Minimizing Empty Truck Miles Driven via Link Prediction on a Temporal Transportation Graph and Decision-Theoretic Online Learning

Yoonseo Ko

Advisor(s): Cong Ma
(at least one statistics faculty member)

Approved ______ (Signatures of the advisors)

Date ______ (Date of the signatures)

Abstract

Reducing total miles driven by unloaded trucks is a critical challenge in the transportation industry with significant economic and environmental implications. This paper frames the issue as a statistical decision problem in an online setting, where a driver who has just completed a delivery must decide whether to wait at the current location for a new order or return to the starting point.

The study presents a link prediction algorithm applied to a time-varying transportation demand graph, utilizing Temporal Random Walks within the Node2Vec embedding framework. Then, it assesses the performance of Hedge algorithm and Weighted Majority algorithm based on predicted probabilities of new edges to neighboring nodes generated by both the proposed prediction algorithm and traditional static graph link prediction algorithms. The study demonstrates the proposed algorithm's advantage by achieving sublinear regret.

Keywords: Time-Varying Graph, Representation Learning, Link Prediction, Hedge Algorithm, Online Learning

Acknowledgements

I would like to thank Professor Cong Ma for his insightful advising, Lineage Logistics LLC.'s Data Science Department for providing the data for experiments, my parents, and my friends for their infinite support throughout writing this thesis.

Contents

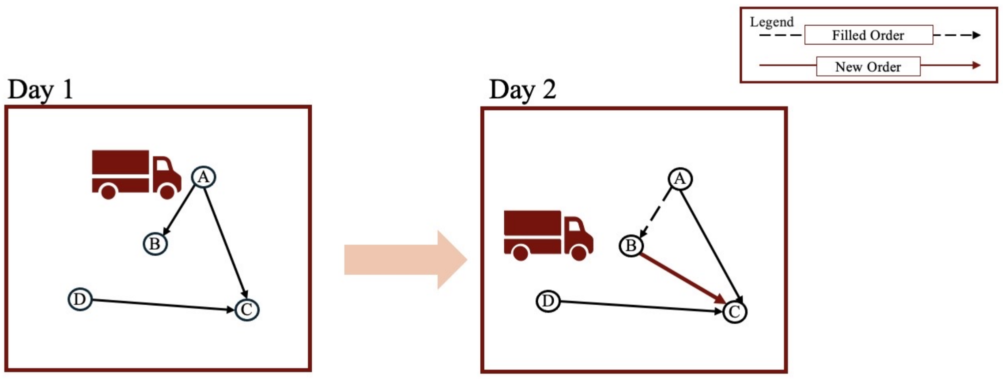
1 Introduction

1.1 Introduction

Lineage Logistics is a warehouse company that specialises in cold storage and transportation. It offers storage services in the nation-wide facilities the company owns, transportation services, as well as connecting the clients with third-party / company's delivery services. The company thus is able to pick up the loads from the clients, stores them in the facilities, then delivers them to the end destination that the client wants the load to be at. For example, consider Subway, a sandwich company, as an example. They may order a truck to pick up vegetables from a local farm, deliver them to Lineage's facility, store them until one of their store needs a restock, then will schedule a restock order. Then an outbound load is scheduled and will be delivered from the facility to a Subway store.

In a transportation industry, it is a common practice to sign a round trip contract with drivers. Consider stops a, b for facility the load was stored in and destination the load is to delivered to. Based on the contract, a driver that completed an outbound load delivery from a to b now need to travel back to a. This can be understood as detecting a cycle in the graph given a start node. He may fail to find a delivery order from b to a that corresponds to his schedule on the way back and drive without any load on the truck. Such miles driven are thus called *empty miles*. Furthermore, we can also consider a scenario where the truck drivers do not necessarily have to return to the origin, but simply needs to move to a new location c to pick up new orders. Then, if there is no delivery scheduled from b to c, this will add to the total *emtpy miles* driven.

Figure 1: A visualization of an empty mile problem



There are multiple ways to reduce empty miles, including but not limited to finding a direct delivery from the destination back to facility. That is, we can think about a truck moving from Stop A to Stop B then waiting for a couple hours until, at stop B, they find a load scheduled for delivery back to Stop A. Such idea has already been explored in the company's previous research.

The goal of this thesis is to reduce the total empty miles driven by all drivers using staticial approaches. From the explained scenario, a natural question we can ask are:

1. What new orders from B will there be?
2. Should the driver wait until the new order from B is scheduled?

That is, we want to conduct a prediction of potential delivery opportunities present for each route, utilize it to make decisions of whether to wait, and , ultimately, minimize *empty miles*.

1.2 Research Questions

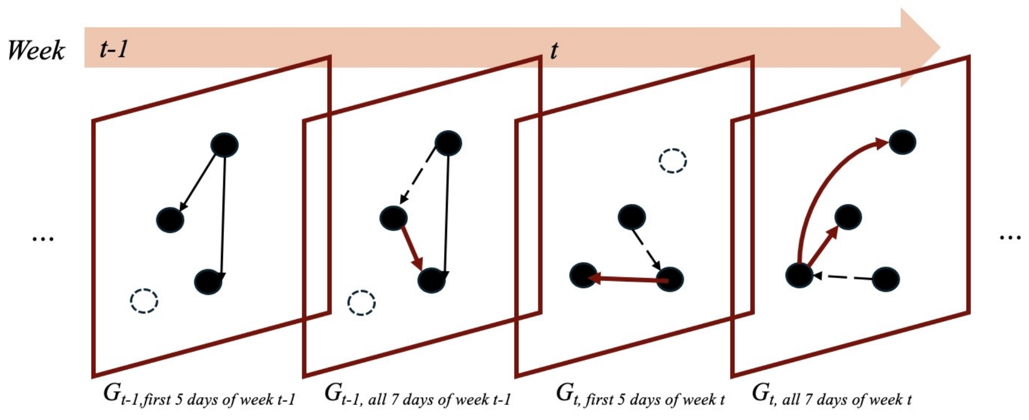
1.2.1 Temporal Graph Definition

First we define relevant terms in use. Let $G_t = (V_t, E_t)$ be a static, directed graph in week t, $1 \leq t \leq N$ where

- $N \in \mathbb{N}$ is the number of weeks in the given data,
- V_t is the set of nodes in G_t, and
- E_t is the set of all directed edges $\{e_{ij,t} = (v_i, v_j)\}$ where $v_i, v_j \in V_t$, e_{ij} is a directed edge from node v_i to node v_j and its attribute is the time it was scheduled.

Let $G_{t,old} \subset G_t$ such that $G_{t,old} = (V_{t,old}, E_{t,old})$ where, by some day n $(1 \leq n \leq 7)$ of week t, $E_{t,old}$ is the set of edges existing in $G_{t,old}$ by day n and $V_{t,old}$ is the collection of nodes connected by edges $E_{t,old}$. Let $\mathbf{G} = \{G_1, G_2, \cdots G_i, \cdots G_N\}$ be the temporal graph, or the collection of all such static graphs G_t's. Also, let $V = \cup_{i=1}^{N} V_t$, $N_V = |V|$, $E = \cup_{i=1}^{N} E_t$, and $N_E = |E|$.

Figure 2: A visualization of a temporal graph


Week
t-1
t
...
...
$G_{t-1, first\ 5\ days\ of\ week\ t-1}$
$G_{t-1,\ all\ 7\ days\ of\ week\ t-1}$
$G_{t,\ first\ 5\ days\ of\ week\ t}$
$G_{t,\ all\ 7\ days\ of\ week\ t}$

1.2.2 Problem Statement

Notice that the problem then can be separated into not only computing a probability, but making a decision based on the probability computed. Then, the problem consists of two parts, where we first compute the probability of a link existing in a graph and then make decision based on the probability of existence.

Prediction Problem First, let us define the prediction problem. Suppose for each week t, $1 \leq t \leq N$, we know $G_{t,old}$.

1. $\forall e_{ij} \in \{e_{ij} \notin E_{t,old} \mid v_i, v_j \in V_{t,old}\}$, can we predict whether $e_{ij} \in E_{t,new}$?
2. Does taking into account temporal data of each edge lead to improving prediction accuracy?

In other words, define an indicator function for some nodes $v_i, v_j \in V_{t,old}$, $\mathbb{1}_{ij}$ such that

$$\mathbb{1}_{ij} = \begin{cases} 1, & \text{if } e_{ij} \in E_t \\ 0 & \text{else .} \end{cases}$$

Then, we can define *positive edges* as $\{e_{ij} \in G_{t,new}$ such that $\mathbb{1}_{ij} = 1\}$ and *negative edges* as $\{e_{ij} \notin G_t$ such that $\mathbb{1}_{ij} = 0\}$. The power of our model would be how well the model predicts the positive edges while rejecting negative edges' existence. We will hence use ROC-AUC score that measures how well the model distingushes between two positive and negative classes.

Decision Problem Now, we define the decision problem. Suppose for each week t, $1 \leq t \leq N$, we have a set of probabilities of any potential edges in $G_{t,new}$, $\{p_{ij} \mid v_i, v_j \in V_{t,old}\}$, that was computed based on $G_{t,old}$ using some link prediction algorithm. Let $L_t : \{e_{ij} \notin E_{t,old} \mid v_i, v_j \in V_{t,old}\} \to \mathbb{R}$ be the loss function for all positive and negative edges in G_t.

1. Can we compute a dynamic threshold for each edge e_{ij}, $\alpha_{t,ij}$, such that if $p_{ij} > \alpha_{t,ij}$, we predict that the edge would exist and, therefore, achieve a sublinear regret over time?
2. Furthermore, can we adjust the threshold by punishing false positives more harshly than false negatives?

1.3 Review of Relevant Literature

Static & Temporal Link Prediction Algorithms Recent decades have seen significant advances in link prediction algorithms. The DeepWalk method, introduced by Perozzi et al. [PAS14], leverages random walks to learn social representations, effectively mapping the social structure into a low-dimensional space. Expanding on this, Saxena et al. [SFP21] developed NodeSim, which enhances prediction accuracy by focusing on node similarities within network embeddings.

Then, there have been attempt to extend static link prediction into temporal link prediction, which accounts for the dynamic aspects of networks. Nguyen et al. [Ngu+18] proposed a model for Continuous-Time Dynamic Network Embeddings that introduces a temporal random walk mechanism, adapting embeddings over time to forecast links. A comprehensive examination and categorization of various frameworks in this area are provided by Qin et al. [QY23], who offer a unified framework and taxonomy for temporal link prediction, summarizing recent developments and methodologies in the field.

Online Learning & Hedge Algorithm The domain of online learning and decision-making under uncertainty has been shaped by foundational theories and algorithms. [FS97] were pioneers with their introduction of the hedge algorithm, which has influenced numerous applications in online learning, particularly in boosting techniques. The concept of the Weighted Majority Algorithm, introduced by Littlestone and Warmuth [94], further exemplifies the development of decision-theoretic approaches that manage sequential data and uncertainties in predictive modeling. Elad Hazan and Sanjeev Arora [Haz23] have contributed significantly by elucidating complex concepts in online convex optimization and decision-making under total uncertainty, respectively, enhancing the theoretical underpinnings and practical applications of online learning algorithms.

1.4 Data in Use & Constraints

The data in use is provided by Lineage Logistics, LLC., a U.S. cold storage warehouse company. The data consists of inbound delivery order from an origin that a load is picked up from to a Lineage's warehouse and outbound delivery order from a Lineage's warehouse to another warehouse or the final destination. All stops are in the U.S. The data is from January 1st, 2021 to June 1st, 2023 and can be aggregated into zip3 or Key Market Area level (pre-defined area level by the company).

Given the business background of the data, we also have some business and statistical constraints that must be considered when formulating our approach.

1. Interpretability: We aim to construct a probabilistic model that is interpretable. Therefore, no deep learning. This significantly limits our approach as deep learning is one of the most common approach taken in temporal link prediction problems as shown in the literature review.

2. Lack of Parameters: We do not know what covariates influence the scheduling of delivery orders. Therefore, we must consider simple models that only take into account time and graph structure.

3. Node Attributes: Our data only contains the route the truck driver took, the stops in the routes, and date the order was scheduled. Then, we want to design a feature-learning algorithm. Note that we do not consider geographic distance between stops.

2 Link Prediction Algorithm

2.1 Node & Edge Embedding

2.1.1 Representation Learning for Node Attributes

To resolve the lack of node attributes, we must first decide how to learn node features. In this paper, we consider two approaches:

- Static Random Walk
- Temporal Random Walk using exponential biases or node similarity biases of choosing next edge.

Static Random Walk First, we introduce the well-known random walk algorithm used for node embedding [PAS14]. Let $l \in \mathbb{N}$ be the walk length the algorithm should take. Given $G_{t,old}$, $\forall v_i \in V_{t,old}$, let $\mathcal{W}_{v_i}$ be the random walk starting from the node v_i whose total length is l. For each node in the graph, $\mathcal{W}_{v_i}$ is a sequence of nodes $v_i = v_1, v_2, \cdots, v_l$ where $\exists e_{i,i+1} \in E_{t,old} \forall 1 \leq i \leq l-1$. In other words, starting from a node v_i, the algorithm uniformly selects a new node v_{i+1} to visit from v_i's neighbourhood.

Temporal Random Walk Now, we introduce the temporal random walk algorithm suggested in [Ngu+18]. Let $\mathcal{T}_t : E_t \to \mathbb{R}^+$ that maps edges in E_t to its timestamp and $l' \in \mathbb{N}$ be the maximum length of a walk. For $v_i, v_j \in V_T$, a temporal random walk from v_i is defined as a sequence of vertices $\{v_1 = v_i, v_2, \cdots, v_{n-1}, v_{-'}\}$ such that $\mathcal{T}_t(e_{i,i+1}) \leq \mathcal{T}_t(e_{i+1,i+2}) \forall 1 \leq i \leq l'-1$. Then, instead of taking discrete snap shots of G_t on each day, we can consider taking a temporal random walk that does not necessarily have to take a walk each day.

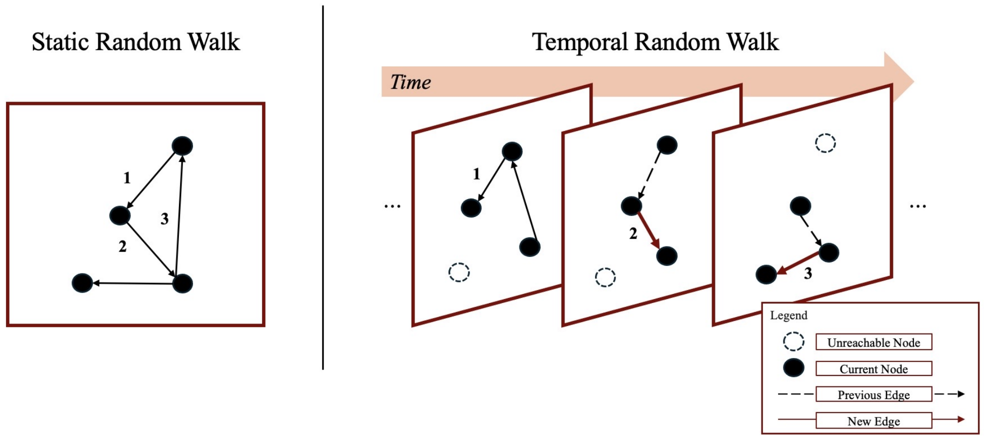
We must note that there can be cases where there are shorter walks than others in the case that there were no temporal neighbors left to choose from. To account for such case, we introduce $\omega \in \mathbb{N}$, a minimum length of each walk. Then, each walk W_{v_i} must have length between w and l'. This also suggests case of zero temporal walk that contains a particular node, due to it being too short or it did not appear in any random walks. In such case, we use zero as the node embedding to indicate the node is 'uninformative'.

Exponential Weighting Instead of uniformly drawing the next node to visit, we incorporated exponential walk bias that favors edges with shorter time gaps that was proven superior [Ngu+18]. That is, for an edge $e_{ij} \in E_t$ with timestamp t, each temporal neighbor of e_{jk} has probability of getting chosen as

$$Pr(e_{jk}) = \frac{\exp(\tau(e_{jk}) - \tau(e_{ij}))}{\sum_{e_{jl}} \exp(\tau(e_{jl}) - \tau(e_{ij}))}.$$

Node Similarity Weighting While using random walks as the main node embedding algorithm, we also explore an additional feature in deciding the probabilities of choosing an edge in each step of random walk. In particular, we also consider using a node-similarity based edge selection method where the random walk takes into account how "similar" two

Figure 3: A comparison of temporal and static random walks



nodes are. For the similarity measure, we use the normalized Jaccard index that takes into account the shared neighbours of a node pair [SFP21]. In transportation order graph's case, this would be taking into account how many shared delivery origin any two stops have.

$$J(v_i, v_j) = \frac{|Neigh_{v_i} \cap Neigh_{v_j}|}{|Neigh_{v_i} \cup Neigh_{v_j}|}$$
$$\Rightarrow Pr(e_i j) = \frac{J(v_i, v_j)}{\sum_{k,l} J(v_k, v_l)}.$$

2.1.2 Node Embedding using Skip-Gram

Now given a set of temporal random walks W_T, we choose a node embedding function to learn the embedding for each node with a goal to maximize probability of observing the random walks. In this paper, we use CTDNE suggested by Nguyen et.al that displays superiority over other common node embedding algorithms such as Deepwalk and Node2Vec. It is a generalization of Skip-Gram that maximizes observing all neighbors of an edge given its embedding using gradient descent:

$$\begin{aligned}\Phi_v &= \max_f \sum_{W_t \in W_T} \log Pr(W_t = \{v_{i-\omega}, \cdots, v_{i+\omega}\} \setminus v_i \mid f(v_i)\}) \\ &= \max_f \sum_{W_t \in W_T} \sum_{v_j \in W_t} \log Pr(v_j \mid f(v_i))\})\end{aligned}$$

The algorithm can be expressed as following [PAS14]:

Algorithm 1 `Skip Gram Algorithm`$(\Phi, \{W_{v_i} \mid v_i \in V_t\}, \omega,\ \alpha = 0.1$

```
**Input** Initial matrix of node embedding $\Phi$, a set of random walks for time $t$ $\{W_{v_i} \mid v_i \in V_t\}$,
context window size $w$
**Output** Matrix of node embedding $\Phi$

for $W_{v_i} \in \{W_{v_i}\}$ do
    for $v_j \in W_{v_i}$ do
        for $v_k \in W_{v_i}[j - w : j + w]$ do $J(\Phi) = -\log Pr(v_k \mid \Phi(v_i))$
            Gradient descent $\Phi = \Phi - \alpha \cdot \frac{dJ}{d\Phi}$
        end for
    end for
end for
```

Note that although in this thesis, we chose to use Skip-Gram model, the temporal random walks can be used as input vectors for other models as well.

2.1.3 Edge Embedding

Let $\Phi_v : V_t \to \mathbb{R}$ such that for $v_i \in V_t$, $\Phi_v(v_i)$ is the node embedding of v_i based on some link prediction algorithm. With the given node embeddings on $V_{t,old}$, we then apply four commonly used binary operators and use a validation set to choose the best performing operator to base our test prediction on. [Ngu+18]

Binary Operator List for $v_i, v_j \in V_t$	
Binary Operator Name	Formula
Product	$\Phi_v(v_i) \cdot \Phi_v(v_j)$
L2 norm	$(\Phi_v(v_i) - \Phi_v(v_j))^2$
L1 norm	$\|(\Phi_v(v_i) - \Phi_v(v_j)\|$
Subtraction	$\Phi_v(v_i) - \Phi_v(v_j)$

2.2 Probability Computation

Let $\Phi_e : \{e_ij = (v_i, v_j) \mid v_i, v_j \in V_{t,old}\} \to \mathbb{R}$ such that for $\Phi_e(e_{ij})$ is the edge embedding of a potential edge e_{ij} based on some edge embedding operator. Let $s : \Phi_e(\to [0, 1]$ be the softmax function such that

$$s_t(e_{ij}) = \frac{exp(\Phi_e(e_{ij}))}{\sum_{e_{kl} | v_k, v_l \in V_{t,old}\}} exp(\Phi_e(e_{kl}))}.$$

Given a set of edge embeddings over all potential edges, we can use this softmax function on the edge embeddings and compute probability of each edge based on the embedding.

3 Decision Threshold

3.1 Expert Problem

Based on the probabilities of an edge existing in the last two days of week t, now we want to construct an algorithm that

1. minimizes false prediction
2. while specifically controlling for false positives.

We consider abstracting the problem into an expert learning problem: At each time step $t = 1, 2, \cdots N$, a driver faces a choice between

- waiting until a new delivery appears to some location in mind
- returning to the origin without delivering additional loads.

The decision maker has assistance in the form of M experts that offer their advice based on M thresholds $[d_1, d_2, \cdots, d_M = 1]$ where d_i is the ith expert's decision threshold such that if the probability of the link of exploration existing $p(e_{ij}) \geq d_i$ then the expert recommends waiting until the link appears.

3.2 Hedge Algorithm

We consider applying a famous algorithm for expert problem that guarantees sublinear regret over time in comparison to naive threshold of 0.5: the hedge algorithm or the randomized weighted majority algorithm. [FS97]

Suppose $l_t(i)$, $1 \leq i \leq M$ is the loss incurred by following the ith expert's advice.

Algorithm 2 `Hedge Algorithm`$(\{\mathbb{1}_{e_{ij,t}}\}_{t=1}^{N}, \epsilon)$

```
Input $\{\mathbb{1}_{e_{ij,t}}\}_{t=1}^{M}$, learning rate $\epsilon$, expert_num
Output Weight vector $W_{e_{ij}}$ that has weight assigned for each expert

initialise weight vector for edge $e_{ij}$, $W_{1,e_{ij}} \leftarrow \mathbf{1}/M \in \mathbb{R}^M$

for $t \in range(1, N+1)$ do
    Choose an action $i_t = i$ with probability $x_t(i) = \frac{W_{t,e}(i)}{\sum_j W_{t,e}(j)}$
    Compute loss $l_t(i_t)$
    Update weights $\forall i$, $W_{i,t+1,e} = W_{t,e} \cdot e^{-\epsilon l_t(i)}$
    Normalize weights $\forall i$, $W_{i,t+1,e} = \frac{W_{i,t+1}}{\sum_{j=1}^{\texttt{expert\_num}} W_{j,t+1,e}}$
end for
```

We denote the expected loss of the algorithm by

$$E[l_t(i_t)] = \sum_{i=1}^{N} x_t(i) l_t(i) = x_t^T l_t.$$

[FS97]

3.3 Exponential Weighted Majority Algorithm

In addition to the hedge aglorithm, we present Weighted Majority algorithm with exponential weight update for expert problem in comparison to naive threshold of 0.5.
[94]

Suppose $l_t(i)$, $1 \leq i \leq M$ is the loss incurred by following the ith expert's advice.

Algorithm 3 Exponential Weighted Majority Algorithm($\{Pr(e_{ij,t})\}_{t=1}^{N}$, $\{\mathbb{1}_{e_{ij,t}}\}_{t=1}^{N}$, ϵ)

```
Input $\{\mathbb{1}_{e_{ij,t}}\}_{t=1}^{M}$, learning rate $\epsilon$, expert_num
Output Weight vector $W_e$ that has weight assigned for each expert

Initialize weight vector for edge $e$, $W_{1,e} \leftarrow \mathbf{1}/M \in \mathbb{R}^M$
Initialize expert_thresholds $\leftarrow$ [1/expert_num] $\cdot$ expert_num

for $t \in range(N)$ do
    Initialize wait_experts $\leftarrow [0] *$ expert_num
    Initialize not_wait_experts $\leftarrow [1] *$ expert_num

    for $i \in range($expert_num$)$ do
        if $Pr(e_{ij,t}) >$ expert_threshold[$i$] then
            wait_experts[$i$] $\leftarrow 1$
            not_wait_experts[$i$] $\leftarrow 0$
        end if
    end for

    if $W_{t,e} \cdot$ wait_experts $> W_{t,e} \cdot$ not_wait_experts then
        action $\leftarrow 1$
    else
        action $\leftarrow 0$
    end if
    Compute loss $l_t($action$)$

    Update weights $\forall i$, $W_{i,t+1,e} = W_{t,e} \cdot e^{-\epsilon l_t(i)}$
    Normalize weights $\forall i$, $W_{i,t+1,e} = \frac{W_{i,t+1}}{\sum_{j=1}^{\text{expert\_num}} W_{j,t+1,e}}$
end for
```

3.4 Parameters & Design Choices

For the hedge algorithm, there are a set of parameters chosen arbitrarily or for false discovery control. We compute the regret of hedge and naive threshold (using threshold of 0.5) over time based on several different parameters chosen as following:

Parameters	
Parameter	Choices
Expert Thresholds	[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
Action	1: Wait, 0: Return
Regret 1	1 for all regrets
Regret 2	3 for false positive, 1 for false negative

For each choice, we compare the cumulative regret over time as well as false positive counts, false negative counts, and true positive counts over time.

4 Experiments & Results

4.1 Experiment setup

There are 126 weeks in the data, from January 1st, 2021 to June 1st, 2023. Based on the lookback size of 5 days, we separate the data of week t into data of first 5 days and last 2 days. We will use the former as training and validation data, and the latter as test data. For link prediction, we execute the following steps:

1. Apply each temporal and static random walk to learn the node features.
2. Transform the features into embeddings ($\in \mathbb{R}$) by using the Word2Vec function.
3. Apply four operator functions to learn edge embeddings.
4. Apply softmax function on the edge embeddings to compute predicted probabilities of each edge.

For decision making algorithm, we execute the following steps:

1. Apply the link prediction algorithm based on temporal random walk and product operator.
2. Based on the computed probabilities of each edge, make a decision based on hedge, weighted majority, etc.
3. Compute the loss and update the weight vector for each expert.

4.2 Link Prediction Algorithm

From both KMA level aggregation and ZIP3 level aggregation, we observed that the product operator performs the best with outstanding validation accuracy compared to the other three operators. Moreover, we also observed that the subtraction operator performs the worst in all cases.

Figure 4: Validation accuracy of prediction algorithms per operator on KMA Level Data

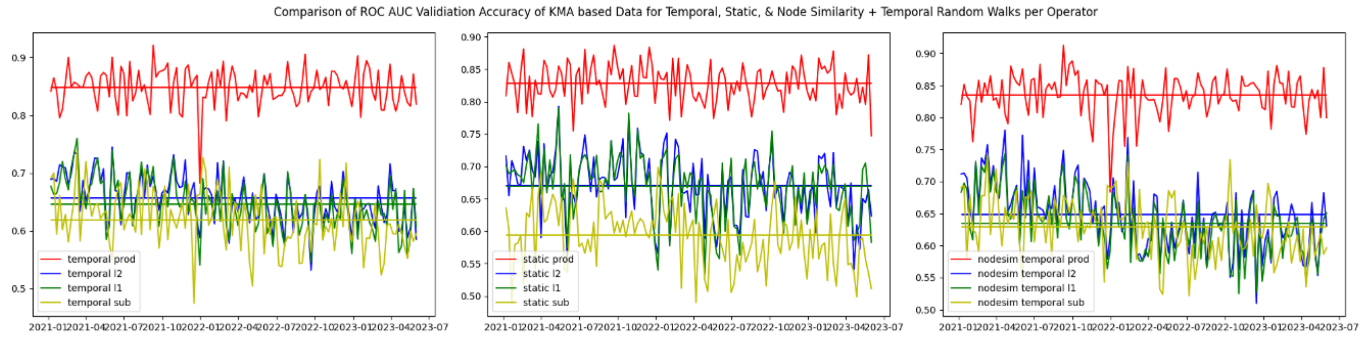
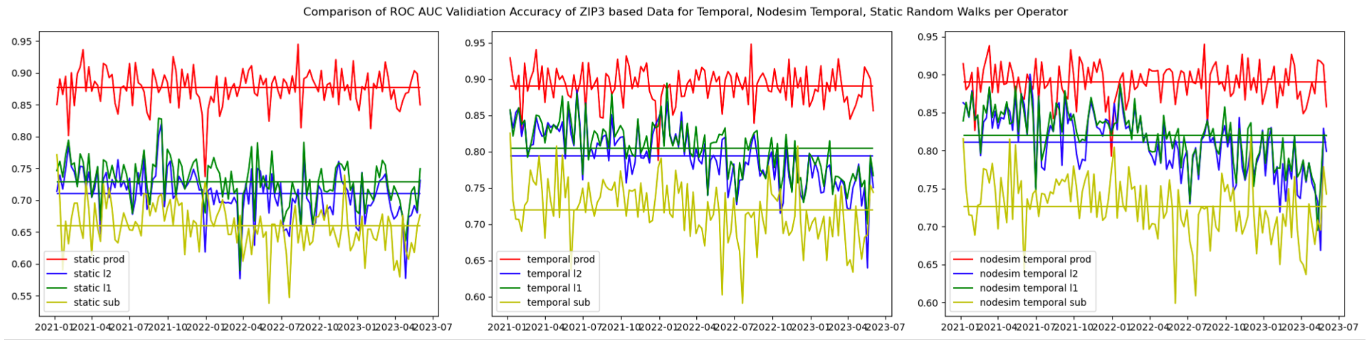
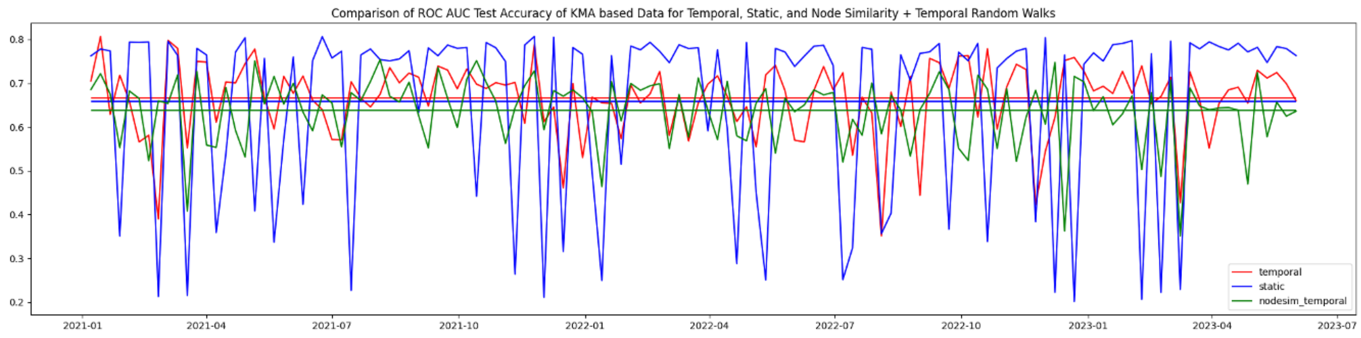


Figure 5: Validation accuracy of prediction algorithms per operator on ZIP3 Level Data



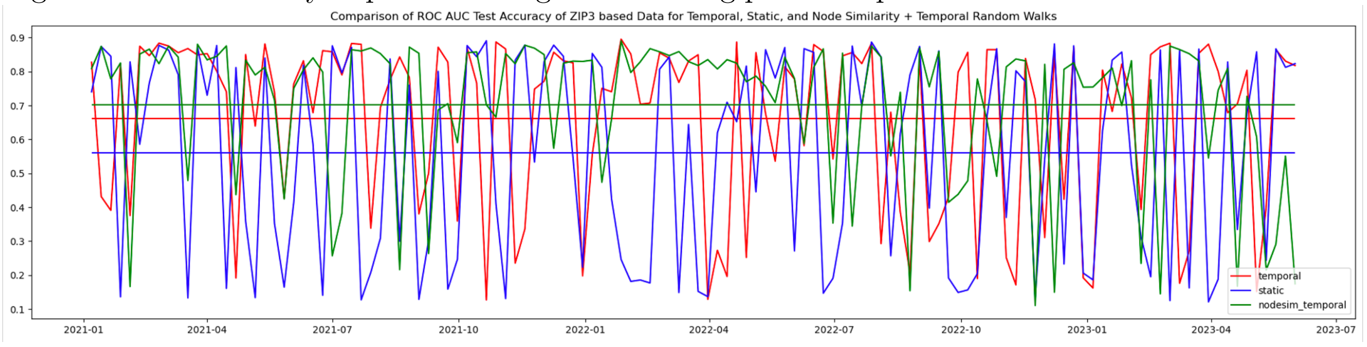
Then, using the product operator, we observed that in KMA level, link prediction based on temporal random walk with exponential weighting outperforms based on static random walk and temporal random walk with node-similarity based weighting.

Figure 6: Test accuracy of prediction algorithms using product operator on KMA Level Data



However, in ZIP3 level, link prediction based on temporal random walk with node-similarity weighting outperforms based on static random walk and temporal random walk with exponential based weighting.

Figure 7: Test accuracy of prediction algorithms using product operator on ZIP3 Level Data



4.3 Decision Making Algorithm

Now, using link predicition based on temporal random walk with exponential weighting, we compute probabilities for each potential edge existing in the new graph. Then, we compared the cumulative regret of hedgge, weighted majority algorithm, and naive threshold (= 0.5) as well as the cumulative counts of false positive and false negatives.

We observed that in KMA level, both weighted majority algorithm and hedge algorithm outperforms the naive threshold over time. However, weighted majority algorithm showed faster convergence to sublinear regret than hedge, as it almost immediately started having lower cumulative regret than naive threshold, while hedge algorithm took almost the entirety of 126 weeks to reach a sublinear regret.

Figure 8: Regret comparison of Hedge Algorithm and Naive Threshold - KMA

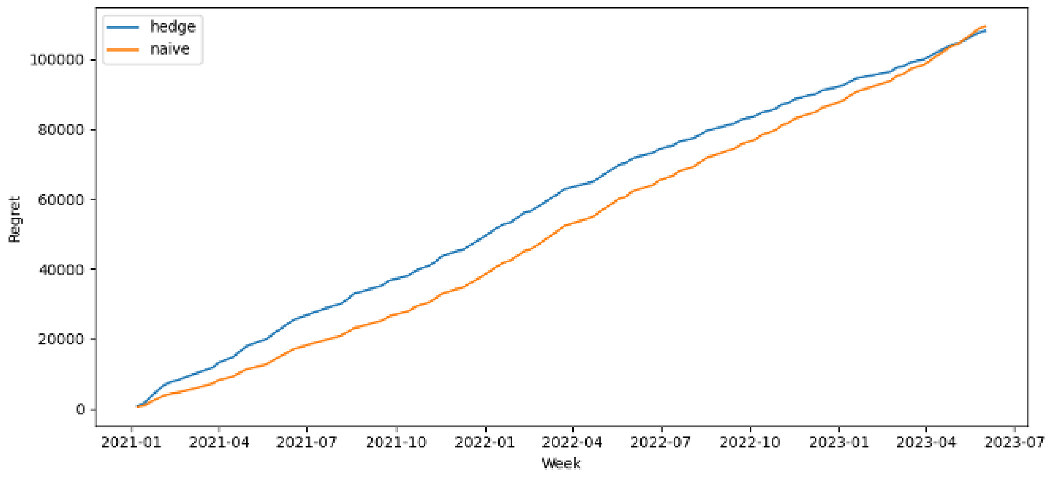
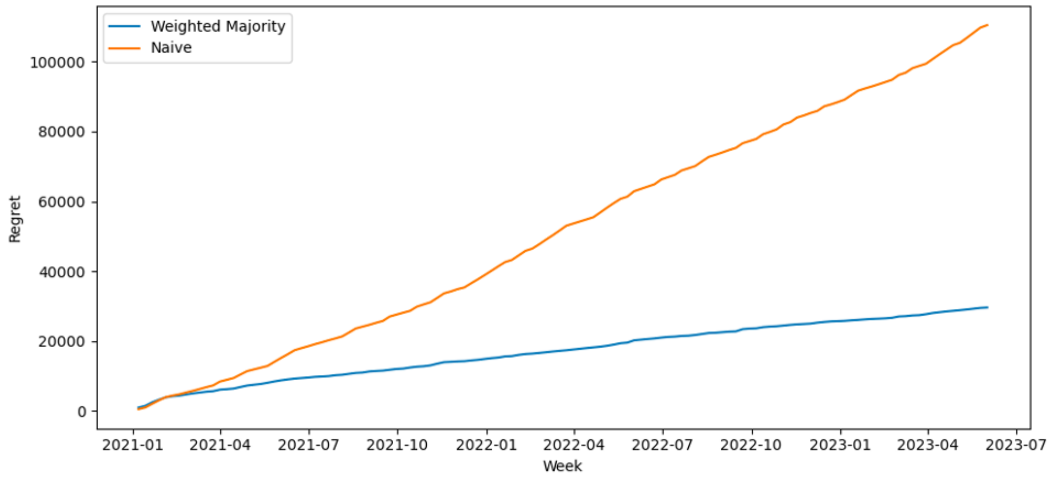


Figure 9: Regret comparison of Weighted Majority and Naive Threshold - KMA



In ZIP3 level, this difference between weighted majority algorithm and hedge algorithm becomes even clearer. Weighted majority outperforms the naive threshold over time while hedge failed to. Similar to the KMA level data, weighted majority algorithm showed fast convergence to sublinear regret than hedge, as it almost immediately started having lower cumulative regret than naive threshold.

Figure 10: Regret comparison of Hedge Algorithm and Naive Threshold - ZIP3

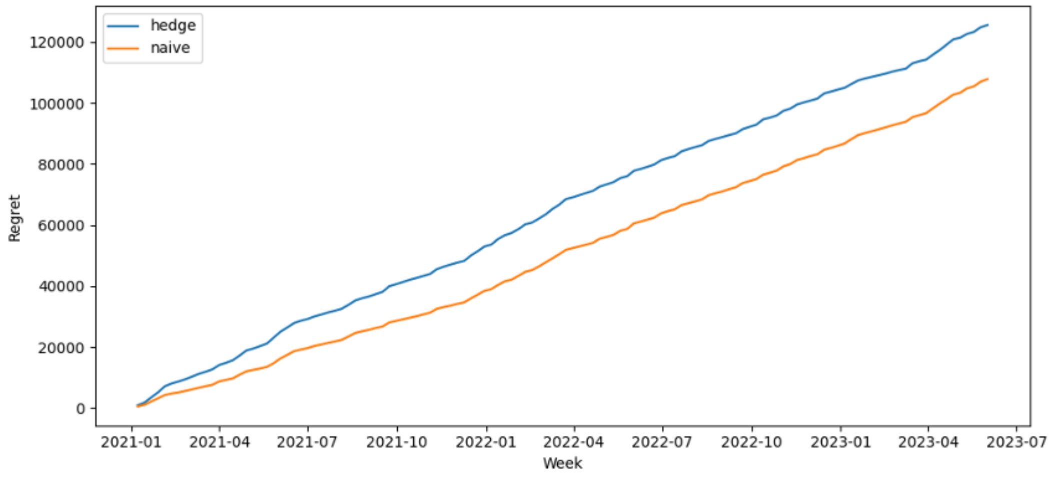
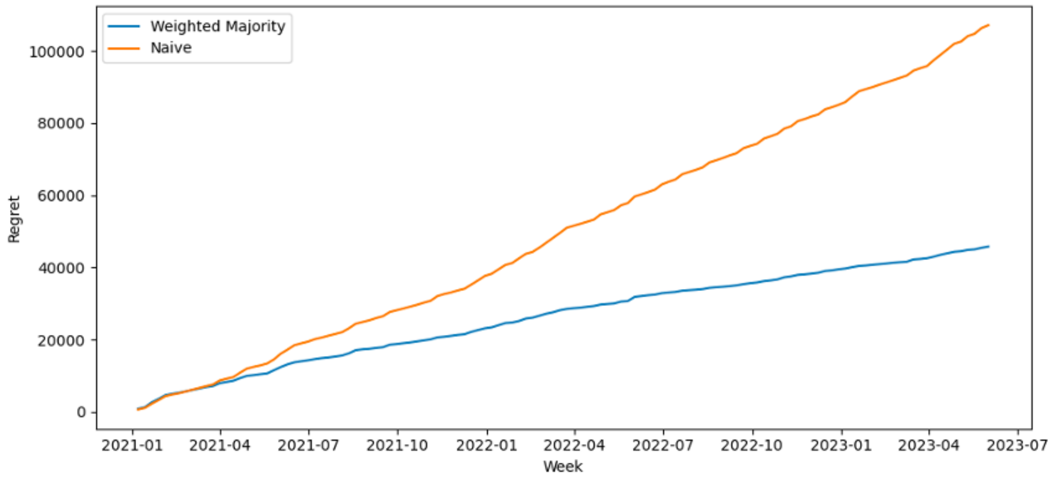


Figure 11: Regret comparison of Weighted Majority and Naive Threshold - ZIP3



Moreover, we also identified that both hedge and weighted majority algorithm outperformed naive threshold in terms of cumulative false positive counts, when we applied harsher punishment on false positives. This is shown in both KMA and ZIP3 level aggregated data.

Figure 12: False Positive Comparison of Hedge and Naive Threshold - KMA

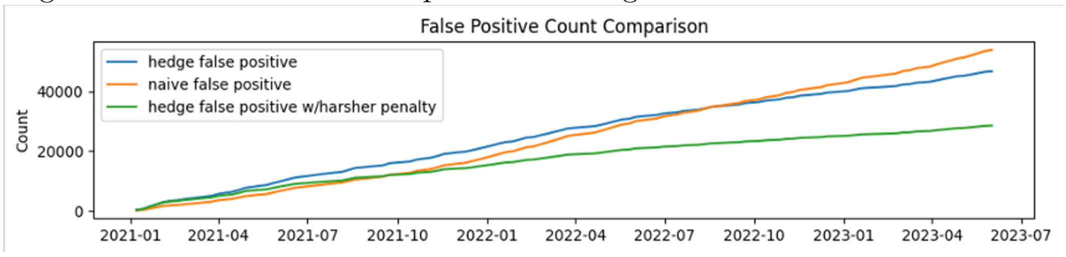


Figure 13: False Positive Comparison of Weighted Majority and Naive Threshold - KMA

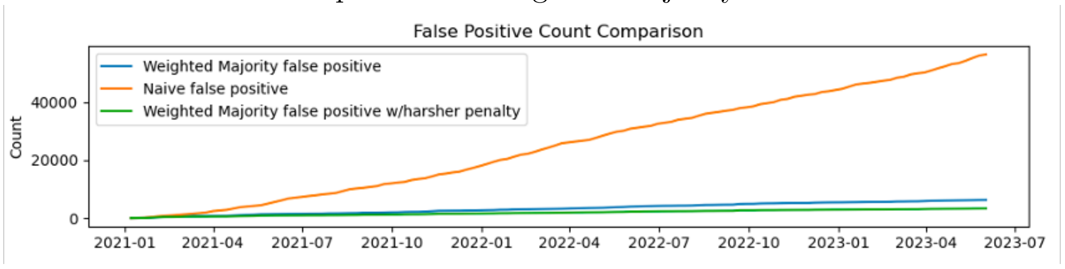


Figure 14: False Positive Comparison of Hedge and Naive Threshold - ZIP3

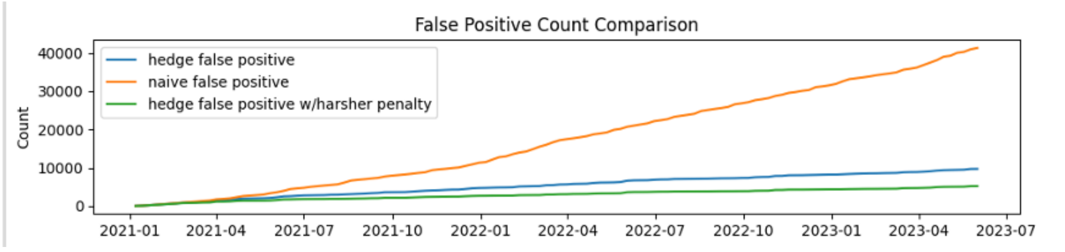
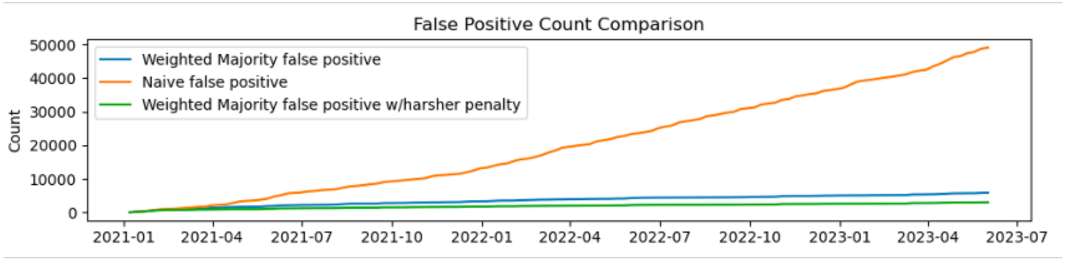


Figure 15: False Positive Comparison of Weighted Majority and Naive Threshold - ZIP3



Finally, we report the final cumulative regret and counts here.

Regret Comparison for Weighted Majority v.s. Hedge v.s. Naive Threshold		
Algorithm/Threshold	KMA	ZIP3
Naive Threshold	109750	111235
Hedge Algorithm	107649	101379
Weighted Majority Algorithm	38702	61237
Naive Threshold w/ harsher penalty	211118	193857
Hedge Algorithm w/ harsher penalty	147656	17608
Weighted Majority Algorithm w/ harsher penalty	45333	67516

False Positive Count Comparison for Weighted Majority Algorithm v.s. Naive Threshold		
Algorithm/Threshold	KMA Level	ZIP3 Level
Naive Threshold	50684	41311
Hedge Algorithm	49637	8501
Hedge Algorithm w/ harsher penalty	25007	6213
Weighted Majority Algorithm	7781	9737
Weighted Majority Algorithm w/ harsher penalty	4359	5234

False Negative Count Comparison for Weighted Majority Algorithm v.s. Naive Threshold		
Algorithm/Threshold	KMA Level	ZIP3 Level
Naive Threshold	59066	69924
Hedge Algorithm	60172	44030
Hedge Algorithm w/ harsher penalty	60204	44018
Weighted Majority Algorithm	30921	51500
Weighted Majority Algorithm w/ harsher penalty	32256	51814

True Positive Count Comparison for Weighted Majority Algorithm v.s. Naive Threshold		
Algorithm/Threshold	KMA Level	ZIP3 Level
Naive Threshold	138405	180797
Weighted Majority Algorithm	179993	213129
Weighted Majority Algorithm w/ harsher penalty	176566	208627
Hedge Algorithm	17256	21987
Hedge Algorithm w/ harsher penalty	17203	21602

5 Conclusion & Further Explorations

5.1 Conclusion

In addressing the Temporal Link Prediction Problem, our analysis highlights that the best performing node features vary depending on the specific context of the network data. At the KMA level, a Temporal Random Walk with Exponential Edge Weighting proves most effective, whereas, at the ZIP3 level, a Temporal Random Walk integrated with Node Similarity Edge Weighting emerges as superior. With the learned embedding, the study then incorporated Skip-Gram Embedding to produce node embeddings. The Product Operator stands out as the best performing operator over L2 norm, L1 norm, and subtraction operators, suggesting its utility in capturing complex node interactions over time effectively.

In the realm of decision problems, our findings affirm the robustness of the Weighted Majority Algorithm with an Exponential Update rule. This algorithm not only adapts efficiently to evolving data scenarios but also benefits significantly from the imposition of a larger penalty on false positives, thereby reducing the cost of incorrect predictions in critical decision-making processes.

5.2 Further Explorations

Given these insights, future studies could expand the multi-directed graph analysis by extending the current link prediction algorithm into predicting not only if there will be a potential edge, but also how many there will be. Research could also focus on optimizing lookback sizes to improve predictive accuracy, as we only used lookback size of five days in this research. By systematically varying the lookback period, researchers might identify the optimal temporal window that balances historical relevance with predictive performance. Finally, one could also utilize Hidden Markov Models to capture latent variables in network data. The thesis has shown that link prediction on temporal delivery order graph can be approached statistically, and, therefore, suggests some latent variables affecting the delivery orders. This approach may reveal hidden covariates or unobserved processes that influence transportation network, providing a more nuanced understanding of the underlying mechanisms.

References

[94] “The Weighted Majority Algorithm”. In: *Information and Computation* 108.2 (1994), pp. 212–261. ISSN: 0890-5401. DOI: https://doi.org/10.1006/inco.1994.1009. URL: https://www.sciencedirect.com/science/article/pii/S0890540184710091.

[FS97] Yoav Freund and Robert E Schapire. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. ISSN: 0022-0000. DOI: https://doi.org/10.1006/jcss.1997.1504. URL: https://www.sciencedirect.com/science/article/pii/S002200009791504X.

[PAS14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “DeepWalk: online learning of social representations”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD ’14. ACM, Aug. 2014. DOI: 10.1145/2623330.2623732. URL: http://dx.doi.org/10.1145/2623330.2623732.

[Ngu+18] Giang Hoang Nguyen et al. “Continuous-Time Dynamic Network Embeddings”. In: *Companion Proceedings of the The Web Conference 2018* (2018). URL: https://api.semanticscholar.org/CorpusID:13741853.

[SFP21] Akrati Saxena, George Fletcher, and Mykola Pechenizkiy. “NodeSim: Node Similarity based Network Embedding for Diverse Link Prediction”. In: (2021). arXiv: 2102.00785 [cs.SI].

[Haz23] Elad Hazan. “Introduction to Online Convex Optimization”. In: (2023). arXiv: 1909.05207 [cs.LG].

[QY23] Meng Qin and Dit-Yan Yeung. “Temporal Link Prediction: A Unified Framework, Taxonomy, and Review”. In: (2023). arXiv: 2210.08765 [cs.SI].