos_r^+ or pos_r^-.

EXAMPLE 4. *We illustrate the Top-k Case Matching algorithm for $k = 2$ and $\theta = 7$. Table 4 shows the state of the algorithm in each iteration i. The first column shows an iteration counter i, the second the buffer with the k current best timestamps, and the last column the threshold τ. The buffer entries are tuples of the form $(t, \delta(t))$. In iteration $i = 1$, the algorithm moves the pointer to $t = 4$ in list $\mathbf{L}^{r_1}$ and adds $(t = 4, \delta(4) = 0.2°)$ to the buffer. Since $\delta(4) = 0.2° > 0.0° = \tau$, the algorithm continues. The pointer in $\mathbf{L}^{r_2}$ is moved to $t = 6$, and $(6, 0.4°)$ is added to the buffer. In iteration $i = 4$, timestamp 6 is replaced by timestamp 1. Finally, in iteration $i = 6$, the error at timestamp $t = 1$ is smaller or equal to τ, i.e., $\delta(1) = 0.3° \leq \tau_6 = 0.3°$. The algorithm terminates and returns the two timestamps $\mathbf{D} = \{4, 1\}$.*

Iteration i	Buffer	Threshold τ_i
1	$(4, 0.2°)$	0.0°
2	$(4, 0.2°), (6, 0.4°)$	0.0°
3	$(4, 0.2°), (6, 0.4°)$	0.1°
4	$(4, 0.2°), (1, 0.3°)$	0.1°
5	$(4, 0.2°), (1, 0.3°)$	0.2°
6	$(4, 0.2°), (1, 0.3°)$	0.3°

Table 4: Finding the $k = 2$ most similar historical situations.

4.3 Implementation

Algorithm 1 shows the pseudo code of the Top-k Case Matching algorithm. The algorithm has three input parameters: a set of time series $\mathbf{R}_s$, the current timestamp θ, and the parameter k. It returns the top-k most similar timestamps to the current timestamp θ. In line 2 the algorithm initializes the bounded buffer of size k, and in line 4 the pointers pos_r^+ and pos_r^- are initialized for each reference time series $r \in \mathbf{R}_s$. In each iteration of the loop in line 7, the algorithm advances either pos_r^+ or pos_r^- (by calling Algorithm 2) and reads a new timestamp t. The timestamp t is added to the bounded buffer using the semantics described before. In line 15, the algorithm computes the threshold τ. If the buffer contains k timestamps and we have $\delta(\underline{t}) \leq \tau$, the top-$k$ most similar timestamps were found and the algorithm terminates.

Algorithm 2 is responsible for moving the pointers pos_r^+ and pos_r^- for each list $\mathbf{L}^r$. The algorithm uses three utility functions. The first is `next()`, which takes a pointer as input and returns the next position by either incrementing or decrementing, depending on the direction of the pointer. If `next()` reaches the end of a list, it returns NIL. The utility functions `timestamp()` and `value()` return the timestamp and value of a list $\mathbf{L}^r$ at a given position, respectively. There are four cases, which the algorithm has to distinguish:

```
Algorithm 1: Top−k Case Matching
Data: Reference time series $\mathbf{R}_s$, current time $\theta$, and $k$
Result: $k$ timestamps that minimize $\delta(t)$
1  $L \leftarrow \{\mathbf{L}^r | r \in \mathbf{R}_s\}$
2  buffer ← boundendBuffer($k$)
3  for $r \in \mathbf{R}_s$ do
4      $pos_r^-, pos_r^+ \leftarrow$ position of $r(\theta)$ in $\mathbf{L}^r$
5  end
6  while $L <> \emptyset$ do
7      for $\mathbf{L}^r \in L$ do
8          $t \leftarrow$ AdvancePointer($\mathbf{L}^r$)
9          if $t =$ NIL then
10             $L \leftarrow L \setminus \{\mathbf{L}^r\}$
11         else
12             if $t \notin$ buffer then
13                 buffer.addWithPriority($t$, $\delta(t)$)
14             end
15             $\tau \leftarrow$ ComputeThreshold($L$)
16             if buffer.size() $= k$
                    and buffer.largestError() $\leq \tau$ then
17                 return buffer
18             end
19         end
20     end
21 end
22 return buffer
```

1. None of the two pointers reached the beginning or end of the list. In this case, the algorithm checks which pointer to advance (line 5). The pointer that is closer to $r(\theta)$ *after* advancing is moved by one position. In case of a tie, we arbitrarily decided to advance pos_r^+.

2. Only pos_r^- reached the beginning of the list: the algorithm increments pos_r^+ (line 11).

3. Only pos_r^+ reached the end of the list: the algorithm decrements pos_r^- (line 13).

4. The two pointers reached the beginning respective end of the list: no pointer is moved.

In the first three cases, the algorithm returns the timestamp that was discovered after advancing the pointer. In the last case, NIL is returned.

At the moment we use an in-memory implementation of the algorithm, which loads the whole data set into main memory. More specifically, we keep two copies of the data in memory: the data sorted by timestamp for fast random access and the data sorted by value and timestamp for fast sorted access.

Note that we did not normalize the raw data using some standard technique like the z-score normalization, as we cannot compute that efficiently for streams of data without increasing the complexity of our algorithm.

4.4 Proof of Correctness

The correctness of the Top-k Case Matching algorithm follows directly from the correctness of the *Threshold Algorithm*. What remains to be shown, however, is that the aggregation function $\delta(t)$ is monotone.

Algorithm 2: AdvancePointer

```
Data: List $\mathbf{L}^r$ where to advance a pointer
Result: Next timestamp to look at or NIL
pos ← NIL
if next($pos_r^+$) <> NIL and next($pos_r^-$) <> NIL then
    $\Delta^+ \leftarrow |r(\theta) - \texttt{value}(\mathbf{L}^r[\texttt{next}(pos_r^+)])|$
    $\Delta^- \leftarrow |r(\theta) - \texttt{value}(\mathbf{L}^r[\texttt{next}(pos_r^-)])|$
    if $\Delta^+ \leq \Delta^-$ then
        $pos, pos_r^+ \leftarrow$ next($pos_r^+$)
    else
        $pos, pos_r^- \leftarrow$ next($pos_r^-$)
    end
else if next($pos_r^+$) <> NIL and next($pos_r^-$) = NIL then
    $pos, pos_r^+ \leftarrow$ next($pos_r^+$)
else if next($pos_r^+$) = NIL and next($pos_r^-$) <> NIL then
    $pos, pos_r^- \leftarrow$ next($pos_r^-$)
end
if pos <> NIL then
    return timestamp($\mathbf{L}^r[pos]$)
else
    return NIL
end
```

THEOREM 4.1. *The aggregation function $\delta(t)$ is a monotonically increasing function.*

PROOF. Let t_1 and t_2 be two timestamps such that $|r(\theta) - r(t_1)| \leq |r(\theta) - r(t_2)|$ for each $r \in \mathbf{R}_s$. Then it trivially follows that $\delta(t_1) \leq \delta(t_2)$ as the aggregation function δ is the sum of $|r(\theta) - r(t_1)|$ over each $r \in \mathbf{R}_s$ and, by definition, each component of $\delta(t_1)$ is less than or equal to the corresponding component in $\delta(t_2)$. □

4.5 Theoretical Bounds

The space and runtime bounds of the algorithm follow directly from the probabilistic guarantees of TA, which has sub-linear cost with high probability and linear cost in the worst case. Note that sorting the raw data to build the lists $\mathbf{L}^r$ is a one-time preprocessing step with complexity $O(n \log n)$. After that the system can insert new measurements efficiently into the sorted lists with logarithmic cost.

5. EXPERIMENTAL EVALUATION

In this section, we present preliminary results of an experimental evaluation of the proposed Top-k Case Matching algorithm. First, we study the impact of parameter k on the Top-k Case Matching and a baseline algorithm. The baseline algorithm, referred to as "Simple Average", imputes the missing value $s(\theta)$ with the average of the values in the reference time series at time θ, i.e., $s(\theta) = \frac{1}{|\mathbf{R}_s|} \sum_{r \in \mathbf{R}_s} r(\theta)$. Second, we compare our solution with two state of the art competitors, REBOM [4] and CD [5].

5.1 Varying k

In this experiment, we study the impact of parameter k on the accuracy and the runtime of our algorithm. We picked five base stations distributed all over South Tyrol, each having two to five reference stations. We simulated a failure of the base station during a time interval, w, of 8 days in the month of April 2013. This amounts to a total of 11452 missing values. We then used the Top-k Case Matching (using both the average and median as aggregation function g) and Simple Average algorithms to impute the missing values. As a measure of accuracy we use the average absolute difference between the real value $s(\theta)$ and the imputed value $s^*(\theta)$, i.e., $\Delta = \frac{1}{|w|} \sum_{\theta \in w} |s(\theta) - s^*(\theta)|$

Figure 2 shows how the accuracy of the algorithms changes with varying k. Interestingly and somewhat unexpectedly, Δ *decreases* as k *increases*. This is somehow contrary to what we expected, since with an increasing k also the error function $\delta(t)$ grows, and therefore less similar historical situations are used for the imputation. However, after a careful analysis of the results it turned out that for low values of k the algorithm is more sensitive to outliers, and due to the often low quality of the raw data the imputation is flawed.

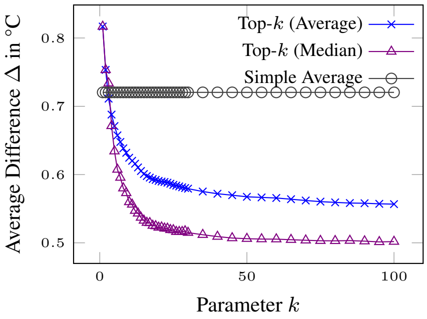

Figure 2: Impact of k on accuracy.

Table 5 shows an example of flawed raw data. The first row is the current situation, and we assume that the value in the gray box is missing and need to be recovered. The search for the $k = 3$ most similar situations using our algorithm yields the three rows at the bottom. Notice that one base station value is 39.9° around midnight of a day in August, which is obviously a very unlikely thing to happen. By increasing k, the impact of such outliers is reduced and hence Δ decreases. Furthermore, using the median as aggregation function reduces the impact of outliers and therefore yields better results than the average.

Timestamp	s	r_1	r_2	r_3
2013-04-16 19:35	18.399°	17.100°	19.293°	18.043°
2012-08-24 01:40	18.276°	17.111°	19.300°	18.017°
2004-09-29 15:50	19.644°	17.114°	19.259°	18.072°
2003-08-02 01:10	39.900°	17.100°	19.365°	18.065°

Table 5: Example of flawed raw data.

Figure 3 shows the runtime, which for the Top-k Case Matching algorithm linearly increases with k. Notice that, although the imputation of missing values for 8 days takes several minutes, the algorithm is fast enough to continuously impute missing values in our application at the *SBR*. The experiment essentially corresponds to a scenario, where in 11452 base stations an error occurs at the same time. With 120 weather stations operated by the *SBR*, the number of missing values at each time is only a tiny fraction of the missing values that we simulated in this experiment.

5.2 Comparison with CD and REBOM

In this experiment, we compare the Top-k Case Matching algorithm with two state-of-the-art algorithms, REBOM [4] and CD [5]. We used four time series, each containing 50.000 measurements, which corresponds roughly to half a year of temperature measurements. We simulated a week of missing values (i.e., 2017 measurements) in one time series and used the other three as reference time series for the imputation.

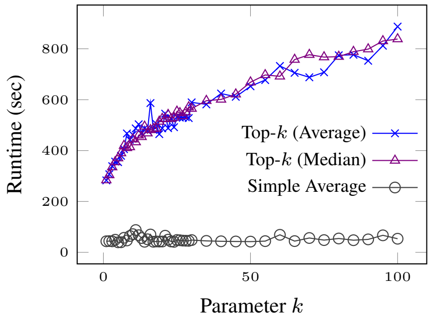

Figure 3: Impact of k on runtime.

The box plot in Figure 4 shows how the imputation error $|s(\theta) - s^*(\theta)|$ is distributed for each of the four algorithms. The left and right line of the box are the first and third quartile, respectively. The line inside the box denotes the median and the left and right whiskers are the 2.5% and 97.5% percentile, which means that the plot incorporates 95% of the values and omits statistical outliers. The experiment clearly shows that the Top-k Case Matching algorithm is able to impute the missing values more accurately than CD and REBOM. Although not visualized, also the maximum observed error for our algorithm is with 2.29° (Average) and 2.21° (Median) considerably lower than 3.71° for CD and 3.6° for REBOM.

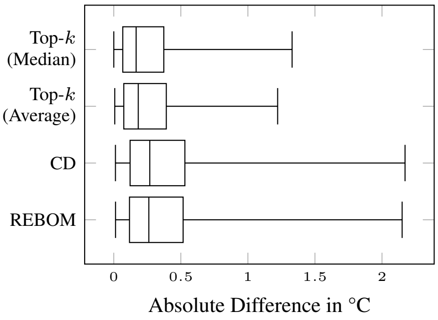

Figure 4: Comparison with REBOM and CD.

In terms of runtime, the Top-k Case Matching algorithm needed 16 seconds for the imputation of the 2017 missing measurements, whereas CD and REBOM needed roughly 10 minutes each. Note, however, that this large difference in run time is also due to the fact that CD and REBOM need to compute the Pearson correlation coefficient which is a time intensive operation.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented a simple yet efficient and accurate algorithm, termed Top-k Case Matching, for the imputation of missing values in time series data, where the time series are similar to each other. The basic idea of the algorithm is to look for the k situations in the historical data that are most similar to the current situation and to derive the missing values from the data at these time points. Our Top-k Case Matching algorithm is based on Fagin's Threshold Algorithm. We presented the results of a first experimental evaluation. The Top-k Case Matching algorithm achieves a high accuracy and outperforms two state of the art solutions both in terms of accuracy and runtime.

As next steps we will continue with the evaluation of the algorithm, taking into account also model based techniques such as DynaMMo [6] and other statistical approaches outlined in [9]. We will further study the impact of complex weather phenomena that we observed in our data, such as the foehn. The foehn induces shifting effects in the time series data, as the warm wind causes the temperature to increase rapidly by up to 15° as soon as the foehn reaches another station.

There are several possibilities to further improve the algorithm. First, we would like to explore whether the algorithm can dynamically determine an optimal value for the parameter k, which is currently given by the user. Second, we would like to make the algorithm more robust against outliers. For example, the algorithm could consider only historical situations that occur roughly at the same time of the day. Moreover, we can bend the definition of "current situation" to not only consider the current timestamp, but rather a small window of consecutive timestamps. This should make the ranking more robust against anomalies in the raw data and weather phenomena such as the foehn. Third, right now the similarity between time series is based solely on temperature data. We would like to include the other time series data collected by the weather stations, such as humidity, precipitation, wind, etc. Finally, the algorithm should be able to automatically choose the currently handpicked reference time series based on some similarity measures, such as the Pearson correlation coefficient.

7. ACKNOWLEDGEMENTS

The work has been done as part of the DASA project, which is funded by the Foundation of the Free University of Bozen-Bolzano. We wish to thank our partners at the Südtiroler Beratungsring and the Research Centre for Agriculture and Forestry Laimburg for the good collaboration and helpful domain insights they provided, in particular Armin Hofer, Martin Thalheimer, and Robert Wiedmer.

8. REFERENCES

[1] G. E. P. Box and G. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[2] R. Fagin. Combining fuzzy information from multiple systems (extended abstract). In *PODS'96*, pages 216–226, New York, NY, USA, 1996. ACM.

[3] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS '01*, pages 102–113, New York, NY, USA, 2001. ACM.

[4] M. Khayati and M. H. Böhlen. REBOM: recovery of blocks of missing values in time series. In *COMAD'12*, pages 44–55, 2012.

[5] M. Khayati, M. H. Böhlen, and J. Gamper. Memory-efficient centroid decomposition for long time series. In *ICDE'14*, pages 100–111, 2014.

[6] L. Li, J. McCann, N. S. Pollard, and C. Faloutsos. Dynammo: Mining and summarization of coevolving sequences with missing values. In *KDD'09*, pages 507–516, New York, NY, USA, 2009. ACM.

[7] A. Mueen, S. Nath, and J. Liu. Fast approximate correlation for massive time-series data. In *SIGMOD'10*, pages 171–182, New York, NY, USA, 2010. ACM.

[8] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB'04*, pages 648–659. VLDB Endowment, 2004.

[9] C. Yozgatligil, S. Aslan, C. Iyigun, and I. Batmaz. Comparison of missing value imputation methods in time series: the case of turkish meteorological data. *Theoretical and Applied Climatology*, 112(1-2):143–167, 2013.