Divide & Conquer Data Imputation for Time Series Gaps

Prathmesh Santosh Choudhari*, Andrew Rippy†
Department of Computer Science
*†University of Florida, Gainesville, USA
Email: ps.choudhari@ufl.edu, arippy@ufl.edu

***Abstract*—In large-scale IoT and sensor networks, sensors often produce incomplete time series due to power loss or network interruptions. This paper proposes a Divide & Conquer (D&C) approach to handle localized missingness by recursively splitting data into uniform segments, applying localized imputation, and merging results efficiently. This approach provides improved adaptability and computational efficiency over traditional global models.**

I. Real Problem Identification

Domain: Artificial Intelligence / Data Cleaning / Time Series Analysis

In large-scale IoT and sensor networks (such as air-quality monitoring or smart-grid systems), sensors often produce incomplete time series data due to communication failures, power loss, or environmental interference. These missing segments lead to unreliable analytics, poor forecasting, and incorrect anomaly detection.

Existing global imputation methods like ARIMA, KNN-Imputer, or deep LSTM models fail to handle localized missing patterns effectively and can be computationally expensive. Thus, a scalable and adaptive method is needed that focuses on **local missingness patterns** rather than fitting a single global model.

We propose a **Divide & Conquer (D&C)** approach to recursively split the time series based on uniformity of missingness, apply locally suitable imputation models, and then merge the reconstructed parts smoothly. This approach captures both local temporal behavior and global continuity while being computationally efficient.

II. Problem Abstraction

Let the time series be represented as:

$$S = \langle s_1, s_2, s_3, \dots, s_n \rangle, \quad s_i \in \mathbb{R} \cup \{\perp\}$$

where $\perp$ denotes missing values.

We abstract this as a **binary recursion tree** T:

- Each node v corresponds to a sub-sequence $S_v = S[a_v : b_v]$.
- Each node stores its **missing ratio**: $r_v = \frac{|I_v|}{b_v - a_v}$, where $I_v = \{i \mid s_i = \perp\}$, a, b are the start and end data point numbers of the segment. For example, if there were 6 data points, a would be 0, b would be 6.
- The node splits further if left and right sub-series differ in missingness beyond a threshold T.
- Leaf nodes represent "uniform" sub-series to be imputed locally.

Merging nodes corresponds to reconstructing the full imputed series in a hierarchical fashion.

III. Solution to the Abstract Problem

A. Algorithm

Algorithm: Divide & Conquer Imputation

```
Algorithm DnC_Impute(S, threshold T, model_selector M)

1. function BuildAndImpute(a, b):
2.     r = missing_ratio(S[a:b])
3.     if is_uniform(a, b, r, T) or (b-a+1) <= MIN_SEG:
4.         model = M(S[a:b])
5.         S_hat[a:b] = Impute_model(S[a:b], model)
6.         return S_hat[a:b]
7.     else:
8.         mid = (a+b)//2
9.         left  = BuildAndImpute(a, mid)
10.        right = BuildAndImpute(mid+1, b)
11.        merged = MergeSegments(left, right)
12.        return merged

13. return BuildAndImpute(1, n)
```

Explanation:

- Compute missing ratio in each segment.
- If left and right halves differ by more than threshold T, split further.
- Local imputation models are chosen adaptively:
  - Mean/median interpolation for small clean segments.
  - Linear regression for moderate-length gaps.
  - LSTM or AR model for longer missing blocks.
- Merged segments are smoothed via weighted overlap averaging to ensure continuity. For this implementation, we used mean/median interpolation for the segments.

B. Running Time Analysis

If the algorithm always splits evenly, we can express the recurrence as:

$$T(n) = 2T(n/2) + O(n)$$

Solving gives $T(n) = O(n \log n)$ for linear local models.

When deep models (e.g., LSTMs) are used only for large missing blocks:

$$O(n \log n) \le T(n) \le O(n \cdot C_{model})$$

where C_{model} is the training cost of the heaviest model.

Memory requirement is $O(n)$, and recursion depth is $O(\log n)$.

C. *Proof of Correctness (10 pts)*

We prove correctness via structural induction.

Base Case: For segment length $\leq$ MIN_SEGMENT, the function imputes all missing values directly using a local model.

Inductive Step: Assume all sub-segments shorter than m are fully imputed. For a segment of length m:

- It is uniform, hence imputed locally; or
- It is split into smaller sub-segments, each imputed by the inductive hypothesis.

Merged output uses overlap smoothing ensuring all indices are filled and continuity preserved.

Hence, by induction, every missing index in the series is imputed.

IV. Algorithm in the Problem Domain (5 pts)

In an IoT air-quality monitoring system:

- The algorithm identifies sensor downtime segments by recursively splitting the data where missing ratios differ substantially.
- For each downtime region, it selects an appropriate model — simple interpolation for small outages, or predictive learning (LSTM) for longer gaps.
- Each repaired window is merged back to reconstruct a full, continuous time series of PM2.5 readings.

V. Implementation and Experimental Verification

A. *Implementation*

- **Language:** Python
- **Libraries:** `pandas`, `numpy`, `scikit-learn`, `PyTorch`, `matplotlib`
- **Functions:**
  - **missing_ratio(series, start, end)** - Computes the fraction of missing values (NaNs) in a segment of the time series between indices start and end. Used for determining how "missing" each segment is, so the algorithm can decide whether to split further or stop.
  - **split_series(series, start, end, threshold, segments)** - Recursively divides the time series into subsegments based on how different the missing data ratios are between the left and right halves. Used for implementing the Divide step of the algorithm. If two halves have very different missingness (difference ¿ threshold), they are split again; otherwise, the segment is accepted as "uniform."
  - **divide_and_conquer_split(series, threshold=0.2)** - A simple wrapper around split_series() that initializes recursion and returns the final list of (start, end) index pairs for each segment. Used for providing a clean, high-level way to obtain all segment boundaries without manually setting up recursion.
  - **local_impute(segment, global_mean)** - Fills missing values (NaNs) within a given segment using linear interpolation between existing values. If all values are missing, it replaces them with the global mean of the entire series. Used for the Conquer step — locally reconstructing missing data within each uniform segment.
  - **divide_and_conquer_impute(series, threshold=0.2)** - Recursively splits the time series into uniform segments. Performs local imputation on each segment. Returns the fully reconstructed series and segment boundaries. Used for end-to-end execution of the full Divide & Conquer imputation algorithm.

B. *Datasets*

For this project, random data was generated using numpy and imputed using the algorithm.

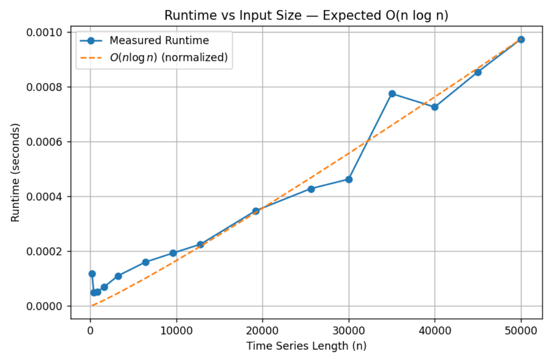

Fig. 1. Run time with predicted vs actual runtime.

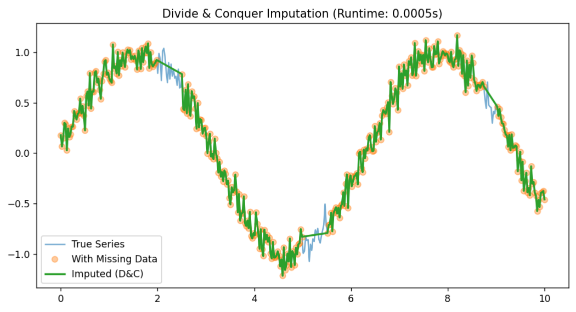

Fig. 2. Imputation with algorithm

VI. Example for Splitting and Threshold Intuition

Consider a time series:

$$[25.2, 26.1, 26.3, NaN, NaN, NaN, 27.0, 27.3, NaN, 28.0, 28.2]$$

Let threshold $T = 0.2$.

- Left half missing ratio = $3/6 = 0.5$
- Right half missing ratio = $1/5 = 0.2$

Difference $|0.5 - 0.2| = 0.3 > 0.2 \Rightarrow$ Split further.

Interpretation: Small T = more fine-grained splits (high adaptivity, more computation). Large T = fewer splits (faster, but less precise imputation).

VII. Conclusion

This Divide & Conquer Imputation algorithm provides:

- Adaptive imputation across heterogeneous missingness patterns.
- Reduced computation cost by avoiding global models.
- Hierarchical structure allowing parallel processing.

This algorithm gives an adaptable way to impute data, with room for growth and additional choices of imputation style. It solves an important problem in an efficient manner, and can be used on many different types of data generated by Iot sensors.

VIII. Appendix

The code used for this assignment is included in the analysis-of-algorithms-project-1.zip file, called divideandconquer.py. The github containing the code is at this link. https://github.com/rippy1849/analysis-of-algorithms-project-1

References

[1] ChatGPT https://chatgpt.com/c/6908ca2f-5568-832e-aff5-0cefa73f2f0f
[2] ChatGPT https://chatgpt.com/share/6908cdc3-039c-8010-93e3-f78b73062712
[3] Github Repository for Greedy and Divide and Conquer Project 1, Analysis of Algorithms https://github.com/rippy1849/analysis-of-algorithms-project-1