HashNWalk: Hash and Random Walk Based Anomaly Detection in Hyperedge Streams (Supplementary Document)

1 Extension to Hyperedge Deletions

While we focus on updating the hypergraph summary with respect to incoming hyperedges, the proposed update scheme is easily extensible to deleting existing hyperedges. Assume m hyperedges $e_1, \cdots, e_m$ have arrived at time t_m and we aim to delete e_k which had arrived at time t_k ($1 \leq k \leq m$). Let $\tilde{P}^{(m,k)}_{\tilde{u},\tilde{v}}$ be the updated proximity from supernode $\tilde{u}$ to supernode $\tilde{v}$. Accordingly, let $S^{(m,k)}_{\tilde{u},\tilde{v}}$ and $T^{(m,k)}_{\tilde{u}}$ be the updated matrix S and T, respectively. Then, $\tilde{P}^{(m,k)}_{\tilde{u},\tilde{v}}$ can be written as

$$\tilde{P}^{(m,k)}_{\tilde{u}\tilde{v}} = \frac{S^{(m,k)}_{\tilde{u},\tilde{v}}}{T^{(m,k)}_{\tilde{u}}}$$

where $S^{(m,k)}_{\tilde{u},\tilde{v}}$ and $T^{(m,k)}_{\tilde{u}}$ can be updated in constant time as:

$$\begin{aligned} S^{(m,k)}_{\tilde{u},\tilde{v}} &= S^{(m)}_{\tilde{u},\tilde{v}} - \alpha^{-t_k} \cdot \mathbb{1}(\tilde{u} \in \tilde{e}_k) \cdot \frac{\gamma_{\tilde{e}_k(\tilde{v})}}{\tilde{R}_{\tilde{e}_k}}, \\ T^{(m,k)}_{\tilde{u}} &= T^{(m)}_{\tilde{u}} - \alpha^{-t_k} \cdot \mathbb{1}(\tilde{u} \in \tilde{e}_k). \end{aligned}$$

These are immediate from the definition of $S^{(m)}_{\tilde{u},\tilde{v}}$ and $T^{(m)}_{\tilde{u}}$:

$$\begin{aligned} S^{(m,k)}_{\tilde{u},\tilde{v}} &= \sum_{i \in \{1,\cdots,m\} \setminus \{k\}} \alpha^{-t_i} \cdot \mathbb{1}(u \in \tilde{e}_i) \cdot \frac{\gamma_{\tilde{e}_i}(v)}{\tilde{R}_{\tilde{e}_i}} = S^{(m)}_{\tilde{u},\tilde{v}} - \alpha^{-t_k} \cdot \mathbb{1}(\tilde{u} \in \tilde{e}_k) \cdot \frac{\gamma_{\tilde{e}_k(\tilde{v})}}{\tilde{R}_{\tilde{e}_k}}, \\ T^{(m,k)}_{\tilde{u}} &= \sum_{i \in \{1,\cdots,m\} \setminus \{k\}} \alpha^{-t_i} \cdot \mathbb{1}(u \in \tilde{e}_i) = T^{(m)}_{\tilde{u}} - \alpha^{-t_k} \cdot \mathbb{1}(\tilde{u} \in \tilde{e}_k) \end{aligned}$$

As in Lemma 4, these updates takes $O(\min(M, |e|^2))$ time to update S and T.

2 Experimental Details

In this section, we discuss details of experimental settings.

Parameter settings of anomaly-detection algorithms: We discuss how we search the hyperparameters of each method to obtain the best performance reported in the experimental results. For baselines, we include configurations requiring more space than ours. The range of configurations used for each method is as follows:

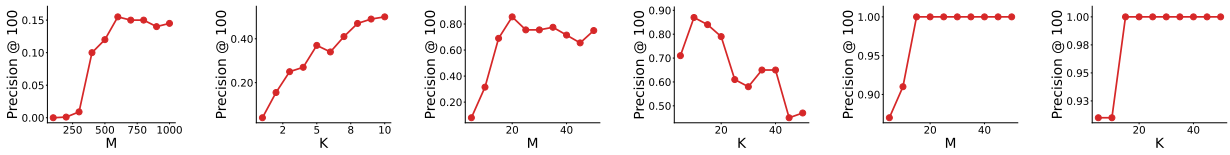
- **HashNWalk:** In `SemiU` & `SemiB`, we set $\alpha = 0.98$, $K = 15$, and $M = 20$. In `Transaction`, we set $\alpha = 0.98$, $K = 4$, and $M = 350$.
- **SedanSpot:** In `SemiU` & `SemiB`, we search from restart probability $\in \{0.10, 0.15, 0.20\}$, sample size $\in \{10000, 50000\}$, and number of random walks $\in \{50, 100\}$. In `Transaction`, we search from restart probability $\in \{0.10, 0.15, 0.20\}$, sample size $\in \{100000, 500000\}$, and number of random walks $\in \{50, 100\}$.

- **Midas:** In `SemiU` & `SemiB`, we search from number of buckets $\in \{5000, 10000, 20000\}$, number of hash functions $\in \{2, 8\}$, and decaying factor $\in \{0.3, 0.5, 0.7\}$. In `Transaction`, we search from number of buckets $\in \{10000, 50000, 100000\}$, number of hash functions $\in \{2, 8\}$, and decaying factor $\in \{0.3, 0.5, 0.7\}$.
- **F-FADE:** In `SemiU` & `SemiB`, we search from time interval for model update $\in \{100, 1000\}$, number of epochs $\in \{5, 10\}$, upper limit of memory size $\in \{10000, 20000\}$, online train steps $\in \{10, 20\}$, and cut-off threshold $\in \{0.1, 0.01, 0.001\}$. In `Transaction`, we search from time interval for model update $\in \{100, 1000\}$, number of epochs $\in \{5, 10\}$, upper limit of memory size $\in \{10^4, 10^5\}$, online train steps $\in \{10, 20\}$, and cut-off threshold $\in \{0.1, 0.01, 0.001\}$. The embedding size is fixed to 200 and the time decaying parameter is set to 0.999.
- **LSH:** For all datasets, we search from number of bands $\in \{2, 4, 8\}$ and the length of each band $\in \{2, 4, 8\}$.

Notably, HashNWalk covers 28.6% and 22.5% of the space used in the original hypergraphs in `Transaction` and semi-real hypergraphs, respectively. Meanwhile, note that LSH uses 4 times of the space of the original hypergraphs.

All anomalies are injected after time t_{setup}, and thus all methods are evaluated from time t_{setup}. We inject 200 anomalous hyperedges in `email-Enron` by InjectionU and InjectionB and generate `SemiU` and `SemiB`, respectively. We set $t_{\text{setup}} = t_{100}$ $(< t_{|E| \cdot 0.01})$. In InjectionU, we set $g = 200$. In InjectionB, we set $m = 20$, $|N| = 5$, and $l = 10$.

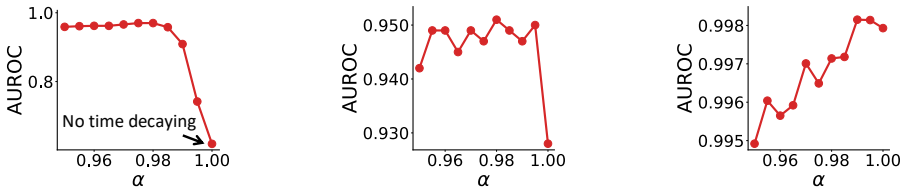
Parameter analysis: We evaluate HashNWalk with different numbers of supernodes M and hash functions K. As shown in Figure 1, the performance of HashNWalk depends on these parameters, and in most cases, there is a positive correlation with M and K. Intuitively, a larger number of supernodes and hash functions collectively reduce the variance due to randomness in hash functions. However, the space usage is dependent on these parameters, and hence an appropriate trade-off between the accuracy and the space usage should be taken into consideration.



(a) M in TS (b) K in TS (c) M in SU (d) K in SU (e) M in SB (f) K in SB

Figure 1: The performance (specifically, Precision@100) of HashNWalk depends on the number of supernodes (M) and the number of hash functions (K). TS: `Transaction`. SU: `SemiU`. SB: `SemiB`.

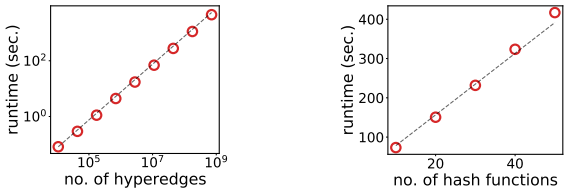
Since HashNWalk is modeled to decay the weights of past hyperedges, we analyze how this strategy influences the accuracy of HashNWalk. To this end, we evaluate the performance of the HashNWalk with different αs. As shown in Figure 2, a proper weight decaying strategy improves the accuracy of HashNWalk. This indicates that not only structural information but also the temporal information is critical in detecting anomalies in hyperedge streams.



(a) `Transaction` (b) `SemiU` (c) `SemiB`

Figure 2: Decaying the weights of past hyperedges properly improves the accuracy of HashNWalk.

Scalability: As shown in Figure 3, the total runtime of HashNWalk is linear in the number of hyperedges and the number of hash functions.



(a) Number of hyperedges (b) Number of hash functions

Figure 3: The total runtime of HashNWalk is linear in the number of hyperedges ($\mathcal{E}$) and the number of hash functions (K) in the upscaled hypergraph of `email-Enron`.

Case study on `cite-patent`: We provide the titles of the patents presented in Figure 4(b) in the main paper. The numbers correspond to those in the figure.

Unexpected patent

1. Semiconductor integrated circuit and method for controlling semiconductor integrated circuit

Normal patents

2. Pushing force deviating interface for damping mechanical vibrations
3. Multistage compressor
4. Hot swappable computer card carrier

Bursty patents

5. Querying of copyright host, printing of copyright information and host registration of copyright data
6. Sheet metal rocker arm, manufacturing method thereof, cam follower with said rocker arm, and assembling method thereof
7. Turbine shroud thermal distortion control