Analysis of Probabilistic Temporal Networks

Xiang Fu (xf74), Shangdi Yu (sy543)

May 16, 2018

1 Introduction

In most of the current literatures, the networks analyzed have deterministic edges. However, in some circumstances, we may want to know about the behavior of a non-deterministic network. For example, ride sharing network constantly has uncertain supply between different locations. When we don't know exactly what the network will look like in the future, we propose the *probabilistic temporal network* that we can encode this uncertainty into the network by giving each edge a probability of being "available" during each time slot. Interesting questions include routing and cascading in such network. Our idea of this probabilistic networks is inspired by the temporal network models, in which edges have some kind of time-relevant labeling. In our project we focus on the routing problem in a simple *probabilistic temporal network* where each edge have uniform independent probability of being "available" in each time slot.

2 Paper Summary

2.1 Network reachability of real-world contact sequences

In the paper "Network reachability of real-world contact sequences"[Hol], Holme examined four real-world contact sequence datasets on Internet communications to reach qualitative conclusions about network reachability. The author uses metric $\hat{\tau}$ and f to measure the reachability of the networks. $\hat{\tau}$ denotes the average reachability time of all pairs connected by a time-respecting path. f denotes the fraction of the node-pairs that have the above property (connected by a time respecting path between them). According to the author, at the time of this paper (2004), the shortest time to reach a node from another node at a certain time is hard to compute for all node-pairs given the large sizes of the datasets. [Hol] So the paper analyzed a random sampled subset of each of the four original datasets. The data sampled are all from a specific time interval. The random sampled datasets are characterized by node numbers, edge numbers, number of contacts, and the sampling time interval. The four datasets used in the paper all have the following characteristics: "short

path lengths, frequent contacts, and a periphery to which information is relatively unlikely to reach".[Hol] Performing five conditional uniform tests on the four datasets, the paper describes how the reachability is influenced by the characteristics of the network (such as distribution of contacts per edge, the sequence of contexts) and individual nodes and edges (such as the number of degrees).

The problem described in the paper relates to the cascading behavior problems discussed in the course. While in class we learned about the theories of the spreading and transmission of information, this paper gives insights into how the information is flowed between people, for example, through Internet communication and emails in real world.

The paper provides interesting insights into how the real world temporal network behaves; however, the paper's conclusion are largely based on the analysis of the random samples from the four datasets and there are not much mathematical proofs. The conclusions would be more convincing if the paper include some more mathematical evidence to support the claims. The behaviors observed on the four datasets do not necessarily generalize well to other networks. The four datasets all share some common characteristics such as frequent contacts and low-degree vertices, and less information reaching the peripheral nodes, but other datasets may not have these properties. Moreover, we can see from the plots in Fig3 that though the similarities described by the author exist, the $\hat{\tau}$ and f of the four datasets actually have quite different behaviors in general in the five tests. [Hol] Since the variance is very large, it is hard to say that other datasets will definitely have similar behaviors as these four datasets. Another weakness of the paper, as also mentioned by the author, is that since the data used in testings are sampled from a specific time range, the results cannot be extrapolated to the limit of large times.

The paper answers the questions "How fast can information, or disease, spread across an empirical contact network and how much of the network can be reached from one vertex through a series of contacts"[2]. The model we proposed in this reaction paper also considers how fast one can reach from one node to another in time-relevant networks. Our problem is different, though, in that the availability of the edges in our model is not deterministic and we are considering a single time-respecting path between two given nodes. In other words, unlike the transmission of information or diseases where the object being transmitted can have several copies, the object of our transmission has only one copy and cannot be split.

2.2 Paper Summary: Connectivity and Inference Problems for Temporal Networks

While the paper described in the previous section focuses on revealing the behavior of networks using real-world dataset, this paper, "Connectivity and Inference Problems for Temporal Networks" by Kempe, Kleinberg, and Kumar [KKK00], is entirely based on mathematical modeling and proofs.

The paper mainly explores two groups of problems on temporal networks: connectivity problems and infer-

ence problems. Employing the concept of subdivision and topological minor, the paper specifies a group of graphs in which Menger's Theorem holds and develops a way to find out if a graph is Mengerian and if so gives a maximum set of disjoint time-respecting paths between node pairs. It also introduces an algorithm that reconstructs a partially specified time labeling of a temporal network.[KKK00]

While some real world scenarios where the result of the paper can be applied are mentioned by the authors, the paper does not talk much about how often would a graph encoding the real world information actually be Mengerian. In other words, how likely will a real-world network have this property. Since finding the disjoint time-respecting paths would be valuable to problems like "Scheduled Transportation Networks" suggested by the paper, it would be helpful to know how often will we be able to apply the algorithm developed in the paper to a real world network and find these paths.

The paper gives an interesting extension to the various kinds of networks we learned in class. While the networks we talked about in lectures all have edges that can be used across the whole time horizon, the temporal networks in this paper are the special cases where the edges can only be used or a path is only feasible if the time labels of the edges satisfy certain conditions. This time sensitive property inspires our modeling of a "probabilistic temporal network".

In the paper, the reduction of a temporal graph to a standard directed graph allows the following question regarding a temporal network to be answered : "Starting at node u at time t, how early can one reach node v using a time-respecting path?"[KKK00]. We aim to answer to a similar question on a probabilistic temporal network: in a probabilistic temporal network, how many time slots do we need to reach node t from node s using only available edges in each time slot. The details of the model will be described in the following sections.

3 Model Description

We propose a probabilistic network model with temporal dynamic. We say a *probabilistic temporal network* PTN(G,p), where underlying graph $G = (V, E)$ is a directed graph and p is a probability, is a sequence of directed graph: $G_1, G_2, ..., G_t, ...$, where each $G_t = (V_t, E_t)$ in time slot t is generated independently by letting $V_t = V$, and for each edge $e \in E$, $e \in E_t$ with probability p. We say edge e is available at time slot t if $e \in E_t$. A small example is given below:

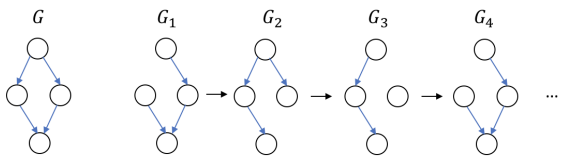

Figure 1: A simple example of the probabilistic temporal network.

Now that given the definition of the probabilistic temporal network, we are interested in the routing problem under the following settings: Say we start from node s and we want to get node t, assuming there exists a path from s to t in the underlaying graph. At each time slot, we are only able to travel through one available edge to the next node. For example, assume now we are at node i in graph G_t (so we are at time slot t), and we have edge $(i, j) \in E_t$. If we decide to move from node i to node j during time slot t, then in time slot $t+1$, we will be at node j in the graph G_{t+1}. In other words, we are only able to move to a subset of nodes S_{it} in time slot t where $S_{it} = \{v : (i, v) \in E_t\}$, given we are at node i in time slot t. There is a **traveling cost** $c_{vw} \geq 0$ incurred if we travel on the edge (v, w) and a **stalling cost** $c_{vv} \geq 0$ incurred if we do not move and stay at the node v for one time slot before we get node t. We can think stalling as, there exists a self loop edge (v, v) for each node $v \neq t$ that shows up at every time slot and the cost for that edge is c_{vv}. The traveling/stalling costs are incurred in each time slot and add up cumulatively as we travel through the networks.

Now routing in this probabilistic temporal network becomes a nontrivial problem because the edges are not deterministic anymore. Assuming in the underlying graph G = (V,E), t is reachable from s. Then in the probabilistic temporal network, we can get t from s for some **total cost** T. Consider the shortest s-t path P in the underlaying graph G given by Dijkstra, we will have in the probabilistic temporal network, $T \geq \sum_{e \in P} c_e$. Let $T^* = \sum_{e \in P} c_e$, which is actually the minimum total traveling cost from s to t. Note that T includes not only the traveling costs, but also the stalling costs. Given some routing policy A, we will get the expected total cost T_A under this policy. We define the **delay ratio** as $\frac{T_A}{T}$ so that we can compare performance on different graphs. The graphs are in Simulation Result section with $\sqrt{delay\ ratio}$ as x-axis. The square root is applied so that the graph looks more compact. Now in the probabilistic temporal network, we might not be able to take the shortest path in every time slot as the graph is changing in each time slot and the edge for the shortest path may not be available in the time slot we need to take it. We can either stall and pay the stalling cost c_{vv} or we can change our route. If we change our route and deviate from the optimal path in the underlying graph, we will end up taking a longer path and pay extra traveling costs; however, if we stick to the shortest path in the underlying graph and wait for the edges in the path to be available, we need to pay the stalling cost. There is no way to know in advance which edges will be available in the future time slots, so we have to face this trade off. Since we do not know the exact total cost from s to t before we finish our route, we can only estimate the cost of different routes. We are interested in finding good routing policies in order to minimize the expected total cost for getting node t from node s.

4 Conditional Optimal Routing Algorithm

4.1 Algorithm Description

The optimal routing policy minimize the expected total cost to travel from node s to node t. We define $w(v)$ to be the expected total cost under the optimal routing policy to travel from node v to node t for all $v \in V$. The conditional optimal routing algorithm is only optimal when the following assumption holds:

Assumption 1.

$$c_{vv} \leq c_{xy}, \forall v, x, y \in V, x \neq y$$

That is, all stalling costs are less than or equal to the minimum traveling cost.

Only when this assumption holds, does the graph has a topological order in terms of w. That is, when this assumption holds, in the optimal routing policy, a node v with $w(v)$ and neighbor $N(v)$ will only want to go to nodes with smaller w values. This makes the dynamic program approach feasible: the value of $w(v)$ only depends on the values of w's for other nodes that are smaller than $w(v)$. We also have $w(t) = 0$ as the base case.

Now assume the assumption holds. The optimal routing policy works as the following and we will prove the correctness of this algorithm in the next section.

Say in some time slot i we are at node v, the routing policy will give the next node we will be at in time slot $i + 1$. We will check the possible destination in the graph G_i. We will only want to go to node with smaller (expected total cost to t + cost(v to that node)) than (expected total cost from v to t + cost(stalling at v)) from node v otherwise we would rater stay at node v. And we will always want to go the the node with smallest (expected total cost to t + cost(v to that node)) if that edge is available in G_i. That is, among all the possible destinations for node v in time slot i(always includes node v itself), we will always choose the node x with smallest $w(x) + c_{vx}$ as the node we will be at in time slot i+1.

There is an example later furthering explain the routing algorithm. Now we can formulate the following stochastic dynamic program 1 to calculate $w(v)$ for all $v \in V$ with the input to be the underlying graph G. For the simplicity of our algorithm, we define the following notations:

- Define $N(v) = \{x : (v, x) \in \mathcal{E} \text{ and } x \neq v\}$: $x \in N(v)$ if $x \neq v$ and there is a directed edge (v, x) in $\mathcal{E}$. $N(v)$ is just the possible destinations for node v (other than staying at v).
- Say we are at node v in the current time slot and we are moving under the optimal routing policy, denote $P(x)$ as the probability that v will go to node x. Note that the probability that we will stay at v is just $P(v) = 1 - \sum_{x \in N(v), c_{vx} + w[x] < c_{vv} + w[v]} P(x)$.

For example, for the underlying graph G in the following figure 2, if we are currently at the node s and we are trying to go to node t, we will have $N(s) = \{a, b, c\}$. Also we can stay at node s. For all possible destinations for s, sort all the nodes according to (the expected total cost from that node to t + cost(s to that node)), we will have $a < b < s < c$. So when we are at node s, we will never want to go to node c. If the edge (s, a) is available, we will always want to go to node a. if the edge (s, a) is not available and the edge (s, b) is available, we will always want to go to node b. If both edge (s, a) and edge (s, b) are not available, we will just stay at node s as we will never want to go to node c.

In this case, we have:

$$P(a) = \mathbb{P}(\text{edge (s,a) available}) = p$$

$$\begin{aligned} P(b) &= \mathbb{P}(\text{edge (s,a) not available and edge (s,b) available}) \\ &= \mathbb{P}(\text{(s,a) not available})\mathbb{P}(\text{(s,b) available}) = (1-p)p \end{aligned}$$

$$P(v) = \mathbb{P}(\text{stay at node v}) = 1 - \sum_{x \in N(v), c_{vx}+w[x] < c_{vv}+w[v]} P(x) = 1 - p - p(1-p)$$

So according to our routing policy, we can get the expected total cost from node s to node t, $w(s)$, by solving the equation:

$$w(s) = p(w(a)+1) + p(1-p)(w(b)+2) + (1-p-p(1-p))(w(s)+3)$$

In general for any node v, we can solve for $w(v)$ by solving the equation:

$$w(v) = \sum_{x \in N(v), c_{vx}+w(x) < c_{vv}+w(v)} (c_{vx} + w(x))P(x) + (1 - \sum_{x \in N(v), c_{vx}+w(x) < c_{vv}+w(v)} P(x))(c_{vv} + w(v))$$

which gives:

$$w(v) = \frac{\sum_{x \in N(v), c_{vx}+w(x) < c_{vv}+w(v)} (c_{vx} + w(x))P(x) + (1 - \sum_{x \in N(v), c_{vx}+w(x) < c_{vv}+w(v)} P(x))c_{vv}}{\sum_{x \in N(v), c_{vx}+w(x) < c_{vv}+w(v)} P(x)}$$

And we will be able to solve for w(a) and w(b) following the similar procedure. Note the expected total cost from node t to node t is just 0. That is, $w(t) = 0$.

After we calculate $w(v)$ for all node $v \in V$, we can make routing decision according to the policy stated above: Among all the available destinations(including the current node itself), we always go to the node with the smallest expected total cost(w value).

It's not hard to notice that, in order to calculate $w(v)$ correctly for some node v, we will need to know the set: $\{x \in N(v), c_v x + w(x) < c_v v + w(v)\}$. How can we know this set before we know w(v)? In order to solve this problem we will use the dynamic programming algorithm 1, in which we initialize $w(t) = 0$ and $w(v) = \infty$ for all $v \in V, v \neq t$, and then keep updating w for every node according to the formula given above. If $w(v)$ doesn't change for all nodes $v \in V$, we have w converged. We then stop the algorithm and output w, which is the correct expected total cost under optimal policy for each $v \in V$. We will prove that

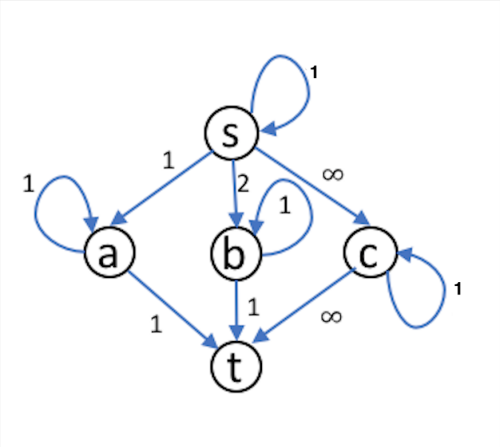

Figure 2: An example underlying graph. The traveling cost/stalling cost is marked along each edge.

this algorithm will converge within finite iterations and prove the correctness of this algorithm in the next section.

Algorithm 1 Generating Optimal Routing Table

Set w[t] = 0, set w[v] = ∞ for all $v \in V, v \neq t$
$flag = 1$
while $flag$ **do**
    $flag = 0$
    for all $v \in V$ **do**
        $w'[v] \leftarrow \frac{\sum_{x \in N(v), c_{vx}+w[x]<c_{vv}+w[v]}(c_{vx}+w[x])P(x)+(1-\sum_{x \in N(v), c_{vx}+w[x]<c_{vv}+w[v]} P(x))c_{vv}}{\sum_{x \in N(v), c_{vx}+w[x]<c_{vv}+w[v]} P(x)}$
        if $w'[v] < w[v]$ **then**
            $flag = 1$
            $w[v] \leftarrow w'[v]$
return w

4.2 Algorithm Correctness

We give the full details of proofs in the appendix, and only sketch the construction here.

Lemma 1. *Given policy: In some time slot i, say we are currently at node v, among all the available destinations* ($\{v, x : (v, x) \in \mathcal{E}_i\} \cap (N(v) \cup \{v\})$)*, we select the node with the smallest sum of cost and w value to be the node we will be at in time slot (i+1), say this node is x. We claim that our algorithm calculates the expected value under this policy and under this policy, we will not move to any of the nodes* $\{x_{large} : w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}\}$ *in the next time slot.*

Lemma 2. *We say w(v) correct if it is indeed the expected cost under the "going to smallest" policy we*

described above. We claim our algorithm gives the $(i+1)^{th}$ smallest w(v) to node v in i^{th} iteration(starting from iteration 0). And if we fail to calculate the correct w value for some node v, we must have overestimated it.

Lemma 3. *We won't change the label of a node if we have labeled it correctly.*

Lemma 4. *Recall our routing policy: In some time slot i, say we are currently at node $v \neq t$, among all the available destinations ($\{v, x : (v, x) \in \mathcal{E}_i\} \cap (N(v) \cup \{v\})$), we select the node x with the smallest $w(x) + c_{vx}$ to be the node we will be at in time slot (i+1).*
Claim: calculate the expected cost form each node to t under this policy gives w(v) for all node $v \in V, v \neq t$: the smallest expected total cost from each node $v \in V$ to the destination node t. i.e.: our routing policy is optimal.

Theorem 1. *The optimal routing algorithm gives the routing policy with smallest expected cost and converges in finite steps.*

Proof. Firstly, we show if the algorithm is correct, the algorithm terminates in finite time.
By Lemma 2, we will get one node correct at each step. By Lemma 3, we won't change the label of a node if the node has been labeled with the true $w(v)$. As a result, in each step, the number of nodes labeled correctly will increase by at least 1 in each step. Since we have n nodes, in at most n steps, we will have correctly labeled all nodes.
Correspondingly in Algorithm 1 line 7-9, we only change flag to 1 if we update the label of a node. After we correctly labeled all nodes, in the next iteration, the condition in line 3 "while flag = 1" does not hold. The algorithm converges and thus stops. At this time, all nodes are labeled with the true $w(v)$. By definition, all nodes are labeled with the smallest expected cost to go from this node to destination node. Secondly, we prove by induction that this routing algorithm is indeed optimal. □

5 Heuristic Routing Algorithms

For a routing policy A, we say the expected total cost using the routing policy A is T_A. And we say the *Delay* of policy A is $T_A - T^*$. So even the optimal routing policy for the problem will have positive delay if $p < 1$. We start by analyzing the following heuristic policies to investigate the routing behavior of the probabilistic temporal network. We define $l(v)$ to be the total cost(length) of the shortest path from v to t and $l(v) = \infty$ if there doesn't exist a path from v to t. Say at time slot t, we are at node i, and we have the available destinations $S_{it} = \{v : (i, v) \in E_t\}$ to move to or we can stall.

5.1 Take Available Shortest Path policy (TASP)

Under the Take Available Shortest Path policy, we will stall only when $S_{it} = \emptyset$ or $\forall v \in S_{it}$, t is not reachable by v. We will move to node v^* where:

$$v^* = \begin{cases} argmin_{v \in S_{it}} l(v) + c_{iv} & \text{if } \exists v \in E_t \text{ such that } l(v) + c_{iv} \text{ is finite} \\ i & \text{otherwise} \end{cases}$$

Under this policy, we try our best to avoid stalling.

5.2 Always Wait policy (AW)

Under the Always Wait policy, we will always stall when we are not able to take a shortest path in the underlying graph G. The next node v^* to go to ("go to" the current node itself if stall) is given by the expression:

$$v^* = \begin{cases} v' & \text{if } \exists v' \in E_t \text{ such that } c_{iv} + l(v') = l(i)) \\ i & \text{otherwise} \end{cases}$$

Under this policy, we will always keep stalling if we are not able to take a shortest path in the underlying graph. If there exists unique shortest path in the underlying graph, we will have the number of stall under this policy follows a negative binomial distribution: say the there are q edges on the shortest path form s to t, number of stall $= NB(q, 1-p)$. So the expected number of stall is just $\frac{(1-p)q}{p}$. So the expected delay is $\frac{(1-p)q\xi}{p}$.

There might exist Dynamic Programming Algorithm for the optimal routing policy. We will investigate the formulation for the dynamic program and its performance or conditional bound in details in the course project.

- Define the policy that minimize E[X] for all nodes to be the optimal policy O.
- Define $N(v) = \{x : (v, x) \in \mathcal{E} \; and \; x \neq v\}$: $x \in N(v)$ if $x \neq v$ and there is a directed edge (v, x) in $\mathcal{E}$.
- Define $P(x)$ to be the probability that in any time slot in the optimal policy v will go to node x. Note that $P(v) = 1 - \sum_{x \in N(v), c_{vx} + w[x] < c_{vv} + w[v]} P(x)$ (Give some examples)

Solve the equation :

$$w'[v] = \sum_{x \in N(v), c_{vx} + w[x] < c_{vv} + w[v]]} (c_{vx} + w[x])P(x) + (1 - \sum_{x \in N(v), c_{vx} + w[x] < c_{vv} + w[v]} P(x))(c_{vv} + w'[v])$$

to get $w'[v]$

6 Simulation Result

We simulated the two heuristics(AW and TSAP) and the optimal algorithm on random k-degree graphs with different parameters. The result of simulation shows that our optimal algorithm consistently performs well in different graphs. Simulation also shows that when assumption 1 does not hold, our policy still serves as a good heuristic.

Though the AW policy performs well in the random k-degree graphs, this is not necessarily the case in other graphs. We will show below by giving an example that the AW policy will perform extremely bad in some graphs.

For all simulations, we test each algorithm on 500 random graphs. Let D be the cost of the shortest path in the underlying graph, and D' be the actual cost to go from s to t in one simulation iteration. We define the delay rate as d = (D'-D)/D. The x-axis of the graphs is $\sqrt{d}$. The y-axis are the frequency of the delay rates out of 500 iterations per policy.

In our simulations on k-degree random graphs, the AW performs really well. We think its due the following reasons:

- According to the assumption we make, the "stall cost" is relatively small. The AW policy takes advantage from this assumption as AW policy only pay extra stalling cost and never pay extra traveling cost.
- The k-degree random graphs have small-world property, so there tends to be a very short path from the source node to the destination node. As we can observe in a lot of cases OPT even makes the same decisions as the AW policy.
- OPT guarantees smallest expected cost, but the variance can be very high according to simulation. So when the cost of self loops are very small, intuitively AW is a competitive policy and has very low variance, this is the same result we get in simulation. The delay rate is bounded below by 0. Though the variance is large, for OPT we cannot extrapolate to negative. So we observe that in some simulations AW policy even have slightly better numbers than OPT.

OPT guarantees smallest expected cost, but the variance can be very high according to simulation. So when the cost of self loops are very small, intuitively AW is a competitive policy and has very low variance, this is the same result we get in simulation. The the cost of self loop is small, the OPT policy will have similar or even worse cost. The delay rate is bounded below by 0. Though the variance is large, we cannot extrapolate to negative.

6.1 AW and OPT performs significantly better than TASP

In this section, we show that the performance of TASP in 3-degree random graphs is significantly worse than the AW policy and the OPT policy. Let p denote the probability that an edge is available in a certain timeslot. For each $p = 0.1, 0.2, \ldots, 0.8, 0.9$, we run each of the three algorithms, TASP, AW, and OPT, on 500 randomly generated 3-degree graphs.

From the result of the simulation, we can see that when p is mall, TASP's delay rates are significantly larger. As p goes to 1, the distribution of the delay rates of TASP becomes more and more like that of the other two policies. This means that when the edges are likely to be available, TASP will make similar decision as the other two policies. This is what we expected because when p is large, the shortest path in the underlying graph is more likely to be available; notice that AW policy will only take paths that are shortest in the underlying graph.

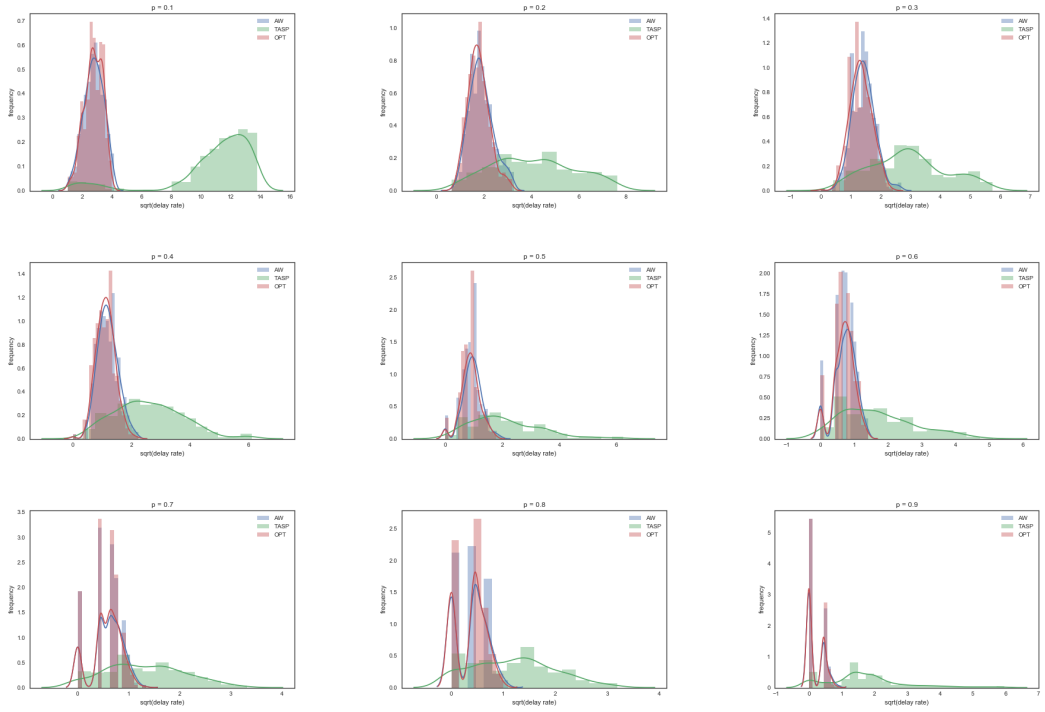

Figure 3: AW, TASP, and OPT run on 500 3-degree random graphs. The number of nodes n is 100 and all edge costs are 1. The probability of each edge p $\in [0.1, 0.9]$

Another noticeable pattern in Figure 3 is that the delay rates of the OPT and AW policy are very similar. We will show later that this is not always the case - in some graphs, AW policy will make very different decisions as the OPT policy. Here, according to our simulation results, AW policy produces very similar delay rates as OPT does in k-degree random graphs. We discussed the reason at the beginning of this section. In fact, it is even possible that under some seeds, the AW will make exactly the same decision as OPT in

all 500 randomly generated temporal graphs. One example is Figure 4. The red(OPT) and blue(AW) areas overlap and show a color of purple.

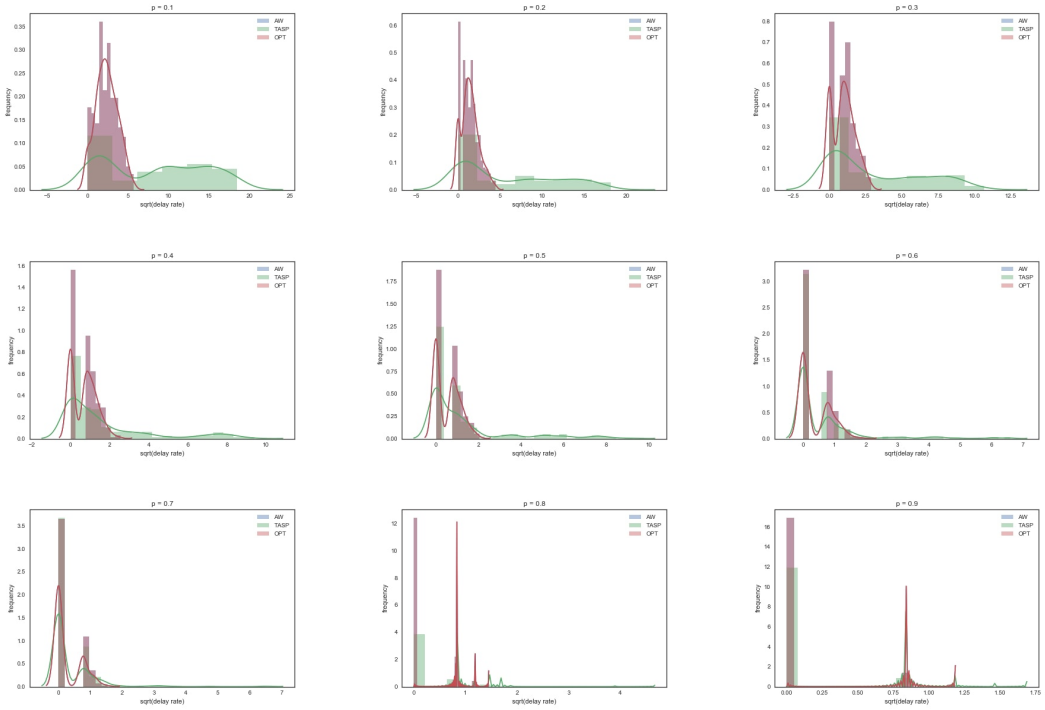

Figure 4: AW, TASP, and OPT run on 500 3-degree random graphs. The cost of each edge(except self-loops) is a random number between 1 and 2. The cost of self-loops are between 1 and the minimum non-self-loop edge cost. The number of nodes n is 500. The probability of each edge p $\in [0.1, 0.9]$ seed = 666.

Figure 5 below are graphs that contain only AW and OPT policy. We exclude the TASP because we showed in 3 that TASP has significantly larger delay rates. Removing TASP from the plots allows us to plot the delay rates of AW and OPT more compactly. The plots in Figure 5 again show that AW has similar performance as OPT in k-degree random graphs when our assumption 1 holds.

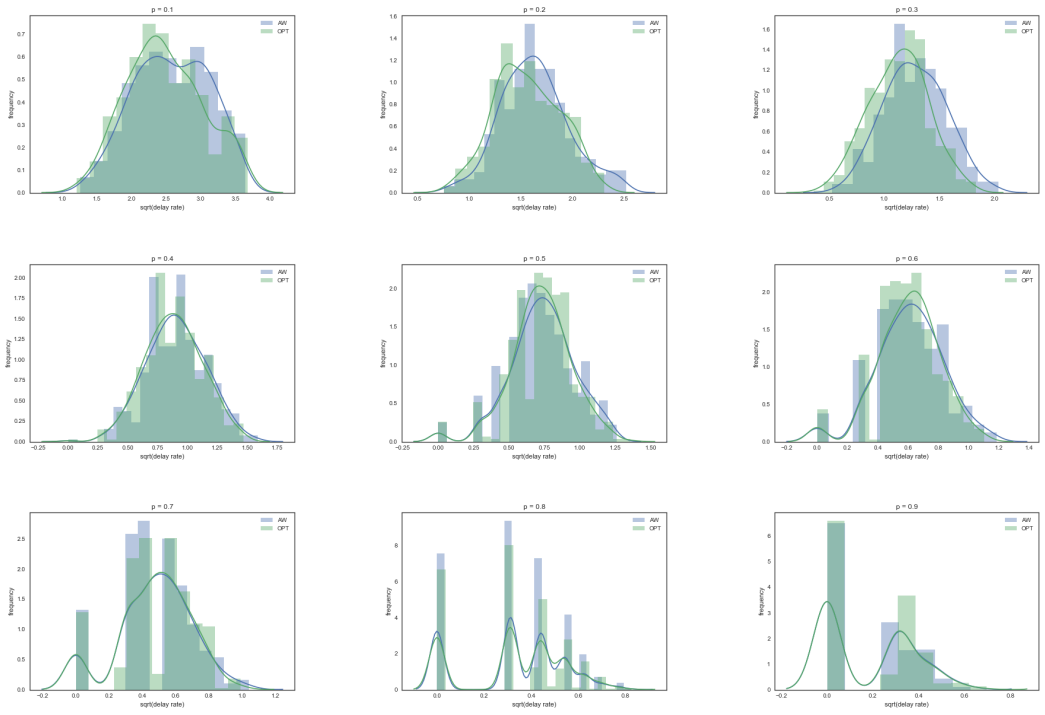

Figure 5: AW and OPT run on 500 2-degree random graphs. The cost of each edge(except self-loops) is a random number between 1 and 2. The cost of self-loops are between 1 and the minimum non-self-loop edge cost. The number of nodes n is 100. The probability of each edge p $\in [0.1, 0.9]$ seed $= 5$.

6.2 PTN Instance that AW Policy performs badly

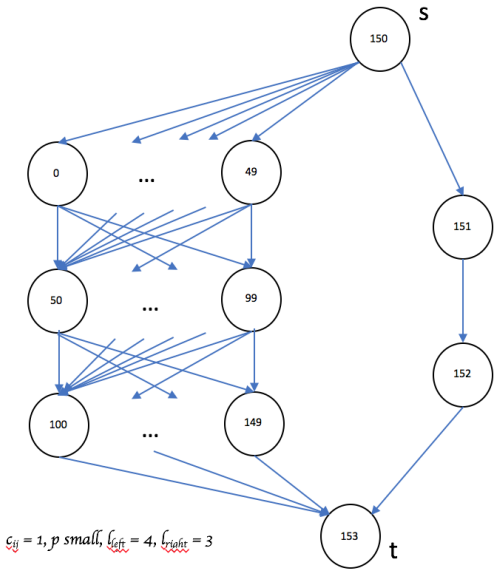

Figure 6: PTN Instance that AW Policy performs badly. All traveling/stalling cost = 1.

For a PTN with underlying graph as above. The AW policy, which performs pretty well in random k-degree graph will perform really badly when p is small. Here in the graph there is a "broad way" on the left with distance 4 to the destination node t and a "narrow way" of length 3 on the right to the destination node t. The AW policy will always choose to take the left "narrow way".

Here, the "broad way" contains 3 layers of nodes, each contains 50 nodes, all connected to every node in the next layer; s can go to all 50 nodes in the first layer and ultimately the third layer can all go to the destination node. On the "borad way" the probability of stalling in each step is $(1-p)^{50}$, which is very small. While the "narrow way" is just one way, where the probability of stalling in each step is $(1-p)$. So we can expect a lot of stalls on the narrow way when p is small. Therefore, we can expect the AW policy performs badly in this PTN instance when p is small. We have the following simulation results:

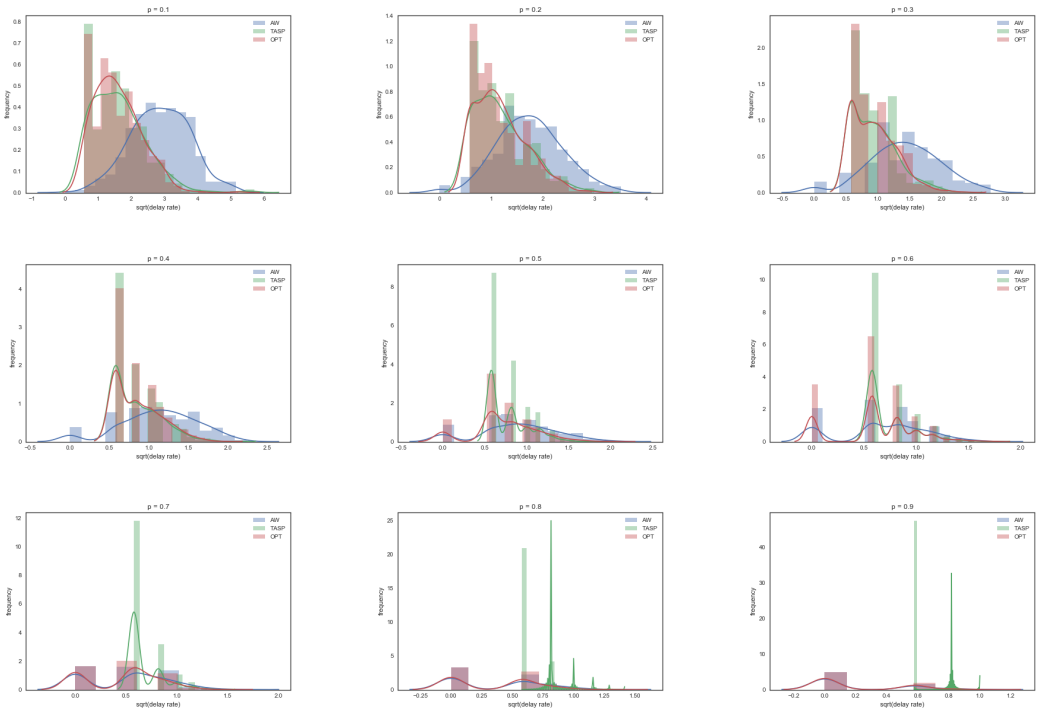

Figure 7: PTN Instance that AW Policy performs badly, graph as shown above. AW policy performs badly when p is small.

As we expected, when p is small, the AW policy has much larger delay rate than the optimal policy and the TASP policy. As p increase, the gap between the AW policy and the other two policies narrows. Since the assumption 1 is satisfied, the optimal algorithm always performs well in this instance, no matter what p is.

6.3 OPT as a Heuristic when "Self Loop Minimum" Assumption Fails

Our OPT algorithm requires that the "stalling costs" cannot be larger than any "traveling cost" as this requirement is necessary for us to find a topological order and perform the stochastic dynamic programming algorithm. Nevertheless, the result of our simulation shows that our OPT algorithm still performs relatively well when the assumption is violated. As shown in Figure 8, our OPT algorithm produce smaller delay rates in general under all levels of p.

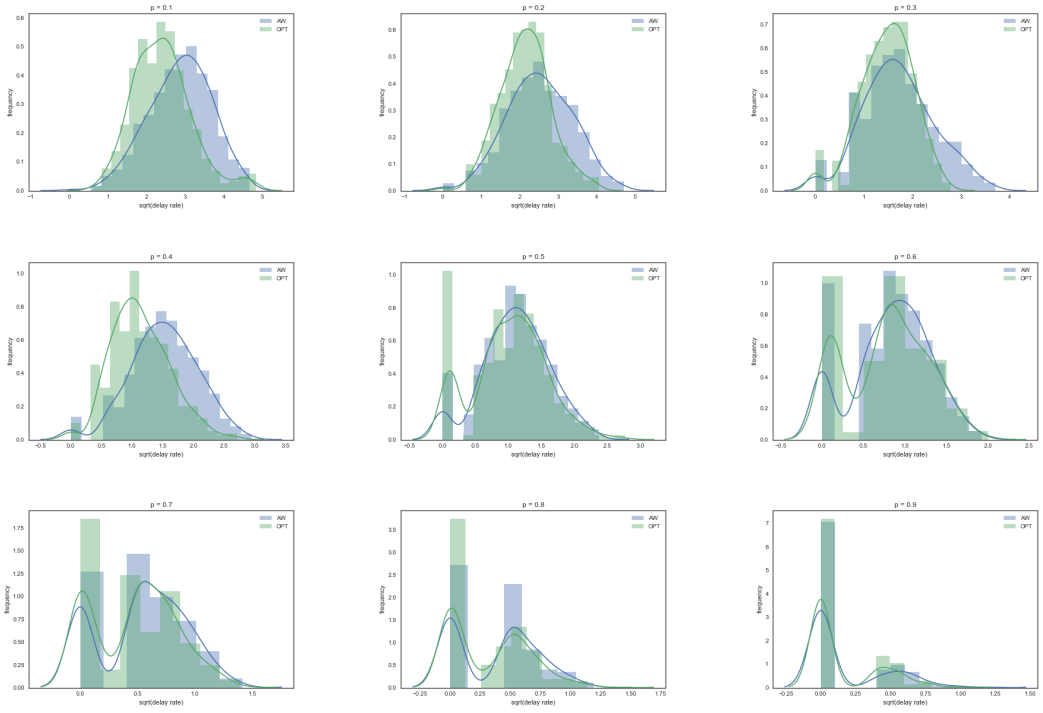

Figure 8: AW and OPT run on 500 3-degree random graphs. The cost of each edge(including self-loops) is a random number between 1 and 5. The number of nodes n is 30. The probability of each edge p $\in [0.1, 0.9]$. Note that assumption 1 is violated.

6.4 Distribution of $\sqrt{\text{delay rate}}$ has a Normal Shape

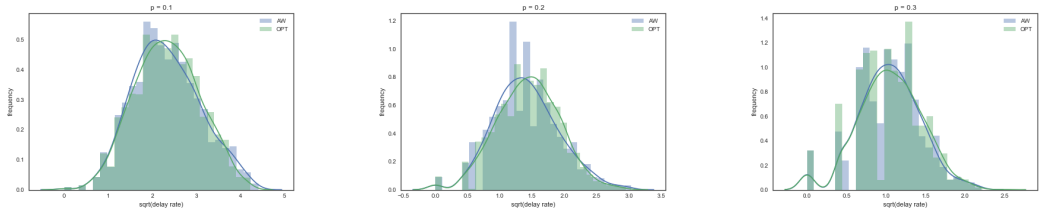

Figure 9: AW and OPT run on 2000 3-degree 25-node random graphs. The cost of each edge(except self-loops) is a random number between 1 and 2. The cost of self-loops are between 1 and the minimum non-self-loop edge cost. The probability of each edge p $\in [0.1, 0.3]$.

According to our simulation the result, the distribution of the square root of the delay rate on a k-degree random graph is approximately normal when p is small.

7 Conclusion

In our project, we review several papers relevant to temporal graphs and propose the probabilistic temporal network (PTN) model. In the PTN model, in different time slots, edges are not always available. Each edge is available in each time slot with a probability. This model captures the uncertainty in a lot of real-life networks: such like ride-sharing network, information propagation network, and transportation network, etc. We investigate the routing problem in PTNs. We propose two heuristic routing policies and one conditional optimal routing policy. We prove the conditional optimal routing policy is indeed optimal when the assumption 1 holds. We run simulations on k-degree random graphs and some special graphs with these 3 routing policies and get some insight about routing in PTNs.

Some future directions that extends our current work include: (1) running simulations on random graphs that do not have the small world property; (2) developing an algorithm that finds the smallest expected cost of going from one node to another when the "stall cost smallest" assumption does not hold; (3) investigating other problems on probabilistic temporal networks, such as cascading problem. (4) Different probability models; for example, the adjacency matrix can be modeled as a Markov Chain.

8 Appendix: Proof

Lemma 1. *Given policy: In some time slot i, say we are currently at node v, among all the available destinations ($\{v, x : (v, x) \in \mathcal{E}_i\} \cap (N(v) \cup \{v\})$), we select the node with the smallest sum of cost and w value to be the node we will be at in time slot (i+1), say this node is x. We claim that our algorithm calculates the expected value under this policy and under this policy, we will not move to any of the nodes $\{x_{large} : w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}\}$ in the next time slot.*

Proof. First, we define

$N_{large}(v) = \{x_{large} : w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}, (v, x_{large}) \in \mathcal{E}\}$,

$N_{small}(v) = \{x_{small} : w(x_{small}) + c_{vx_{small}} < w(v) + c_{vv}, (v, x_{small}) \in \mathcal{E}\}$.

E_{small} := the set of events that at least one of the edges $\{(v, x_{small})\}$ is available.

E_{large} := the set of events that none of those edges is available.

$A = E_{small} \cup E_{large}$. Notice that P(A)=1 since the two events are the complement of each other.

Let $\mathcal{E}_a$ be the edges available in event $a \in A$.

$$w(v) = \sum_{a \in A} P(a) * min_{(v,x) \in \mathcal{E}_a} w(x) + c_{vx}$$
$$= \sum_{a \in E_{small}} P(a) * min_{(v,x) \in \mathcal{E}_a} c_{vx} + w(x) + \sum_{a \in E_{large}} P(a) * min_{(v,x) \in \mathcal{E}_a} c_{vx} + w(x)$$

Claim. For each node $v \in V$, we have

$$\frac{\sum_{a \in E_{small}} P(a) * min_{(v,x)+c_{vx}) \in \mathcal{E}_a} w(x) + (1 - \sum_{a \in E_{small}} P(a)) * c_{vv}}{\sum_{a \in E_{small}} P(a)}$$

Note that according to Assumption 1, we have the inequality $w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}$ always holds. and under our policy we always go to the available node with the smallest sum of cost and w value. So according to the inequality above, when events in E_{large} happens, we always stay at the current node. Then we have:

$$w(v) = \sum_{a \in E_{small}} P(a) * min_{(v,x) \in \mathcal{E}_a} (c_{vx} + w(x)) + \sum_{a \in E_{large}} P(a) * (c_{vv} + w(v))$$
$$= \sum_{a \in E_{small}} P(a) * min_{(v,x) \in \mathcal{E}_a} (c_{vx} + w(x)) + (1 - \sum_{a \in E_{small}} P(a)) * (c_{vv} + w(v))$$
$$= \sum_{a \in E_{small}} P(a) * (min_{(v,x)) \in \mathcal{E}_a} (w(x) + c_{vx}) + (1 - \sum_{a \in E_{small}} P(a)) * (w(v) + c_{vv})$$
$$= \frac{\sum_{a \in E_{small}} P(a) * min_{(v,x)+c_{vx}) \in \mathcal{E}_a} w(x) + (1 - \sum_{a \in E_{small}} P(a)) * c_{vv}}{\sum_{a \in E_{small}} P(a)}$$

This is exactly the recurrence we used in our algorithm.
Since $w(x_{small}) + c_{vx_{small}} < w(v) + c_{vv} \leq w(x_{large}) + c_{vx_{large}}$, we know $min_{(v,x) \in \mathcal{E}_a} w(x) + c_{vx}) \neq w(x_{large}) + c_{vx_{large}}$ $\forall x_{large}$.

So when an edge to x_{small} is available, we won't take any of x_{large}. And when none of the edges to x_{small} is available, we will take the self loop. Thus, our policy will only go to nodes that has has smaller minimum expected value. So under this policy, we will not move to any of the nodes $\{x_{large} : w(x_{large}) + c_{vx_{large}} \geq w(v) + c_{vv}\}$ in the next time slot. □

Lemma 2. *We say w(v) correct if it is indeed the expected cost under the "going to smallest" policy we described above. We claim our algorithm gives the $(i+1)^{th}$ smallest w(v) to node v in i^{th} iteration(starting from iteration 0). And if we fail to calculate the correct w value for some node v, we must have overestimated it.*

Proof. We prove the correctness of this Lemma by showing the statement: $P(i)$ = "The $(i+1)^{th}$ smallest

w(v) is given to node v in i^{th} iteration, where that value for a node x is:

$$w(x) = \frac{\sum_{a \in E_{small}} P(a) * min_{(v,x)+c_{vx}) \in \mathcal{E}_a} w(x) + (1 - \sum_{a \in E_{small}} P(a)) * c_{vv}}{\sum_{a \in E_{small}} P(a)}$$

. And if we fail to calculate the correct w value for some node v in i^{th} iteration, we must have overestimated it." is true for all $i = 0..n$. We prove this by strong induction.

Inductive Hypothesis:

$P(i)$ = "The $(i+1)^{th}$ smallest $w(v)$ is given to node v in i^{th} iteration. If we fail to calculate the correct w value for some node v in i^{th} iteration, we must have overestimated it." is true for all $i = 0..k$.

Base Case:

The base case when $k = 0$ is true because the 1^{st} smallest $w(v)$ is 0 where v is the destination node t. When we reached destination node, we don't need to pay the stalling cost anymore. In the initialization, we set $w(t) = 0$ explicitly. So in the 0^{th} iteration, we have given the 1^{st} smallest $w(v) = 0$ for $v = t$.

We initialize all other nodes' w values to ∞ in iteration 0. So we must have either (1) correctly initialized w when w is indeed ∞ or, (2) overestimated all the w values in iteration 0.

Case i = k + 1:

Suppose we are at node v in the current iteration. We sort all nodes according to $w(v_i) + c(x, v_i)$, from the smallest to the largest and get the sequence $v_1, v_2, ..., v_n$.

By inductive hypothesis, we know that in k^{th} iteration, we have labeled nodes $v_1, v_2, ..., v_{k+1}$ correctly, where $w(v_1) \leq w(v_2) \leq ...w(v_k) \leq w(v_{k+1}) \leq w(v_{k+1})$. By Lemma 1, our routing algorithm will not move to any of the nodes $\{x : c(v_{k+1}, x) + w(x) \geq c(v_{k+1}, v_{k+1}) + w(v_{k+1})\}$. This is equivalent of saying the value $w(v_{k+1})$ only depends on the nodes $v_1, v_2, ..., v_k$ and indeed our algorithm only uses the w values of these nodes when calculating $w(v_{k+1})$. So the $(k+1)^{th}$ smallest w(v) is calculated from the $1^{st}...(k)^{th}$ smallest w values, which by inductive hypothesis are correct at this iteration. As a result, the $w(v_{k+1})$ is correctly calculated at this iteration.

By inductive hypothesis, all w values that are not calculated correctly in previous iterations are over estimated. So when we calculate $w(x) = \frac{\sum_{a \in E_{small}} P(a) * min_{(v,x)+c_{vx}) \in \mathcal{E}_a} w(x) + (1 - \sum_{a \in E_{small}} P(a)) * c_{vv}}{\sum_{a \in E_{small}} P(a)}$, the w values we used are either correct or overestimated. Since the probabilities are non negative, we either correctly calculate w(x) or also overestimated w(x) in this iteration.

□

Lemma 3. *We won't change the label of a node if we have labeled it correctly.*

Proof. By line 7-9 in the algorithm, we only update the label of a node if the new label is smaller than the current label. As we showed in Lemma 2, we never underestimate the label of any node. So since every update we make decrease the label and we never underestimate the label, we never change the label of a node if we have labeled it correctly. □

Lemma 4. *Recall our routing policy: In some time slot i, say we are currently at node $v \neq t$, among all the available destinations ($\{v, x : (v, x) \in \mathcal{E}_i\} \cap (N(v) \cup \{v\})$), we select the node x with the smallest $w(x) + c_{vx}$ to be the node we will be at in time slot (i+1).*
Claim: calculate the expected cost form each node to t under this policy gives w(v) for all node $v \in V, v \neq t$: the smallest expected total cost from each node $v \in V$ to the destination node t. i.e.: our routing policy is optimal.

Proof. We prove this claim by strong induction. Say the true smallest expected total cost from node $v \in V$ to node t is $c_s(v)$. We sort all nodes in V according to $c_s(v)$, from the smallest to the largest and get the sequence $v_1, v_2, ..., v_n$, where we actually know $v_1 = t$. Then we just want to show: $w(v) = c_s(v)$ for all $v \in V$.
Base Case: We want to show $w(v_1) = c_s(v_1)$. This is true since we know $v_1 = t$ and we initialize $w(t) = 0$, which is indeed true since if we are already at node t there is no cost to get node t.
Inductive Step: Inductive Hypothesis: $w(v_j) = c_s(v_j)$ for $j = 1, ..., k$. Want to show: $w(v_j) = c_s(v_j)$ for $j = 1, ..., k + 1$.
We have $w(v_j) = c_s(v_j)$ for $j = 1, ..., k$ according to the inductive hypothesis. We are only left showing that $w(v_{k+1}) = c_s(v_{k+1})$.
According to Lemma 1, Lemma 2, when we are currently at node v_{k+1}, we will not go to node $v \in \{v_{k+2}, ..., v_n\}$ in the next time slot, and thus ($w(v_{k+1})$)doesn't depend on these nodes. The value of $w(v_{k+1})$ only depends on $w(v_1), ..., w(v_k)$. Without the loss of generality: Say in some time slot i, there are available destinations: $\{x_1, ..., x_q\} = \{v, x : (v, x) \in \mathcal{E}_i\} \cap (N(v) \cup \{v\})$ where $x_1, ..., x_q$ are sorted according to the $c_v x + w(x)$ value, from the smallest to the largest. Then our policy will choose to go to x_1.
We prove by contradiction that our policy gives the minimum expected cost.
Note that the expected cost from node v to node t $E[v] = \min_{x \in \{available destination\}} c(v, x) = c_s(x)$. Say if our policy does not give the minimum expected cost in this case. there exists some destination $x_l \neq x_1$, where $1 < l \leq q$ such that $c_{vx_l} + c_s(x_l) < c_{vx_1} + c_s(x_1)$. According to our inductive hypothesis we have $c_s(x_l) = w(x_l)$ and $c_s(x_1) = w(x_1)$. So we have $c_{vx_l} + w(x_l) < c_{vx_1} + w(x_1)$. But we know that $x_1, ..., x_q$ are sorted according to the $c_v x + w(x)$ value, from the smallest to the largest. So we have a contradiction.
Then since our policy gives the minimum expected cost for every node in every possible temporal graph G_i, and we also know $w(v_j) = c_s(v_j)$ for $j = 1, ..., k$, we have $w(v_{k+1})$ and the true smallest expected cost form node v_{k+1} to node t is given by:

$$w(v_{k+1}) = c_s(v_{k+1}) = \frac{\sum_{x \in N(v), c_{vx} + < c_{vv} + w(v)} (c_{vx} + w(x)) P(x) + (1 - \sum_{x \in N(v), c_{vx} + w(x) < c_{vv} + w(v)} P(x)) c_{vv}}{\sum_{x \in N(v), c_{vx} + w(x) < c_{vv} + w(v)} P(x)}$$

. □

9 Appendix: Mean and Variance of Simulation

Note: the mean and variance are the mean and variance of delay rate, not the mean and variance of the square-rooted values.

Figure 3:

	Mean			Variance		
p	AW	TASP	OPT	AW	TASP	OPT
0.1	8.09	127	7.94	13.1	2350	10.6
0.2	3.82	19.8	3.36	4.31	224	10.6
0.3	2.17	10.2	1.92	1.29	62.7	3.07
0.4	1.54	8.02	1.36	0.767	46.1	1.01
0.5	1.02	5.81	0.89	0.893	43.4	0.352
0.6	0.647	4.08	0.580	0.191	22.1	0.160
0.7	0.432	2.33	0.406	0.109	5.09	0.0952
0.8	0.240	2.04	0.220	0.0570	4.58	0.0454
0.9	0.106	4.37	0.0976	0.0247	36.2	0.0193

Table 1: Simulation results for 500 iterations (mean and variance for delay rate) for k-degree random graph: n = 100, k = 3, edge cost 1, seed = 5, p range from 0.1 to 0.9

Figure 4:

	Mean			Variance		
p	AW	TASP	OPT	AW	TASP	OPT
0.1	7.39	105	7.39	47.4	11600	47.4
0.2	3.11	67.1	3.11	14.0	7810	14.0
0.3	1.51	19.9	1.51	2.96	784	2.96
0.4	0.841	8.27	0.841	1.11	317	1.11
0.5	0.596	5.68	0.596	0.619	197	0.619
0.6	0.363	1.78	0.363	0.311	33.9	0.311
0.7	0.240	0.654	0.240	0.180	9.62	0.180
0.8	0.158	0.295	0.158	0.137	1.73	0.137
0.9	0.0703	0.0847	0.0703	0.0564	0.101	0.0564

Table 2: Simulation results for 500 iterations (mean and variance for delay rate) for k-degree random graph: n = 100, k = 3, cost range from 1 to 2, seed = 666, p range from 0.1 to 0.9

Figure 7

	Mean			Variance		
p	AW	TASP	OPT	AW	TASP	OPT
0.1	9.04	3.30	3.024	26.9	12.8	7.32
0.2	3.47	1.68	1.624	5.35	2.23	1.87
0.3	2.34	1.02	1.00	2.53	0.632	0.586
0.4	1.51	0.785	0.810	1.09	0.337	0.329
0.5	1.02	0.661	0.692	0.698	0.208	0.361
0.6	0.647	0.587	0.523	0.321	0.166	0.237
0.7	0.421	0.483	0.347	0.189	0.0768	0.113
0.8	0.267	0.424	0.230	0.100	0.0460	0.0694
0.9	0.129	0.365	0.117	0.0468	0.0114	0.0364

Table 3: PTN Instance that AW policy performs badly

Figure 8

	Mean		Variance	
p	AW	OPT	AW	OPT
0.1	8.88	6.34	20.5	16.6
0.2	7.07	5.04	18.8	8.64
0.3	3.48	2.48	7.26	2.40
0.4	2.61	1.52	2.78	1.47
0.5	1.50	1.35	1.26	1.44
0.6	0.917	0.848	0.564	0.627
0.7	0.437	0.354	0.165	0.136
0.8	0.257	0.208	0.0809	0.0741
0.9	0.108	0.0823	0.0428	0.0233

Table 4: Simulation results for 500 iterations (mean and variance for delay rate) for k-degree random graph: n = 30, k = 3, cost range from 1 to 5, seed = 94275, p range from 0.1 to 0.9

Figure 5

	Mean		Variance	
p	AW	OPT	AW	OPT
0.1	7.21	6.72	8.31	8.18
0.2	2.77	2.53	1.25	1.01
0.3	1.71	1.35	0.537	0.372
0.4	0.878	0.836	0.209	0.182
0.5	0.597	0.573	0.107	0.0916
0.6	0.427	0.409	0.0716	0.0570
0.7	0.281	0.274	0.0427	0.0371
0.8	0.156	0.157	0.0182	0.0181
0.9	0.0692	0.0688	0.00727	0.00738

Table 5: Simulation results for 500 iterations (mean and variance for delay rate) for k-degree random graph: n = 100, k = 2, cost range from 1 to 2, seed = 5, p range from 0.1 to 0.9

References

[Hol] P. Holme. Network reachability of real-world contact sequences.

[KKK00] David Kempe, Jon Kleinberg, and Amit Kumar. Connectivity and inference problems for temporal networks. pages 504–513, 2000.

Special thanks to Professor Chris De Sa who suggested using the stochastic dynamic programming method