SAGA: Signal-Aware Graph Aggregation

Maxwell McNeil* Boya Ma* Petko Bogdanov*

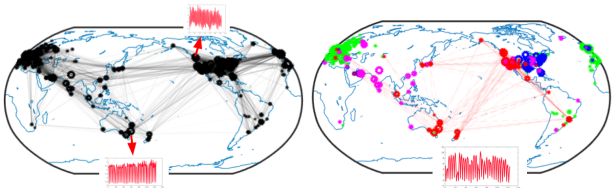
Abstract

Graphs are widely employed models for structural dependencies in complex systems such as social, infrastructure, and information networks. Graph datasets are often massive, computationally challenging to mine, and non-trivial to understand by humans. Hence, there is a large body of literature on graph summarization aiming to improve algorithmic efficiency, quality of analytics tasks, and visualization. Most existing graph summarization methods focus solely on the structure of the graph and assume that node properties are static. In this paper we focus on graphs with temporal measurements on their nodes, which we call temporal graph signals. Our goal is to learn a graph aggregation (summary) that best reflects both the structure and the temporal node measurements.

We propose a signal-aware graph aggregation framework called SAGA. The key idea is to group well-connected nodes whose behavior exhibits consistent temporal patterns. SAGA learns simultaneously how to (i) aggregate the graph into supernode groups and (ii) represent the groups' collective temporal behavior succinctly via a sparse dictionary encoding. The obtained aggregations offer insights into the functional organization of the graph and the learned model enables improved performance for state-of-the-art approaches for downstream tasks like temporal graph signal decomposition, forecasting and link prediction. We demonstrate, in both synthetic and real-world data sets, that SAGA's learned aggregations improve (i) reconstruction quality for temporal graph signals by up to 75%, (ii) link prediction accuracy by up to 40% and (iii) the accuracy of forecasting by up to 63% while also offering 50% scalability improvements.

1 Introduction

Data from many domains can be modeled as time series associated with nodes of a network, also called temporal graph signals (TGS). Examples include temporal readings from physical sensor networks [5, 1, 26], user activity in social media [8], and gene expression over protein interaction networks [3]. The underlying networks are often large (thousands to millions of nodes), making data mining and machine learning tasks in the original space challenging and impractical to interpret or visualize. Graph summarization offers an appealing solution by reducing the number of nodes to a small fraction of supernodes. Such summarization for TGS, however, needs to reflect both the temporal behavior of nodes and the graph structure, which have not been considered jointly to date.



(a) Airport network (b) Aggregate network

Figure 1: (a): The world airport direct-flight network (data from [26]), where each airport is associated with a time series of outgoing flights over time (Sydney and Vancouver time series plotted in red). (b): A summary aggregation in which airports of temporally consistent behavior and connected by direct flights are grouped into supernodes (denoted with shared color on the map). The edges of the mined Trans-Pacific (red) supernode and its learned summary time series is also shown.

The utility of graph aggregation as a summarization tool has been demonstrated for a variety of domains and downstream tasks [18]. Such aggregations offer a succinct high-level view of complex datasets by revealing the high-level graph properties while abstracting noisy or insignificant patterns. For example, Shen et Al. [30] used an aggregation to visualize a large and dense network, while Koutra and colleagues [15] utilized a summary to discover interesting patterns in real-world graphs such as edit wars in collaborative content creation. Existing graph aggregation approaches focus on the graph structure and node labels, but do not take into account temporal signals on the nodes. Graph signal processing offers an appealing alternative by modeling node values as a signal over the graph structure [27, 7]. Methods in this domain, however, focus on static (not temporal) graph signals, and do not produce a summary of the graph.

Consider as an example the global air-traffic network from [26] with airports as nodes, direct flights between them as edges, and a temporal signal reflect-

*Department of Computer Science, University at Albany—SUNY, Emails: {mmcneil2,bma,pbogdanov}@albany.edu

ing the number of outgoing flights from an airport over time. We visualize the network structure and show the flight time series of only two airports, Vancouver and Sydney, in Fig. 1(a). *The challenge is to distill the functional organization of this complex dataset corresponding to major geo-political transportation flows.* Fig. 1(b), presents a summary of the data produced by our proposed method, in which airports grouped in a supernode are marked with the same color. This grouping reflects both their coordinated temporal behavior (i.e., coherent outgoing flights time series) and their locality on the direct-flight network. Specifically, we detect supernodes corresponding to Trans-Atlantic, Trans-Pacific, and several other flows. We also show the aggregate learned temporal trend for all nodes in the Trans-pacific (red) supernode. Being able to automatically produce such intepretable summaries of desired spatial resolution for TGS datasets and employ them in downstream tasks is what motivates our work.

Our goal in this paper is to summarize TGS by aggregating nodes into groups of similar temporal behavior and close locality in the graph. To this end, we propose signal-aware graph aggregation (SAGA), a method to simultaneously learn an aggregation matrix for both the temporal signal and the graph. Our solution combines these two aims into a single optimization objective which encodes the aggregate node time series of a supernode via a temporal dictionary while enforcing smoothness of the grouping on the graph structure. To further expand its applicability, we equip SAGA with the ability to handle signals with partially observed values. We evaluate SAGA's utility as a preprocessing step for three temporal graph mining tasks including (i) data-driven dictionary learning for succinct TGS reconstruction, (ii) link prediction and (iii) future value forecasting. SAGA enables up to 75% reduction in representation error for temporal signal decomposition, up to 40% improvement in link prediction accuracy, and up to 63% reduction in mean square error for future value forecasting.

Our contributions in this paper are as follows:

- **Novelty:** SAGA is the first framework to tackle aggregation of temporal signals on graphs.
- **Applicability:** SAGA's aggregations are advantageous in multiple downstream applications for both improving the quality of results and reducing the computational runtime.
- **Interpretability:** SAGA summarizes a complex temporal graph signal through a small graph of supernodes and corresponding shared temporal trends capturing the functional organization of the network and offering an important analytics tool for practitioners.

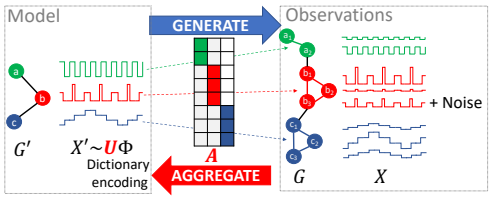


Figure 2: Temporal graph signal aggregation model via a small example. An unknown aggregated graph G' with structured pilot time series X' generate the observed graph and temporal graph signal (G, X). Our goal is to learn (i) the aggregation matrix A and (ii) the temporal structure of X' as a succinct dictionary encoding in U and a fixed dictionary Φ.

2 Problem formulation

The intuition behind our problem formulation is sketched in Fig. 2. The input to our problem is an observed graph G and a corresponding temporal graph signal (TGS) X (*Observations* panel in Fig. 2). We model (G, X) as generated from an aggregate-level graph G' with associated aggregate time series X' for each of its supernodes. The aggregate graph G' from the example in Fig. 2 has three nodes and we assume that their corresponding time series (rows of X') can be expressed succinctly using a dictionary encoding. For example, they might have pronounced periodicity or trend behavior and by employing an appropriate fixed dictionary Φ (e.g. DFT, Ramanujan or Spline) one can succinctly encode them using a coding matrix of coefficients U. The aggregate graph and time series give rise to the observations $(G', X') \rightarrow (G, X)$, where each supernode in G' "generates" a set of well-connected nodes in the observed graph (e.g., b generates a triangle (b_1, b_2, b_3) in Fig. 2). The time series of observed nodes follow a similar trend/periodicity to the corresponding supernode pilot time series. Our goal is to learn (i) an aggregation matrix A that maps observed nodes to supernodes and (ii) a sucinct representation of the temporal structure of the supernode time series X' via dictionary encoding by a matrix U and a fixed dictionary Φ. Next we formulate the problem and introduce necessary notation.

We represent the observed undirected graph G of n nodes by its weighted adjacency matrix $H \in \mathbb{R}^{n \times n}$ whose non-zero entries represent the strength of connections. The combinatorial graph Laplacian matrix L is defined as $L = F - H$, where F is a diagonal degree matrix with elements $F_{i,i} = \sum_q H_{i,j}$. A temporal graph signal (TGS) is a matrix $X \in \mathbb{R}^{n \times t}$, whose n rows contain node time series of length t time snapshots. Our goal is to learn an aggregate graph G' with k supernodes corresponding to disjoint groups of the original nodes such that members of a supernode exhibit shared temporal patterns and are of close proximity in G. Note that k is a user-controlled

desired summary size. We model the members of the i-th supernode as a one-hot encoding vector $\mathbf{a}_i \in \mathbb{R}^n$ with non-zero elements indicating participating nodes from G. We stack the k supernode vectors into the columns of an *aggregation matrix* $A = [\mathbf{a}_1^T, \mathbf{a}_2^T ... \mathbf{a}_k^T] \in \mathbb{R}^{n \times k}$.

Shared aggregate network locality. Supernodes give rise to groups of well-connected nodes in G. To promote high degree of connectivity among supernode constituents one can seek to minimize a Laplacian quadratic form $\mathbf{a}L\mathbf{a}^T$, where $\mathbf{a}$ is any of the supernode one-hot vectors. This quadratic form is widely adopted in graph signal processing to promote signal smoothness over the graph. Here we adopt it to promote "smoothness" or good connectivity for our aggregation.

Shared aggregate temporal behavior. We assume that nodes corresponding to a supernode have a shared temporal behavior "coordinated" by the aggregate time series of the supernode. In addition, we assume a simple and intepretable temporal structure for the aggregate time series in X', which we model via a sparse dictionary encoding. Specifically, if X'_i is the time series of the i-th supernode, we represent it as $X'_i \approx U_i\Phi$, where U_i is an encoding vector and Φ is a fixed dictionary matrix. For example, the periodic time series associated with supernode a in Fig. 2 can be expressed succinctly in frequency domain using a DFT dictionary.

Finally, the observed graph temporal signal X can be reconstructed via the aggregation matrix as follows:

$$X \approx AX' \approx AU\Phi, \tag{2.1}$$

where the non-zero elements in A's columns "replicate" and scale each aggregate time series to obtain the corresponding observations (e.g., the time series associated with a_1 and a_2 are scaled versions of the periodic time series of the supernode a in Fig. 2).

A non-negative orthonormal relaxation. Note that learning binary one-hot supernode encodings in A would require non-trivial combinatorial optimization. Particularly, if X is an all-constant time series, learning a smooth A over the Laplacian matrix is equivalent to minimizing the ratio cut formulation of spectral clustering which is an NP-hard problem [36]. Hence, we relax the one-hot encoding by promoting an orthonormal shape of A with non-negative elements. To avoid overfitting to noise and to control for the sparsity of our learned aggregation we also add $L1$ regularization for both A and U. Finally, since many real-world temporal graph signals may include missing values (due to corrupted data, faulty sensors, etc. [13, 24]), we also add a missing value mask Ω to ensure that our framework only fits observed values. Our overall constraint objective function is as follows:

$$\begin{aligned} \underset{A,U}{\operatorname{argmin}}\ & 1/2 \left\| \Omega \odot (X - AU\Phi) \right\|_F^2 + \lambda_0 \left\| A \right\|_1 \\ & + \lambda_1 \left\| U \right\|_1 + \lambda_2\ tr(A^T L A) \\ & s.t.\ A^T A = I, A > 0, \end{aligned} \tag{2.2}$$

where the four terms in the minimization model the aggregated temporal fit, sparsity in the aggregation matrix A, sparsity in the dictionary encodings of supernode time series U and smoothness of the aggregation on the Laplacian respectively. The regularization parameters λ_i control the importance of sparsity and smoothness, while the constraints ensure that A is non-negative and orthonormal. If there are no missing values in the input temporal graph signal X, one can simply omit the missing value mask Ω in the formulation.

3 Optimization solution: SAGA

We employ ADMM [6] to optimize our objective function from Eq. 2.2. We introduce intermediate variables $D = X$, $Z = A$ and $U = V$ which help ensure that all subproblems have a closed-form solution, resulting in the following objective:

$$\begin{aligned} \underset{A,U,V,Z,D}{\operatorname{argmin}}\ & 1/2 \left\| D - AU\Phi \right\|_F^2 + \lambda_0 \left\| Z \right\|_1 + \lambda_1 \left\| V \right\|_1 \\ & + \lambda_2\ tr(A^T L A) + 1/2 \left\| \Omega \odot (D - X) \right\|_F^2 \\ & s.t.\ U = V, Z = A, A^T A = I, A > 0 \end{aligned} \tag{3.3}$$

The Lagrangian corresponding to Eq. 3.3 is:

$$\mathcal{L} = \frac{1}{2} \left\| D - AU\Phi \right\|_F^2 + \lambda_0 \left\| Z \right\|_1 + \lambda_1 \left\| V \right\|_1 + \lambda_2\ tr(A^T L A) +$$

$$\frac{1}{2} \left\| \Omega \odot (D - X) \right\|_F^2 + \frac{\rho_1}{2} \left\| V - U + \frac{\Gamma_1}{\rho_1} \right\|_F^2 + \frac{\rho_0}{2} \left\| A - Z - \frac{\Gamma_0}{\rho_0} \right\|_F^2,$$

where ρ_1 and ρ_0 are penalty parameters and Γ_1 and Γ_0 are Lagrangian multipliers. We next derive solutions to individual subproblems of participating variables.

Updates for A: We let $U\Phi = B$ to obtain A's optimization subproblem:

$$\underset{A}{\operatorname{argmin}} \frac{1}{2} \left\| D - AB \right\|_F^2 + \lambda_2\ tr(A^T L A) + \frac{\rho_0}{2} \left\| A - Z + \frac{\Gamma_0}{\rho_0} \right\|_F^2$$

Setting the gradient to zero we obtain:

$$-DB^T + ABB^T + \lambda_2 L^T A + \lambda_2 LA + \rho_0 A - \rho_0 Z + \Gamma_0 = 0,$$

which due to L's symmetry can be reorganized in the form of the Sylvester equation:

$$\underbrace{2\lambda_2 L}_{L_h} A + A \underbrace{(\rho_0 I + BB^T)}_{R_h} = \underbrace{DB^T + \rho_0 Z - \Gamma_0}_{C} \tag{3.4}$$

Since both L_h and R_h are symmetric in the above we have a fast solution based on their eigendecompositions: $L_h = Q_A diag(\mathbf{d}_A) Q_A^T$ and $R_h = Q_B diag(\mathbf{d}_B) Q_B^T$ due to [4]. Namely, the solution for A is as follows:

$$A = Q_A [(Q_A^T C Q_B) \oslash (\mathbf{d}_A + \mathbf{d}_B^T)] Q_B^T, \tag{3.5}$$

where $\oslash$ denotes element-wise division. Since L_h is a constant matrix, we can further speed-up our algorithm by pre-computing its eigendecomposition. To satisfy the constraints for A, we (i) set negative values in the solution from Eq. 3.5 to 0 and (ii) ensure orthogonality by utilizing a solution to orthogonal procrustes [35] by

Algorithm 1 SAGA

Input: Input X, L, Ω, dictionary $\{\Phi\}$, k, $\lambda_0, \lambda_1, \lambda_2$

1: Initialize $A = Z = \mathbf{1}$, $U = V = \mathbf{1}$, $Q_A diag(\mathbf{d}_A) Q_A^T = (2\lambda_1 L)$
2: **while** not converged **do**
3: $B = U\Phi$
4: $C = DB^T + \rho_0 Z - \Gamma_0$
5: $Q_B diag(\mathbf{d}_B) Q_B^T = (\rho_0 I + BB^T)$
6: $A = Q_A[(Q_A^T C Q_B) \oslash (\mathbf{d}_A + \mathbf{d}_B^T)]Q_B^T$
7: $A = A \geq 0$
8: $\Upsilon \Sigma \Psi^T = svd(A)$
9: $A = \Upsilon I \Psi^T$
10: $U = (2A^T D\Phi^T + \rho_1 V - \Gamma_1)(2\Phi\Phi^T + I\rho_1)^{-1}$
11: $V_{ij} = sign\,(H_{ij}) \times max\left(|H_{ij}| - \frac{\lambda_1}{\rho_1}, 0\right)$
12: $Z_{ij} = sign\,(H_{ij}) \times max\left(|H_{ij}| - \frac{\lambda_0}{\rho_0}, 0\right)$
13: $D = (P + \Omega \odot X) \oslash (I + \Omega)$.
14: $\Gamma_0^{i+1} = \Gamma_0^i + \rho_0\,(Z - A)$
15: $\Gamma_1^{i+1} = \Gamma_1^i + \rho_1\,(V - U)$
16: $i \leftarrow i + 1$
17: Convergence condition: $\left|f^{i+1} - f^i\right| \leq \varepsilon$, where f^{i+1} and f^i are the objective values of Eq. 2.2 at iterations $i+1$ and i.
18: **end while**

applying SVD to the resulting A as follows:

$$\Upsilon \Sigma \Psi^T = A \rightarrow A = \Upsilon I \Psi^T, \tag{3.6}$$

where I is the identity matrix.

Updates for U: We have the following optimization problem for U:

$$\underset{U}{\text{argmin}}\ \frac{1}{2}\|D - AU\Phi\Big\|_F^2 + \frac{\rho_1}{2}\left\|U - V + \frac{\Gamma_1}{\rho_1}\right\|_F^2 \tag{3.7}$$

Setting its gradient w.r.t. U to 0, we obtain:

$$(2A^T D\Phi^T + \rho_1 V - \Gamma_1)(2\Phi\Phi^T + I\rho_1)^{-1} = U \tag{3.8}$$

When Φ is orthonormal this simplifies further to:

$$(2A^T D\Phi^T + \rho_1 V - \Gamma_1)[(2 + \rho_1)I]^{-1} = U \tag{3.9}$$

Update for D: By letting $P = AU\Phi$, we have the following optimization problem for D:

$$\underset{D}{\text{argmin}}\ \frac{1}{2}\|D - P\|_F^2 + \frac{1}{2}\|\Omega \odot (D - X)\|_F^2 \tag{3.10}$$

Setting its gradient to zero, we get:

$$D = (P + \Omega \odot X) \oslash (I + \Omega). \tag{3.11}$$

Updates for Z, V: The problems w.r.t. Z and V are:

$$\begin{cases} \underset{Z}{\text{argmin}}\ \lambda_2 \|Z\|_1 + \frac{\rho_0}{2}\left\|Z - A + \frac{\Gamma_0}{\rho_0}\right\|_F^2 \\ \underset{V}{\text{argmin}}\ \lambda_1 \|V\|_1 + \frac{\rho_1}{2}\left\|U - V + \frac{\Gamma_1}{\rho_1}\right\|_F^2 \end{cases} \tag{3.12}$$

Closed-form solutions are available due to [17]:

$$\begin{cases} Z_{ij} = sign\left(H_{ij}^{(2)}\right) \times max\left(\left|H_{ij}^{(2)}\right| - \frac{\lambda_0}{\rho_0}, 0\right), \\ V_{ij} = sign\left(H_{ij}^{(1)}\right) \times max\left(\left|H_{ij}^{(1)}\right| - \frac{\lambda_1}{\rho_1}, 0\right), \end{cases} \tag{3.13}$$

where $H^{(1)} = U - \frac{\Gamma_1}{\rho_1}$ and $H^{(2)} = A - \frac{\Gamma_0}{\rho_0}$.

Updates for Γ_i : The Lagrangian multipliers are updated as follows:

$$\Gamma_1^{i+1} = \Gamma_1^i + \rho_1(V - U) \ \text{ and } \ \Gamma_0^{i+1} = \Gamma_0^i + \rho_0(Z - A) \tag{3.14}$$

The overall SAGA algorithm. We list all updates in the overall optimization procedure in Alg. 1. We initialize all variables and precompute the evd of $2\lambda_1 L$ in step 1 and repeat updates in steps 3-16 until convergence. The three steps dominating the runtime of SAGA are (i) the eigenvalue decomposition in step 5 which is in the worst case $O(k^3)$, (ii) the singular value decomposition in step 8 with a worst case complexity $O(nk^2)$, and (iii) the matrix inversion in step 10, which is either $O(p^3)$ for non-orthogonal dictionaries Φ, where p is the number of dictionary atoms, or $O(p)$ for orthogonal dictionaries due to the simplified version in Eq. 3.9.

Dataset	Nodes	Edges	t	Resolution
Synthetic	150-50k	30k-1500	1000	NA
Air [26]	389	27073	124	6 hour
Bike [1]	142	3446	328	1 day
Road [5]	1923	5318	720	1 hour
Reality Mining [8]	94	795	8636	1 hour

Table 1: Summary of datasets used for evaluation.

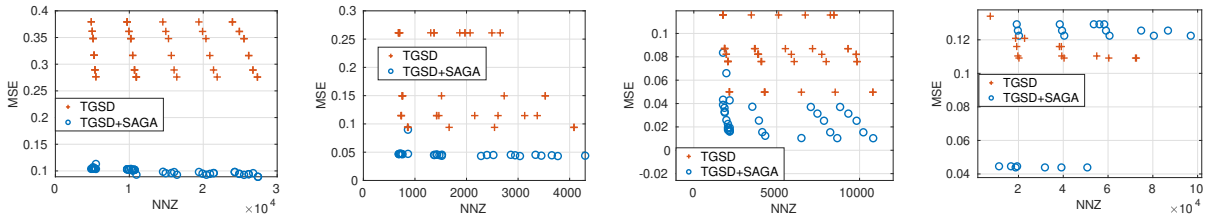
Dictionaries for SAGA. SAGA can accommodate arbitrary temporal dictionaries Φ. In our experiments we utilize three popular alternatives: *DFT (D)* and *Ramanujan (R)* for periodic signals and the *Spline (S)* dictionary for signals with simple smooth trends. Definitions and further details about the dictionaries are available in the supplement.

Tuning SAGA's hyper-parameters. In order to tune hyper-parameters one can adopt a task-specific grid search like as have done for this paper (details in the supplement). However, recent work on TGS [24] offer a more general solution. Specifically, hyper-parameters are tuned using cross-validation for reconstruction by (i) removing random values form the TGS, (ii) imputing the missing values given one setting of hyper-parameters, (iii) measuring the accuracy of imputation, and (iv) repeating (i)-(iii) for a new set of parameters. The setting of lowest imputation error is considered as optimal. SAGA can leverage this existing method by replacing the graph dictionary with our aggregation A (as we do in Sec. 4.2) and repeating the same process to set SAGA's hyper-parameters.

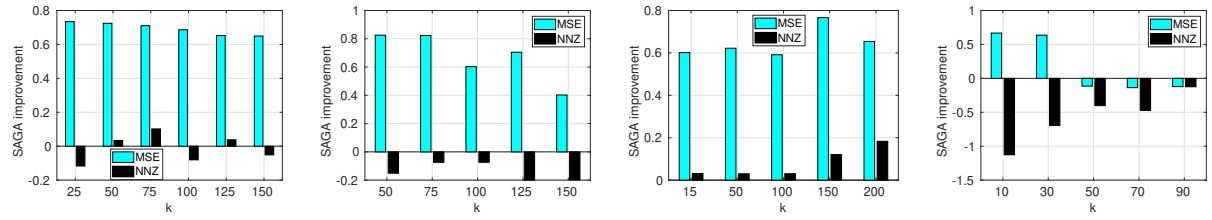
4 Experimental evaluation

We evaluate the quality of SAGA's learned aggregation on three different tasks: data-driven graph dictionary learning for compressed TGS representation, link prediction, and forecasting on four datasets. In all tasks SAGA serves as a preprocessing step that summarizes the dataset and enables quality and running time improvements for subsequent state-of-the-art techniques on the tasks.

4.1 Data sets. We employ four real-world and a series of synthetic datasets in our experiments summarized in Tbl. 1 and further described in the supplement. We generate *Synthetic* data by creating a stochastic block model graphs with high edge probability between members of the same supernode. The corresponding TGS signal is generated based on a vector auto regres-



(a) Synthetic: MSE v.s. size (b) Air: MSE v.s. size (c) Bike: MSE v.s. size (d) Road: MSE v.s. size



(e) Synthetic improvement (f) Air improvement (g) Bike improvement (h) Road improvement

Figure 3: Decomposition quality vs model size, comparing k smallest frequencies in the GFT vs k supernodes found by SAGA. Fig.3(a), 3(b), 3(c), and 3(d) are scatters of the Pareto optimal points for a given k. Fig.3(e), 3(f), 3(g), and 3(h) are the bar plots of the average relative improvement for a given k of SAGA in terms of MSE and NNZ. SAGA utilized the Spline dictionary in *Bike* and Ramanjuan in the rest. TGSD always utilizes the DFT dictionary.

sion (VAR) model associating members with in the same node. The real-world data-sets include *Air*[26] (airport traffic), *Road*[5] (road speeds), and *Bike*[1] (bike rentals) employed for graph dictionary learning for reconstruction, link prediction, and forecasting; *Reality Mining (RM)* [8] dataset is employed for link prediction experiment to evaluate SAGA on a social interaction dataset. Further details of each dataset are provided in the supplement.

4.2 Dictionary Learning. To demonstrate that SAGA is able to find an aggregation which can be used to succinctly represent a temporal graph signal we employ the Temporal Graph Signal Decomposition (TGSD) method, a state-of-the-art representation method for data of this type [24]. TGSD represents a TGS as a low-rank joint encoding via a graph dictionary (Ψ) and temporal dictionary (Φ), such that $D \approx \Psi Y W \Phi$. As we discuss in Related Work the utilization of such graph dictionaries is central to many graph signal processing methods [27].

In this experiment we demonstrate that using our aggregation matrix in place of a "band-limited" Graph Fourier Transform (GFT) dictionary (Ψ in TGSD) results in better-quality representation given a fixed number of dictionary atoms (columns in Ψ). In order to preform this evaluation we repeat the graph signal decomposition experiment from [24] which varies TGSD's parameters and records the number of nonzero coefficients (NNZ) learned and the representation error measured in terms of mean squared error (MSE). One variation of TGSD employs the standard GFT dictionary with up to k columns (denoted TGSD in figures), while the second variation employs the learned aggregation matrix A as a graph dictionary of the same size (denoted TGSD+SAGA). Similar to TGSD's original experiment we zscore-normalize the data by subtracting the mean and dividing by the variance.

We conduct this experiment on *Synthetic*, *Air*, *Bike*, and *Road*, and plot the Pareto-optimal models for a given k and the relative improvement for that k in Fig. 3. SAGA's learned data-driven graph dictionary in A provides significant advantages in *Synthetic*, *Air*, and *Bike* reducing the MSE by at least of 40% while minimally increasing the NNZ by at most 20% and on average nearly no increase. SAGA considers jointly the network and the temporal information to learn a good organization of the original nodes into supernodes (dictionary atoms). Thus, even if the input graph does not align well with the temporal domain across all nodes, SAGA can still find supernodes that can be utilized to represent a TGS succinctly. In contrast, the GFT can only utilize the input graph information to encode the signal even when it "does not align" perfectly with the temporal behavior. In the *Road* dataset SAGA is able to reduce the MSE at small dictionary sizes k at the cost of a higher number of NNZ coefficients. The road speeds on neighboring nodes (highway sections) are well-coupled, making the *Road* graph highly well-aligned to the temporal behavior and leaving limited room for improvement by data-driven dictionaries.

4.3 Link Prediction. The A matrix in SAGA encodes temporally-coherent groupings of nodes. We next

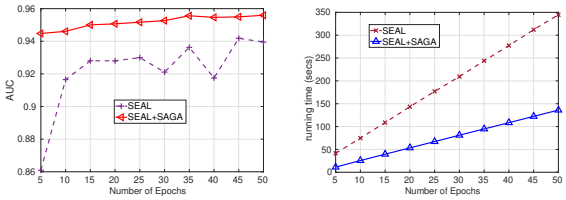
Data	Observed $\|E\|$	Vanilla		SAGA R		SAGA S	
		Acc	AUC	Acc	AUC	Acc	AUC
RM	90%	**90.3**	95.7	88.1	**96.4**	88.6	96.3
	70%	88.3	95.6	88.7	95.3	**88.8**	**95.7**
	50%	87.3	93.3	88.0	94.4	**88.1**	**94.8**
	30%	84.6	90.8	**85.4**	91.9	85.2	**92.0**
Air	90%	85.4	93.7	85.9	**94.2**	**86.4**	**94.2**
	70%	84.8	93.0	85.8	**93.9**	**86.0**	**93.9**
	50%	84.4	92.4	85.2	**93.3**	**85.2**	**93.3**
	30%	82.5	90.8	84.1	**92.3**	**84.0**	**92.3**
Bike	90%	95.5	88.5	99.4	96.1	**96.2**	**99.5**
	70%	87.3	95.2	95.9	99.4	**96.0**	**99.4**
	50%	85.4	93.3	94.6	98.9	**94.7**	**98.9**
	30%	83.4	91.2	93.5	98.4	**93.7**	**98.4**
Road	90%	90.1	95.6	**99.5**	**99.8**	98.8	99.8
	70%	75.1	84.1	**98.2**	**99.6**	97.3	99.1
	50%	67.0	73.6	**97.6**	**99.0**	95.8	97.8
	30%	55.7	59.0	**96.4**	**98.4**	90.0	94.2

Table 2: Accuracy (Acc) and area under the curve (AUC) comparison for link prediction using SEAL [37] given raw node features (Vanilla) and SEAL when using the aggregations produced by SAGA with the Ramanujan (SAGA R) and Spline (SAGA S) dictionaries for various percentages of observed edges

set out to quantify their utility for predicting missing links by utilizing SEAL [37], a state-of-the-art graph neural network link predictor. To predict a link between nodes a and b, SEAL first extracts enclosing subgraphs for a and b and then estimates their proximity based on label and structural information. Finally, it utilizes these features and the given adjacency matrix in a graph neural network (DGCNN [38]) to predict the probability of edges between the two nodes. It is trained on both negative (nonexistent) and positive edges. We utilize SAGA to improve SEAL by employing the learned aggregation matrix A as node attributes.

We compare the accuracy and area under the curve (AUC) of SEAL+SAGA versus the *Vanilla* SEAL method. We follow and extend the experimental design from [37]. We remove a set percentage of the known links in the graph and save them for testing while sampling an equal number of negative links for the test set. We utilize the remaining number of positive links while again sampling an equal number of negative links to produce the training set. The authors of [37] utilized 90% of the positive links for training and 10% for testing. However, we are also interested in the trend as smaller number of edges are observed. Thus, we perform experiments at other levels of missing links (Column "Observed $|E|$ in Tbl 2). We report the average of 5 samples per dataset and remove test edges before we learn SAGA's model to ensure we have no direct information on their existence.

The accuracy and AUC comparison is summarized in Tbl. 2. In every setting and dataset a variation of SEAL+SAGA dominates Vanilla SEAL with the exception of the accuracy in *Reality Mining* at 90% observed edges. SAGA is able to place nodes into distinct communities which have similar temporal patterns and network neighborhoods. The connection between this community information and the probability of an edge between two nodes can then be easily learned by a link prediction method like SEAL.



(a) Epochs v.s. AUC (b) Epochs v.s. runtime

Figure 4: AUC and running time comparison between SEAL and SEAL+SAGA on the RM dataset.

We are also interested in characterizing how SAGA affects the convergence and running time of SEAL and compare the running time and AUC of SEAL and SEAL+SAGA with the Ramanujan dictionary at different epochs of training for the GNN on the *RM* (Fig.4) and the *Road* datasets (figures in supplement). In both datasets SEAL+SAGA converges to a better result faster than vanilla SEAL. SAGA learns a holistic (temporal behavior and structure) representation for graph groups which leads to stable convergence with much less variation between epochs.

4.4 Forecasting. Multivariate time series forecasters typically utilize historical data in order to learn to predict future values [23]. However, not all time series will have predictive interactions. If we can correctly partition nodes into groups retaining predictive interactions, we can reduce the computational complexity and robustness of predictors due to learning fewer parameters. SAGA is particularly well suited for this task, as we partition nodes into supernodes which are both temporally- and network-aware. To test this, we utilize VAR [28] to predict future values 10 steps ahead in a sliding window fashion on the *Air*, *Bike*, *Road* and *Synthetic* datasets (10 independent samples for the latter). We compare the forecasting quality of group-wise VAR predictors, where groups are detected by SAGA and baselines partitioning algorithms: time series k-means [21], spectral clustering on the graph [12], and a stat-of-the-art clustering method for time series on graphs CCTN [19].

We also evaluate the corresponding quality when 25% of the training values are missing values. We use neighbor averages to impute missing values for baselines, while SAGA handles those directly through the mask Ω. One exception is the *Road* dataset where we also use the neighbor imputation method for SAGA.

Dataset	missing	SAGA				CCTN[19]			k-means			Spectral			VAR		
	%	raw MSE	num of groups	max time	Dic	raw MSE	num of groups	max time	raw MSE	num of groups	max time	raw MSE	num of groups	max time	raw MSE	num of nodes	max time
Syn	0	**0.24**	15	0.10s	R	0.47	15	1.33s	0.55	15	1.27s	0.25	15	**0.05s**	0.33	430	1.52s
Syn	25	**0.36**	15	0.11s	R	0.95	15	1.28s	0.92	15	1.26s	0.40	15	**0.06s**	0.63	430	1.51s
Air	0	91.1	100	**0.004s**	R	6011.0	5	0.08s	414.5	5	0.07s	110.0	5	0.05s	**89.6**	389	0.08s
Air	25	**339.5**	100	**0.003s**	R	$7.6 * 10^5$	5	0.07s	1049.2	5	0.06s	459.7	5	0.05s	397	389	0.08s
Bike	0	**607.5**	25	**0.012s**	S	2732.9	5	0.084s	655.0	5	0.020s	708.6	5	0.015s	1663.0	142	0.115s
Bike	25	**1034.5**	25	0.020s	S	3144.7	5	0.092s	1294.7	5	0.023s	1095.0	5	**0.018s**	2096.2	142	0.127s
Road	0	43.5	2	**2.8s**	S	43.9	100	4.8s	44.1	2	3.4s	42.5	2	3.7s	**38.7**	1923	5.7s
Road	25	45.9	2	**2.3s**	S	47.0	100	4.8s	48.4	2	3.6s	46.6	2	3.7s	**43.0**	1923	5.7s

Table 3: Comparison vector auto-regression (VAR) forecasting quality and running time for groups identified by SAGA and alternatives. of (i) MSE forecasting of the next 10 time steps, maximum running time, relative to a VAR on entire dataset.

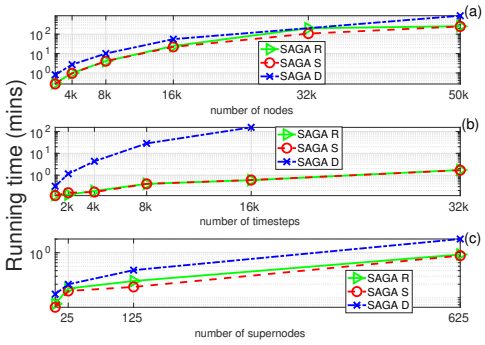


Figure 5: Running time for SAGA as a function of (a): the number of nodes n ($t = 1000, k = 5$), (b): the number of time steps t ($n = 1000, k = 5$), and (c): the number of supernodes k ($n = 1000, t = 1000$).

We search various parameters for all baselines and report the best obtained quality (details of parameter tuning are available in the supplement).

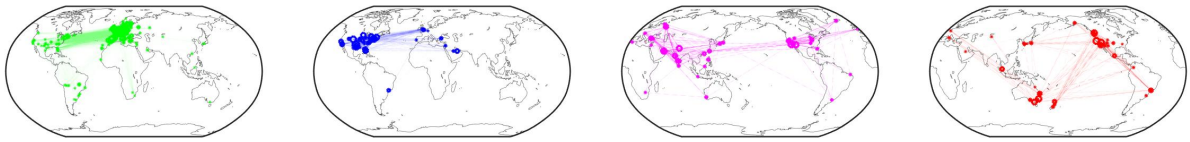
We present the MSE of forecasters and the maximum VAR training time for any partition in Tbl .3. SAGA produces the most accurate forecast in *Synthetic* and *Bike*, both with and without missing values present. In *Air* without missing values and in *Road* SAGA is a close second in terms of MSE after the full VAR model on the whole dataset, but enables a running time reduction to a fraction of full VAR's running time.

4.5 Scalability. We have already demonstrated that SAGA has the ability to speed up downstream applications, but a natural question remains: How long does it take to learn SAGA's aggregation? We employ *Synthetic* datasets of varying number of nodes n, time steps t, and supernodes k and record their effect on the running time (Fig. 5). All dictionary variations of SAGA scale similarly with n (Fig.5(a)) since the temporal dictionary encoding does not depend on the number of nodes. In Fig.5(b) we see a stark difference between the performances of different dictionaries since the full DFT is quadratic in t, while for Ramanujan and Spline dictionaries we use a constant number p of atoms w.r.t. t. Note that the DFT dictionary can also be band-limited in terms of frequency to enable faster model training. Finally, in Fig.5(c) we see increases in running time with the number of supernodes as expected based on our complexity analysis. It is important to note that for significantly large instances, SAGA can be trained within minutes to an hour, making it practical and applicable.

4.6 Case Study: Airport traffic data. SAGA's learned aggregation produces meaningful and interpretable supernodes which can be used to obtain a high level understanding of the functional organization of temporal graph signals. We look closer at the results on the *Air* data from [26] and plot the airport locations and flight links within 4 discovered supernodes in Fig. 6. For this analysis we employed the Ramanujan periodic dictionary with $\lambda_0 = .01$, $\lambda_1 = .01$ and $\lambda_2 = .1$. Each airport is represented by a circle of size proportional to the number of its incoming flights. The 4 supernodes show interpretable air traffic flows. We give each a name that aligns with our interpretation in Fig. 6. This interpretation is supported by the dominant airlines operating in each supernode. For example, the Europe-North America supernode is dominated by airlines such as Ryanair (Ireland), and Lufthansa (Germany) while the Trans-Pacific one is dominated by Southwest (US west), and Virgin Australia (we include a table of top airlines per supernode in the supplement).

5 Related work

Graph summarization seeks to improve analytics efficiency, pattern discovery, anomaly detection, and visualization for complex graph datasets [18]. Many methods reduce the input graph by aggregating nodes into supernodes by reflecting the community structure and node attribute consistency [34, 16]. Labels are typically discrete values [34] or real scalars [39], Our setting can be viewed as a generalization of label-aware summarization where node "labels" are time series. Dynamic graph structure summarization has also been considered [29], however, it works with edge changes



(a) Europe-North America (b) North America (c) Asia-North America (d) Trans-Pacific

Figure 6: Supernodes learned from the *Air* traffic dataset including the comprising airports and their direct flight connections.

over time, while we focus on temporal node behavior in a fixed graph.

Graph signal processing views node values as a smooth signal over the graph structure that can be sparsely encoded using a graph Fourier dictionary [27, 31, 7]. Methods from this active research field do not consider temporal graph signals and assume that the input graph structure is correct and observed at the optimal "spatial" resolution. We challenge the above assumption and demonstrate that an aggregated graph can provide a more advantageous representation for temporal graph signals.

Dimensionality reduction and signal compression techniques can be employed to group the nodal time series and compress their representation. For example, compression methods [20, 32] and dictionary coding approaches [33, 9] for time series can significantly reduce the size of a temporal graphs signal. However, they cannot take advantage of the graph structure as side information. This limitation was addressed in a recent dual graph-time dictionary encoding approach [24], but, it fully trusts the input graph and as we demonstrate experimentally its quality can be improved by our methods summarization of the input graph. While methods in this category employ dictionary encoding to succinctly represent the temporal dimension of signals, we focus on "compressing" the spatial mode (i.e., aggregating the input graph), rendering SAGA complementary to them.

Graph embedding methods learn representations for graph nodes that summarize the local neighborhood structure, often employing random walks to generate a sample of neighbors to train on [11]. Some methods extend this idea to utilize the richer information contained in attributed [2] or timestamped graphs [25]. Closest to our setting is a recent work that identifies groups of signals in a temporal attributed graph that belong to the same user [14]. Methods from this group assume fixed attributes and/or evolving graph structure, while in this work our goal is to summarize a fixed structure among nodes with evolving signals on them.

Community detection for static [12, 22] and temporal [19, 10] graphs is also a relevant topic as our supernode aggregation can be viewed as communities of shared temporal behavior. Most dynamic community methods focus on evolving structure and are not applicable to our temporal graph signal setting. The method most similar to ours is CCTN [19] as it utilizes a fixed graph and a temporal signal on nodes to learn a latent feature space in which communities are detected using k-means. As we demonstrate experimentally, unlike SAGA's supernodes, the communities detected by CCTN and other baselines do not lead to improvements in the quality and scalability of forecasting for temporal graph signals.

6 Conclusion

We proposed SAGA—a general method for aggregation of temporal graph signals (TGS). Our method provides an interpretable coarse-grained representation of the graph informed by both the temporal behavior of nodes and their structural network organization. We demonstrated that SAGA's aggregation can benefit multiple state-of-the-art algorithms for link prediction, forecasting and data compression. In particular, for both synthetic and real-world data sets SAGA's model improved reconstruction quality for temporal graph signals by up to 75%, link prediction accuracy by up to 40%, and the accuracy of forecasting by up to 63% while also offering up to 2-fold speed-up. The discovered aggregation in an Air traffic dataset also aligned well with major geopolitical transportation flows. Our code is available at `http://www.cs.albany.edu/~petko/lab/code.html`.

Acknowledgements: This research is funded by an academic grant from the National Geospatial-Intelligence Agency (Award No. # HM0476-20-1-0011, Project Title: Optimizing the Temporal Resolution in Dynamic Graph Mining). Approved for public release, 22-039.

References

[1] Hubway data:. `http://hubwaydatachallenge.org`.

[2] N. K. Ahmed, R. Rossi, J. B. Lee, T. L. Willke, R. Zhou, X. Kong, and H. Eldardiry. Learning role-based graph embeddings, 2018.

[3] Z. Bar-Joseph. Analyzing time series gene expression data. *Bioinformatics*, 20(16):2493–2503, 2004.

[4] R. H. Bartels and G. W. Stewart. Solution of the matrix equation ax + xb = c [f4]. *Commun. ACM*, 15(9):820–826, Sept. 1972.

[5] P. Bickel, C. Chen, J. Kwon, J. Rice, and E. Zwet. Traffic flow on a freeway network. 01 2002.

[6] S. Boyd, N. Parikh, and E. Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[7] X. Dong, D. Thanou, M. Rabbat, and P. Frossard. Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36(3):44–63, 2019.

[8] N. Eagle and A. S. Pentland. Reality mining: sensing complex social systems. *Personal and ubiquitous computing*, 10(4):255–268, 2006.

[9] V. Goepp, O. Bouaziz, and G. Nuel. Spline regression with automatic knot selection. 08 2018.

[10] A. Gorovits, E. Gujral, V. Papalexakis, and P. Bogdanov. Larc: Learning activity-regularized overlapping communities across time. In *ACM International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD 2018)*, 2018.

[11] A. Grover and J. Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.

[12] J. P. Hespanha. An efficient matlab algorithm for graph partitioning. Technical report, 2004.

[13] M. Huisman. Imputation of missing network data: Some simple procedures. 2009.

[14] D. Jin, M. Heimann, R. Rossi, and D. Koutra. node2bits: Compact time- and attribute-aware node representations for user stitching, 2019.

[15] D. Koutra, U. Kang, J. Vreeken, and C. Faloutsos. Vog: Summarizing and understanding large graphs. In *SDM*, 2014.

[16] K. Lee, H. Jo, J. Ko, S. Lim, and K. Shin. Ssumm: Sparse summarization of massive graphs. In *Proc. of SIGKDD*, pages 144–154, 2020.

[17] Z. Lin, M. Chen, and Y. Ma. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *ArXiv*, abs/1009.5055, 2013.

[18] Y. Liu, A. Dighe, T. Safavi, and D. Koutra. A graph summarization: A survey. *CoRR*, abs/1612.04883, 2016.

[19] Y. Liu, L. Zhu, P. Szekely, A. Galstyan, and D. Koutra. Coupled clustering of time-series and networks. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pages 531–539. SIAM, 2019.

[20] B. Lkhagva, Y. Suzuki, and K. Kawagoe. New time series data representation esax for financial applications. pages x115 – x115, 02 2006.

[21] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[22] K. Macropol and A. Singh. Scalable discovery of best clusters on large graphs. *Proc. VLDB Endow.*, 3(1–2):693–702, Sept. 2010.

[23] G. Mahalakshmi, S. Sridevi, and S. Rajaram. A survey on forecasting of time series data. In *Proc. of ICCTIDE)*, pages 1–8, 2016.

[24] M. J. McNeil, L. Zhang, and P. Bogdanov. Temporal graph signal decomposition. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery amp; Data Mining*, KDD '21, page 1191–1201, New York, NY, USA, 2021. Association for Computing Machinery.

[25] G. H. Nguyen, J. B. Lee, R. A. Rossi, N. K. Ahmed, E. Koh, and S. Kim. Continuous-time dynamic network embeddings. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 969–976, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

[26] X. Olive, M. Strohmeier, and J. Lübbe. Crowdsourced air traffic data from The OpenSky Network 2020 [CC-BY]. Sept. 2021.

[27] A. Ortega, P. Frossard, J. Kovačević, J. M. Moura, and P. Vandergheynst. Graph signal processing: Overview, challenges, and applications. *Proceedings of the IEEE*, 106(5):808–828, 2018.

[28] P. C. B. Phillips. Fully modified least squares and vector autoregression. *Econometrica*, 63(5):1023–1078, 1995.

[29] N. Shah, D. Koutra, T. Zou, B. Gallagher, and C. Faloutsos. *TimeCrunch: Interpretable Dynamic Graph Summarization*, page 1055–1064. Association for Computing Machinery, New York, NY, USA, 2015.

[30] Z. Shen, K.-L. Ma, and T. Eliassi-Rad. Visual analysis of large heterogeneous social networks by semantic and structural abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 12(6):1427–1439, 2006.

[31] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, May 2013.

[32] Y. Sun, J. Li, J. Liu, B. Sun, and C. Chow. An improvement of symbolic aggregate approximation distance measure for time series. *Neurocomputing*, 138:189–198, 2014.

[33] S. V. Tenneti and P. P. Vaidyanathan. Nested periodic matrices and dictionaries: New signal representations for period estimation. *IEEE Trans. Signal Processing*, 63(14):3736–3750, 2015.

[34] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *2008 IEEE 24th International Conference on Data Engineering*, pages 963–972, 2008.

[35] T. Viklands. Algorithms for the weighted orthogonal procrustes problem and other least squares problems. 2006.

[36] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

[37] M. Zhang and Y. Chen. Link prediction based on graph neural networks. *CoRR*, abs/1802.09691, 2018.

[38] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. 32, Apr. 2018.

[39] N. Zhang, Y. Tian, and J. M. Patel. Discovery-driven graph summarization. In *Proc. of IEEE ICDE*, 2010.

Supplement overview

In the supplement we add information that could not be included in the main paper due to space constraints. This information provides a fuller picture of the experiments with a detailed description of how hyperparameters were tuned for all techniques to enable transparency and reproducibility. Specifically, we add details on dictionary definitions (Sec.6.1), dataset description (Sec.6.2), and parameter selection (Sec.6.5). In addition, we provide further details on the Air traffic case study in the form of a table listing the top ten Airlines per detected supernode discussed in the case study (Table.4). We also add an extra experimental figure on link prediction AUC and runtime for the *Road* dataset in Fig.8.

6.1 Detailed dictionary description. The dictionaries used in our evaluation are listed in Table. 5. We provide a short definition for each of them next and citations for further details.

The *Discrete Fourier Transform (DFT) [33]* dictionary is for signals with N time steps and is defined as the following:

(6.15)
$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & ... & 1 \\ 1 & \omega & \omega^2 & ... & \omega^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{(N-1)} & \omega^{2(N-1)} & ... & \omega^{(N-1)(N-1)} \end{bmatrix},$$

letting $\omega = e^{\frac{-2\pi i}{N}}$ and i is the imaginary unit. This basis is unitary.

The *Ramanujan periodic dictionary [33]* is designed for periodic time series and is generated by concatenating period-specific sub-matrices of various widths $R = [\Phi_1, .., \Phi_{g_{max}}]$, where g_{max} is the a maximum period which desired to be modeled and Φ_i is the periodic basis of period g_i. Period-specific matrices $\Phi_g = [D_{d_1}, D_{d_2}, ...D_{d_K}]$ have columns determined by the divisors $\{d_1, d_2, ...d_K\}$ of g. $D_{d_i} \in \mathbb{R}^{g \times \phi(d_i)}$ is a periodic basis for period d_i of the following circulant matrix form:

(6.16)
$$D_{d_i} = \begin{bmatrix} C_{d_i}(0) & C_{d_i}(g-1) & ... & C_{d_i}(1) \\ C_{d_i}(1) & C_{d_i}(0) & ... & C_{d_i}(2) \\ ... & ... & ... & ... \\ C_{d_i}(g-1) & C_{d_i}(g-2) & ... & C_{d_i}(0) \end{bmatrix},$$

where the number of columns, $\phi(d_i)$ denotes the Euler totient function. Elements $C_{d_i}(g)$ are computed as the Ramanujan sum:

(6.17)
$$C_{d_i}(g) = \sum_{k=1, gcd(k,d_i)=1}^{d_i} e^{j2\pi kg/d_i},$$

where $gcd(k, d_i)$ is the greatest common divisor of k and d_i.

The *Spline dictionary [9]* should be utlizied to encode smoothly-evolving temporal signals and is generated by employing B-splines $B_{i,d}(u)$, defined by the Cox-de-Boor formula:

$$B_{i,p} = \frac{u - u_i}{u_{i+p} - u_i} B_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} B_{i+1,p-1}(u),$$

where $B_{i,0} = 1$ if $u_i \leq u < u_{i+1}$, and 0 otherwise. $B_{i,d}(u)$ is non-zero in the range of $[u_i, u_{i+d+1})$. This dictionary is non-orthogonal.

6.2 Datasets description. The *Synthetic* data generation consists of 15 groups which vary in size from 10 to 35 nodes. We randomly assign 30% of possible edges between group members and 25% of possible edges between members of different groups. The signal is 1000 time points long. The first 5 time steps are Gaussian random with mean 0 and variance 1. The following 995 time steps are generated by vector auto regression (VAR) process of lag 5 with a simultaneously added Markovian random walk on each node and 10% of the variance at edge timestamp removed to ensure the signal does not exponentially increase. Finally we subtract the mean of each time step and divide by the variance to normalize the time points.

The *Air Traffic* dataset consist of readings from the OpenSky Network [26] from January 1 2019 to January 31 2019. We utilize readings that contain the origin, destination, callsign and last seen time of observed flights to extract a TGS. The nodes are the airports, the connections in the graph (edges) are the origin and destination, and the temporal signal on the nodes is the number of incoming flights for 6 hour time blocks. We remove airports with less than 8 flights a day.

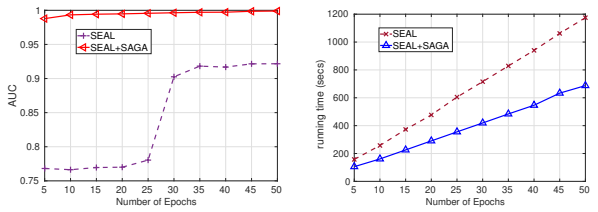
The *Bike* [1] dataset contains daily *Bike* check-out counts at rental stations in Boston. Pairs of stations are connected by an edge if within approximately 2.22 km. The graph in the *Road* [5] dataset corresponds to a highway network where nodes are locations of inductive loop sensors. We use the average speed (at a resolution of 5 mins) at sensors as our evolving graph signal.

Reality Mining (RM) [8] tracks the number of hourly interactions of 142 people at Reality Mining where an edge between two individuals exists if they interacted at least 100 times. We normalize all real world datasets by zscoring (subtracting the mean and then dividing by the variance) the time series vectors of each node for all applications with the exception of forecasting.

6.3 Parameter sensitivity We are interested in characterizing the sensitivity of SAGA to our λ parameters. To achieve this, we first generate a Synthetic dataset of ground truth number k of supernodes each with a unique periodic signal and some random edges between them. We then disaggregate these supernodes

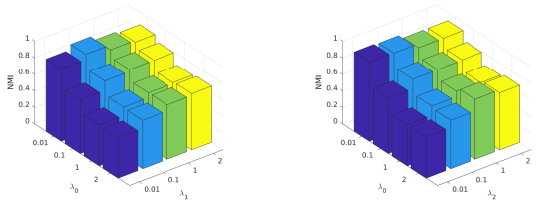
Group Color	top 10 airlines
Green	Ryanair, Lufthansa, easyJet, Air France, British, Eurowings, Scandinavian, Alitalia, Swiss International Air Lines, Royal Dutch
Blue	American, Southwest, United, Delta Air Lines, JetBlue, SkyWest, Republic, Spirit, Jetstar, Qatar, Mesa
Pink	Southwest , Emirates, American, IndiGo, Jet Airways, Turkish , Aeroflot, FedEx Express, Air India, SkyWest
Red	Southwest, Virgin Australia, Qantas, Alaska, United, Jetstar Airways, American, SkyWest, Delta, Air New Zealand

Table 4: The top 10 airlines per supernode found by SAGA in the case study.



(a) Epoch vs area under curve (b) Epoch vs run time Road Road

Figure 8: AUC and running time comparison between SEAL and SEAL+SAGA on the *Road* dataset



(a) λ_0 v.s. λ_1 (b) λ_0 v.s. λ_2

Figure 7: Parameter sensitivity of our method measured by the NMI with GT groups being nodes in a supernode

TGSD

Parameters/Dataset	*Synthetic & Air*
λ_1, λ_2	{.001,.01}
k	[5:5:25]
Parameters/Data	*Bike*
λ_1, λ_2	{.001,.01,.1}
k	[5:5:25]
Parameters/Data	*Road*
λ_1, λ_2	{.001}
k	[25:25:125]

SAGA

Parameters/Dataset	*Synthetic & Air*
λ_0, λ_1, λ_2	{.001,.01}
Parameters/Data	*Bike*
λ_0, λ_2	{.001,.01,.1}
Parameters/Data	*Road*
λ_0, λ_1, λ_2	{.001}

Table 6: Parameter search space for TGSD and SAGA in dictionary learning. Curly braces (e.g. $\{.1, 1, 10\}$) indicate a set of values we tested and brackets (e.g. $[5:5:25]$) indicate that we iterate in an interval from the first to the third values by a step specified in the middle value

	DFT (D)	Ramanujan (R)	Spline (S)
Orthogonal	✓		
Parameter-free	✓		
Temporal Assumption	Periodic	Periodic	Trends

Table 5: Summary of dictionaries Φ we experiment with.

into 4 to 15 "constituent nodes". The edges between the two supernodes are randomly attached to various "constituent nodes" of the two supernodes. We include 70% of the possible edges between members of the same supernode. We randomly split the original signal of a supernode and assign these fractions of the original signal to member nodes and add Gaussian white noise at a single-to-noise ratio of 20. We test the effect of our hyper-parameters $\{\lambda_0, \lambda_1, \lambda_2, \}$ on SAGA's ability to recover the correct supernode membership measured by the normalized mutual information (NMI).

In Fig. 7, we present the sensitivity of our model to these parameters. We fix one parameter and vary the other two. SAGA performs best when λ_0 is small as large values will push the model to avoid picking many large supernodes. SAGA also preforms better in this setting with larger λ_1 and λ_2. The supernodes temporal signal is relatively simple meaning a high λ_1 will avoid overfitting to noise and enable or temporal fit to focus on the generating process. There is also high connectivity for members of supernodes meaning that encouraging more network connectivity within supernodes by raising λ_2 will encourage SAGA to map the member nodes to the correct supernode.

6.4 Additional link prediction experiments. In Fig. 8 we show additional characterization of SEAL's [37] training when employing the original dataset (SEAL) and SAGA's aggregation matrix A as node features (SEAL+SAGA). This additional analysis is for the *Road* dataset and parallels the observations on the RM dataset presented in the main paper.

6.5 Parameter tuning for all experiments. Finally, to ensure reproducibility we detail the parameters utilized in dictionary learning in Tbl. 6, SAGA's optimal parameters for link prediction and forecasting in Tbl. 7, and the grid search ranges for optimal parameters in forecasting for CCTN and SAGA in Tbl. 9. The parameters used for vector auto regression (VAR) forecasting on each dataset are listed in Tbl. 9.

Data-Task	λ_0	λ_1	λ_2	k
RM, Air, Bike, Road - Link prediction-R	.01	1	1	10
RM, Air, Bike, Road - Link prediction-S	.5	.1	.1	10
RM, Air, Bike, Road - Link prediction-D	.01	.01	1	10
RM, Air, Bike, Road - Link prediction-D	.01	.01	1	10
Synthetic - Forecasting-R	.1	.1	0.05	15
Air - Forecasting-R	.05	.1	1	100
Bike - Forecasting-S	.5	.1	1	25
Road - Forecasting-S	.1	.05	0.01	2

Table 7: Parameters for SAGA

CCTN

	number of groups
Synthetic	{15}
Air	{5,10,25,50,100}
Bike	{5,10,25,50,100}
Road	{2,5,10,25,50,100}

SAGA

Parameters/Dictionaries/Dataset	*all/*
λ_0	{.01,.05,.1,.5,1}
λ_1	{.01,.05,.1,.5,1}
λ_2	{.01,.05,.1,.5,1}

Table 8: Parameter search space for competing methods forecasting. Curly braces (e.g. $\{.1, 1, 10\}$) indicate a set of values we tested and brackets (e.g. $[5 : 5 : 25]$) indicate that we iterate in an interval from the first to the third values by a step specified in the middle value ($[5 : 5 : 25]$ represents the values $\{5, 10, 15, 20, 25\}$)

	trainning length	sliding window size	lag
Synthetic	1000	NA	5
Air	95	2	5
Bike	230	4	5
Road	600	10	15

Table 9: Parameters used by VAR for all methods in forecasting.