ICSI499 Capstone Project Report

PySpady Library Extension

Project Team

Michael Paglia (Student 1 Id)
Proshanto Dabnath (Student 2 Id)
Micheal Alexander Smith (Student 3 Id)
Joseph Regan (001442720)
.....
College of Engineering and Applied Sciences
University at Albany, SUNY

Project Sponsor

Petko Bogdanv
Data Mining and Management Lab
Department of Computer Science
1400 Washington Ave, UAB 416
Albany, NY 12222

01-05-2024

Acknowledgements

We collectively extend our heartfelt gratitude to Petko Bogdanov, our sponsor, and teacher, whose unwavering support, invaluable guidance, and dedication shaped the direction and quality of this Capstone Project. We are deeply thankful for Maxwell McNeil, for giving of himself so willingly that we might find it easier to make progress, and to Boya for being there and offering helpful insight when we got stuck. Additionally, we wish to express our profound appreciation to Pradeep Atrey, our advisor in this project. Prof. Atrey always made sure there was a clear and organized path to completion of this project. His support was unwavering, and invaluable throughout the duration of the project.

Their contributions, alongside the support of others, have been instrumental along the way of the successful completion of this Capstone Project, and we are profoundly grateful for the opportunity to have worked for and alongside such exceptional individuals.

Abstract

PySpady enables users from all disciplines to leverage state-of-the-art and classical sparse encoding algorithms and methodologies to jointly model spatial-temporal data by graph and temporal dictionaries. The current implementation efficiently exploits both structural graph regularities and temporal patterns encoded within 2D temporal graph signals (McNeil et. al, 2021) and more generally any multi-way tensors (McNeil and Bogdanov, 2023) with priors on all or a subset of modes. Users can input data in the form of either a JSON configuration file or a pandas DataFrame. This Python library is capable of performing missing value imputation, future value prediction, and downstream tasks such as outlier and community detection, along with generating visualizations and explanatory figures to derive meaningful insights. Perhaps most important is the ability to automatically configure an optimal combination of hyperparameters and dictionaries to perform the best reconstruction of a given matrix or tensor when considering missing entries. This exemplifies PySpady's accessibility and usability since prior knowledge of machine learning or statistical concepts is not required. Ample documentation and various demonstration datasets are provided to the user upon execution, including both synthetic and real-world examples. For instance, one of the datasets included in the library is based on New York City's taxi pickup and dropoff frequencies across nearly three hundred locations throughout 2017. PySpady's implementation provides an easily accessible and user-friendly solution to spatial-temporal data reconstruction along with interactive data visualization methods...

Contents

1 Problem Analysis

This section contains mostly what you did in Milestone 2, part 1, but possibly more detailed, as needed. Starting with answering the following questions (in ≈150 words):

- What problem you are trying to solve?
- Why this is an important problem?
- How is this a challenging problem to solve?

Further, answer the following questions (in ≈300 words):

- What has been done to solve the problem or what are the existing solutions?
- Why existing solutions are not sufficient or what are their key weaknesses and limitations?

Next, answer the following questions (≈100 words):

- What is your core idea and solution?
- What are the key characteristics of the proposed solution?
- How your solution is novel and better than existing solutions?

Finally, summarize the key contributions of the project, maybe in bullets (in ≈50 words). Don't make the reader guess. Be clear and explicit on what problem your system solves and why the reader should care. In the end, describe the organization of this report (in ≈50 words) (not the technical structure of the project).
Additional guidelines:

- Evidence (including statistics and background information) should be cited from credible, relevant sources. Here is an example citing something [**?**]. Here is an example, citing multiple references [**?**], [**?**]. Make sure that anytime you make a claim, the claim is supported with clear evidence.
- Summarize prior related work. You must have a reasonable number of citations.
- Identify gaps and limitations in prior work (particularly gaps your system/application addresses). Specifically, make sure to mention limitations that make existing solutions not useful for the problems mentioned in the abstract/introduction.
- The past works should not be discussed one by one, rather they should be linked with each other.
- Summarize the novelty and distinctiveness of the proposed work against existing work (explicitly in a table illustrating different aspects).

2 Proposed System/Application/Study

Length of this section should be 1000-1500 words.

2.1 Overview

(≈100 words)
Provide an overview of the proposed system/application/study. Also, state how this whole section is organized, and what the readers should expect next.

2.2 Project Requirements

(≈500 words)

2.2.1. User Classes

- Our users will include anyone in need of data analysis, particularly those requiring matrix decomposition, imputation of missing values, and related techniques. This could encompass scientists without a strong background in computer science, computer scientists, and computer hobbyists.

2.3 Functional Requirements

2.3.1. Data Management System

- The software will implement a data management system that allows the user to input data to run TGSD (Temporal Graph Signal Decomposition). Specifically, the end-user will be able to input their datasets as well as save any outputs.

2.3.2. Graphical User Interface (GUI)

- The software will implement a GUI accessible either through a web interface or a local interface, providing users with intuitive interaction and visualization capabilities.

2.3.3. TGSD

- The software will implement TGSD; a highly scalable decomposition algorithm for complete and incomplete data. The algorithm has advantages in matrix decomposition, imputation of missing values, temporal interpolation, clustering, period estimation, and rank estimation [3].

2.3.4. Auto-configuration

- The software will implement an auto-configuration tool that will have default parameters, an estimated time to complete, and return explanatory figures to the user. Dictionaries include Graph/Discrete Fourier Transform, Ramanujan, etc.

2.3.5. Outlier Detection

- The software will use TGSD to implement outlier detection. This functionality allows users to detect and identify anomalies within their datasets. Outlier detection plays a crucial role in various domains, including anomaly detection, fraud detection, and quality control, making it a valuable addition to the software's feature set.

2.4 Technical Design

(≈500 words)
Describe your technical solution. Include high-level drawings and illustrations that can help the reader under the system. Provide use-case diagrams, system architecture and data flow diagrams (Figure **??** and appropriately discuss them in the text. This is mostly what you did in Milestone 4, but possibly more detailed, as needed.

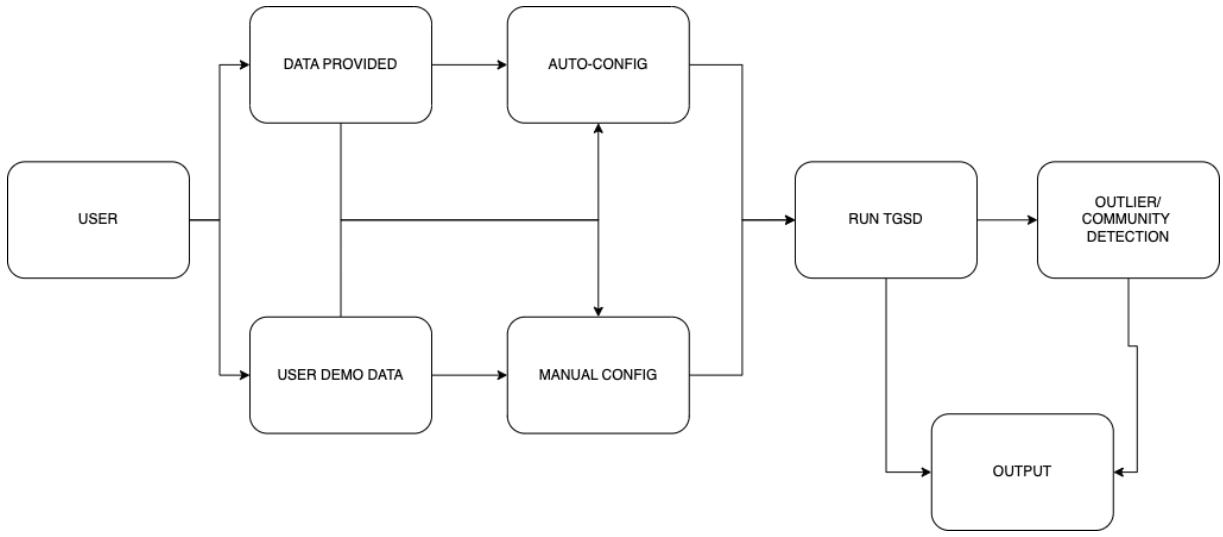


Figure 1: System Flow Diagram

2.5 System Implementation

(≈150 words)
Our team used a variety of development environments - some of us used PyCharm, which others

used VSCode. We handled our source/ version control using GitHub, where we maintained individual branches that we would merge with main when combining our code.

2.6 Use of Computer Science Theory and Software Development Fundamentals

(≈250 words)

In this section, you will discuss the use of computer science theories and software development fundamentals in your project. You are required to explicitly state three instances of application of computer science theories and software development fundamentals each and explain where and how exactly you have used it. Examples of computer science theories may include specific data structures or known algorithms that you may have used, computational complexity analysis of any subproblem that you may have solved, during the course of the project, and database design and normalization. Software development fundamentals may include efficient use of design patterns, modular and reusable coding practices, code documentation, unit testing, and integration tools. Please use the following template:

2.6.1. Use of Computer Science Theories

Some text...

Signal Processing Describe a specific instance of CS theory 1 here.

Data Mining Describe a specific instance of CS theory 1 here.

Sparse Decomposition Describe a specific instance of CS theory 2 here.

Graph Mining Describe a specific instance of CS theory 3 here.

2.6.2. Use of Software Development Fundamentals

Some text.

Source/Version Control Describe a specific instance of software development fundamental 1 here.

Modularity Describe a specific instance of software development fundamental 2 here.

Scalability Describe a specific instance of software development fundamental 3 here.

3 Experimental Design and Testing

In the following subsections, we provide a detailed discussion of our experimental setup, dataset, and results, along with an in-depth analysis of our findings.

3.1 Experimental Setup

The main objectives of the experiments are twofold: first, to validate the correctness of the PySpady library by comparing its output with the established MATLAB source code for matrix and tensor signal decomposition; and second, to evaluate the scalability and efficiency of the Python library as the input size grows exponentially. By conducting these experiments, we aim to demonstrate the reliability and performance of PySpady, ensuring its suitability for handling large-scale signal decomposition tasks.
A MacBook Air (M1, 2020) was used as the testing machine to conduct all experiments. The PySpady library was set up in a virtual environment, and the PyCharm IDE was employed for code development and execution.
It is important to note that Experiments #2 through #5 share the same input data. This approach ensures consistency and comparability across the different performance metrics of PySpady. The input data consists of synthetically generated temporal graph signals and adjacency matrices, created using a pre-existing MATLAB implementation for demonstration purposes.
Three matrix sizes were used as temporal graph signal inputs: a 100×100 matrix, a 500×500 matrix, and a 1000×1000 matrix. For each matrix of size $n \times t$, a Graph Fourier Transform (GFT) dictionary of size $n \times n$ and a Discrete Fourier Transform (DFT) dictionary of size $t \times t$ were employed for temporal graph signal decomposition. Additionally, for each matrix tested, rank values of $k \in \{5, 10, 20\}$ were utilized, such that the temporal graph signal reconstruction of $X \in R^{n \times t}$ can be approximated as $X \approx \Psi Y W \Phi$, where $\Psi \in R^{n \times m}$ is a fixed graph dictionary, $\Phi \in R^{s \times t}$ is a fixed temporal dictionary and $Y \in R^{m \times k}$ and $W \in R^{k \times s}$ are corresponding sparse encoding matrices.

3.1.1. Experiment #1: Verifying MATLAB vs. Python Output The primary objective of this experiment is to validate the correctness of PySpady's output by comparing it with the results obtained from the pre-existing MATLAB source code for matrix and tensor signal decomposition. The MATLAB code, which serves as a reference, is available at this link. This experiment was conducted as a prerequisite to the subsequent experiments, once a minimum viable product for PySpady was developed. The goal was to ensure that the reconstructed matrix/tensor's Euclidean norm matched the MATLAB and Python outputs.

3.1.2. Experiment #2: Runtime Tests vs. Nodes for Different Matrix Ranks This experiment aims to assess the runtime performance of PySpady across various numbers of nodes and matrix ranks, providing valuable insights into its scalability and efficiency. The primary objective is to determine whether the matrix decomposition runtime is adversely affected as the number of matrix elements increases exponentially.

3.1.3. Experiment #3: Memory Allocation vs. Nodes for Different Matrix Ranks This experiment focuses on assessing the memory allocation of PySpady across various numbers of nodes and matrix ranks, helping to understand its resource utilization. The goal was to determine whether or not the memory allocation of temporal graph signal X scaled as the number of matrix elements of X increased exponentially.

3.1.4. Experiment #4: RMSE vs. Nodes for Different Matrix Ranks This experiment aims to analyze the Root Mean Square Error (RMSE) of PySpady's output across different numbers of nodes and matrix ranks, providing a measure of its accuracy. The objective is to calculate the RMSE between the original temporal graph signal X and its reconstructed version obtained through PySpady's decomposition and reconstruction process. A lower RMSE indicates a higher fidelity between the original and reconstructed signals, demonstrating the library's ability to preserve the essential information present in the temporal graph signals.

3.1.5. Experiment #5: Number of Non-Zero Elements vs. Nodes for Different Matrix Ranks This experiment investigates the sparsity of the decomposed signals by examining the number of non-zero elements in the output across different numbers of nodes and matrix ranks. Sparsity is a desirable property in signal processing, as it allows for efficient storage, transmission, and computation of the decomposed signals. A lower number of non-zero elements, relative to the size of Y and W, indicate that the encoding matrices are sparser.

3.2 Dataset

The dataset used for all the experiments mentioned prior was synthetically generated using a pre-existing MATLAB library. This dataset is readily available within the PySpady library, allowing users to test the library's functionality with synthetic data immediately upon launch. Alternatively, the dataset can be downloaded externally from this link.
The synthetic temporal graph signal used in the experiments had an initial size of 175x200. However, to accommodate the testing of matrices with different dimensions, the signal was resized as needed. The input data was randomly generated and then augmented with a small amount of Gaussian noise for variance.
Throughout the experiments, the temporal graph signal decomposition process employed Graph Fourier Transform (GFT) and Discrete Fourier Transform (DFT) dictionaries. The

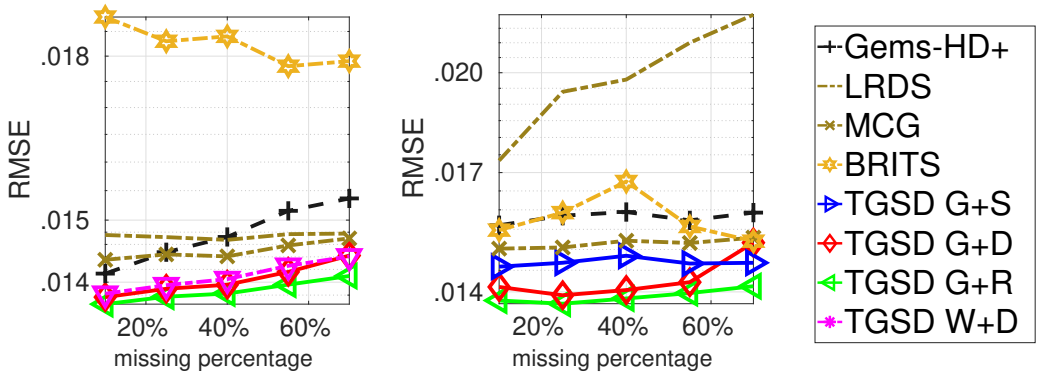
source code for generating these dictionaries was originally available in the previously linked MATLAB file but was then implemented and integrated into the PySpady library as well.

3.3 Results and Analysis

Table 1: Existing Solutions

Task	Baselines
Decomposition	MCG, LRDS, GEMS-HD
Imputation	MCG, LRDS, GEMS-HD, BRITS
Interpolation	MCG, LRDS, GEMS-HD, BRITS

McNeil et. al [3] compared the temporal graph signal decomposition algorithm's performance for matrix decomposition to three baselines: MCG [2], LRDS [4], and Gems-HD [5]. MCG and LRDS utilize rank minimization and graph regularization, while Gems-HD is a graph signal processing technique. As Gems-HD does not handle missing values directly, missing values were imputed prior; this approach can be denoted as Gems-HD+. BRITS [1] was also used to interpolate missing values in times series using a recurrent neural network.



(a) Synthetic-impute (b) Synthetic-interpolate (c) Legend

Figure 2: Comparison of quality for missing value imputation and interpolation [3]

As seen in Figure 2, McNeil et al. [3] compare the performance of TGSD with various baselines on synthetic data with strong periodicity. They find that the GFT + Ramanujan combination of TGSD outperforms all other methods, followed by MCG and LRDS, while Gems-HD+ and BRITS show degraded performance, especially when a large number of values are missing.

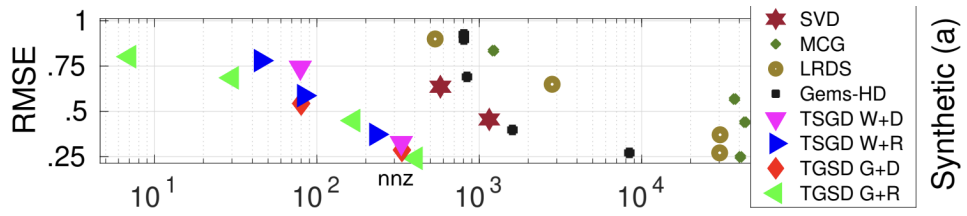


Figure 3: Decomposition quality as a function of model size [3]

In addition to the baselines provided, McNeil et. al. [3] use singular value decomposition (SVD) as a reference. They evaluate the effectiveness of their approach by measuring the root mean square error (RMSE) of the reconstructions as a function of the number of non-zero model coefficients (NNZ). As shown in Figure 3, TGSD variants outperform all baselines with a low NNZ, which is attributed to the joint encoding scheme that requires fewer coefficients to capture the signal trends effectively.

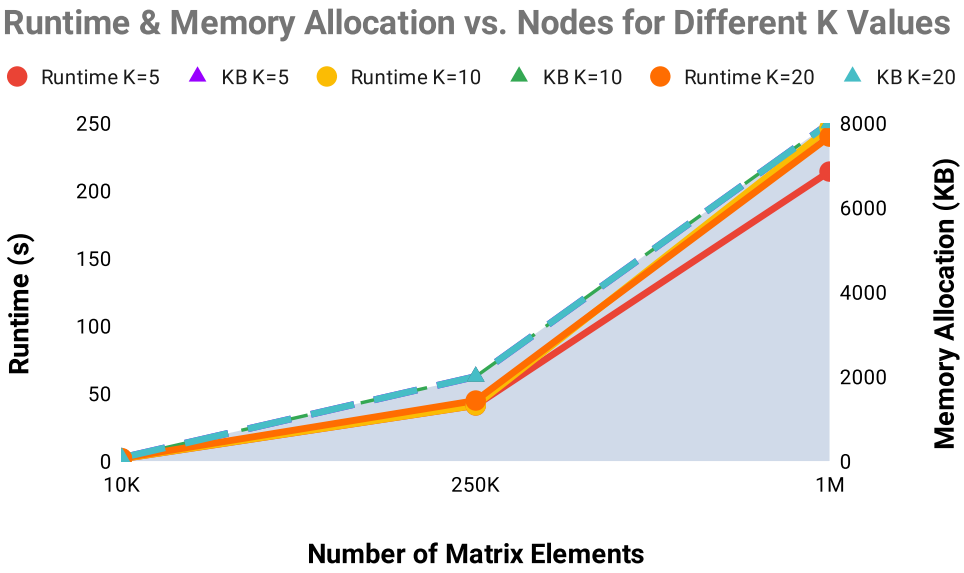


Figure 4: Runtime and memory allocation vs. no. matrix elements for different rank values.

PySpady was evaluated for runtime efficiency and memory allocation across varying rank values (K) and matrix sizes in Figure 4. The hypothesis was that a larger rank value would lead to a longer runtime and increased memory allocation due to the increased number of reconstructed atoms. For a matrix size of 10^4 elements, the runtime increased from 1.654s for K=5 to 2.084s for K=20, while the memory overhead remained constant at 8×10^{-5} bytes. Similarly, for 10^6 elements, the runtime increased from 214.1s for K=5 to 239.94s for K=20, with a constant memory overhead of 8×10^{-3} bytes. The runtime scaled well with increasing matrix size, from

1.654s for 10^4 elements to 214.1s for 10^6 elements at K=5. The memory allocation remained efficient, with only 8×10^{-3} bytes required for 10^6 elements across all tested rank values.

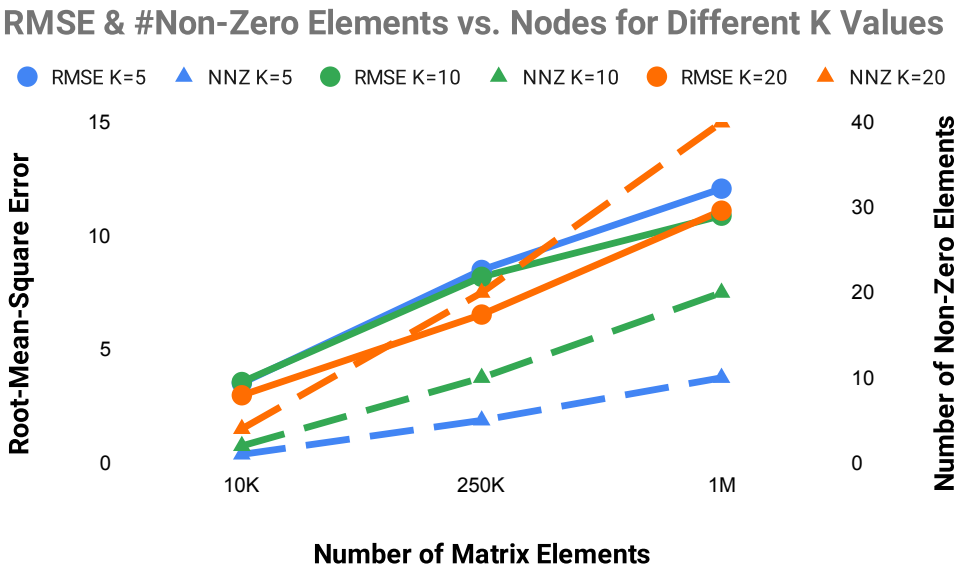


Figure 5: RMSE and no. non-zero elements vs. no. matrix elements for different rank values.

The performance of PySpady was evaluated using the root mean square error (RMSE) and the number of non-zero elements (nnz) across different rank values (K) and matrix sizes in Figure 5. For a matrix size of 10^4 elements, the RMSE decreased from 3.512 for K=5 to 2.98 for K=20, while the nnz increased from 1×10^3 to 4×10^4. A similar trend was observed for larger matrix sizes. For 10^6 elements, the RMSE decreased from 12.08 for K=5 to 11.11 for K=20, and the nnz increased from 1×10^4 to 4×10^4. The RMSE generally improved with increasing rank value, as higher rank values allow for more accurate approximations. However, this comes at the cost of a larger number of non-zero elements, which can impact storage requirements and computational efficiency, thus resulting in a less sparse decomposition.

3.3.1. Failure Cases Potential failure cases for PySpady include overfitting due to using a rank value (K) that is too large, underfitting when K is too small, and suboptimal results from improper hyperparameter selection. These issues can be mitigated through careful hyperparameter tuning, which is where PySpady's auto-configuration option may be useful. Additionally, performance problems may arise when working with extremely large datasets in the PyCharm IDE, leading to slowdowns or crashes.

4 Legal and Ethical Practices

(≈300 words)
Discuss the most important ethical and legal considerations that arose, and more importantly, how these were addressed in your project. These can vary dramatically from project to project. This section must include two subsections, as given below.

4.1 Legal Considerations

- Intellectual property concerns regarding the ownership of the underlying technology, as well as potential copyright or trademark concerns.

4.2 Ethical Considerations

- Informed consent regarding data leverage users or testers.
- Privacy concerns with your proposed system, including the use of personally identifiable data.
- Consideration and mitigation of any risks that might arise in the use of your proposed system.
- Intellectual property concerns regarding the ownership of the underlying technology, as well as potential copyright or trademark concerns.

5 Effort Sharing

(≈200 words)
Since it is a team-based project, team members must explicitly state their separate efforts/ contributions in addition to the joint efforts. This section must clearly state how different tasks of the project were divided into team members, and who did what (as shown in Table 2).

6 Conclusion and Future Work

(≈150 words)
State what you conclude from this work and what are the future work possibilities.

Provide below at least 10-15 references for your work.

Table 2: Effort sharing

Team size	Joint efforts	Member 1	Member 2	Member 3	Member 4	Member 5
5	J (≈25%)	I (≈15%)	I (≈15%)	I (≈15%)	I (≈15%)	I (≈15%)
4	J (≈30%)	I (≈17.5%)	I (≈17.5%)	I (≈17.5%)	I (≈17.5%)	-
3	J (≈31%)	I (≈23%)	I (≈23%)	I (≈23%)	-	-
2	J (≈40%)	I (≈30%)	I (≈30%)	-	-	-

J = description of tasks jointly performed
I = description of tasks individually performed

References

[1] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li. Brits: Bidirectional recurrent imputation for time series. In *Advances in Neural Information Processing Systems 31*, pages 6775–6785, 2018.

[2] V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst. Matrix completion on graphs. *arXiv preprint arXiv:1408.1717*, 2014.

[3] Maxwell McNeil, Lin Zhang, and Petko Bogdanov. Temporal graph signal decomposition. In *ACM International Conference on Knowledge Discovery and Data Mining*, Online, 2021.

[4] K. Qiu, X. Mao, X. Shen, X. Wang, T. Li, and Y. Gu. Time-varying graph signal reconstruction. *IEEE J. of Selected Topics in Signal Processing*, 11(6):870–883, 2017.

[5] Y. Yankelevsky and M. Elad. Finding gems: Multi-scale dictionaries for high-dimensional graph signals. *IEEE Transactions on Signal Processing*, 67(7):1889–1901, 2019.

*Appendix A: Resources [*Anything else you would like to provide...*]