Workplace Recommendation with Temporal Network Objectives: Proofs

Let $G = (V, E)$ be a strongly connected temporal graph whose last edge arrives at t.

Minimum latency in-edges (MLI): given a target node v and a time $t' \leq t$, find k sources $u_1, \ldots, u_k$ such that adding edges $\{(u_i, v, t', t'+1) \mid i = 1, \ldots, k\}$ minimizes latency(G_{t+1}).

Minimum latency out-edges (MLO): given a source node u and a time $t' \leq t$, find k targets $v_1, \ldots, v_k$ such that adding edges $\{(u, v_i, t', t'+1) \mid i = 1, \ldots, k\}$ minimizes latency(G_{t+1}).

Myopic minimum latency in-edges (MMLI): given a target node v, find k sources $u_1, \ldots, u_k$ such that adding edges $\{(u_i, v, t, t+1) \mid i = 1, \ldots, k\}$ minimizes latency(G_{t+1}).

Myopic minimum latency out-edges (MMLO): given a source node u, find k targets $v_1, \ldots, v_k$ such that adding edges $\{(u, v_i, t, t+1) \mid i = 1, \ldots, k\}$ minimizes latency(G_{t+1}).

Note that for all four of these problems, we can equivalently thinking about maximizing the view sum instead of minimizing the latency. We can also define the equivalent edge addition problems for total information, in which we maximize totalInf(G_{t+1}) instead of minimizing latency(G_{t+1}):

Maximum total information in-edges (MTII): given a target node v, time $t' \leq t$, weight w, and decay parameter λ, find k sources $u_1, \ldots, u_k$ such that adding edges $\{(u_i, v, t', t'+1, w) \mid i = 1, \ldots, k\}$ maximizes totalInf(G_{t+1}).

Maximum total information out-edges (MTIO): given a source node u, time $t' \leq t$, weight w, and decay parameter λ, find k targets $v_1, \ldots, v_k$ such that adding edges $\{(u, v_i, t', t'+1, w) \mid i = 1, \ldots, k\}$ maximizes totalInf(G_{t+1}).

Myopic maximum total information in-edges (MMTII): given a target node v, weight w, and decay parameter λ, find k sources $u_1, \ldots, u_k$ such that adding edges $\{(u_i, v, t, t+1, w) \mid i = 1, \ldots, k\}$ maximizes totalInf(G_{t+1}).

Myopic maximum total information out-edges (MMTIO): given a source node u, weight w, and decay parameter λ, find k targets $v_1, \ldots, v_k$ such that adding edges $\{(u, v_i, t, t+1, w) \mid i = 1, \ldots, k\}$ maximizes totalInf(G_{t+1}).

Theorem 1. MLO *is NP-hard.*

Proof. By reduction from VERTEX COVER. Let $G = (V, E), k$ be an instance of VERTEX COVER. Construct a temporal graph G' with all of the nodes in V plus an additional node u^* and additional nodes w_{uv} for each edge $\{u, v\} \in E$. For each undirected edge $\{u, v\} \in E$, add the directed temporal edges $(u, w_{uv}, 3, 4)$ and $(v, w_{uv}, 3, 4)$ to G'. Additionally, add the edges $(v, u^*, 0, 1)$ and $(u^*, v, 1, 2)$ to G' for each $v \in V$ and the edges $(w_{uv}, u^*, 0, 1)$ and $(u^*, w_{uv}, 1, 2)$ for each $\{u, v\} \in E$. Now consider the instance of MLO with temporal graph G', source node u^*, and budget k at time $t' = 2$. We will recover a vertex cover of G of size k from an optimal solution to this MLO instance, or verify that none exists.

First, note that G' is strongly connected due to the edges through u^* departing at $t = 0$ and $t = 1$. After those edges, all original nodes and w-nodes have views of each other at $t = 0$ and views of u^* at $t = 1$. Additionally, u^* has views of all nodes at $t = 0$. After the edges departing at $t = 3$, w_{uv}-nodes have views of u and v at $t = 3$. Thus, if we add no edges, the final view sum is $7|E| + |V|$: each original edge contributes two views at $t = 3$ and one view at $t = 1$, while each original node has one view at $t = 1$.

Suppose there is a vertex cover of G of size k. If we use the nodes in this cover as the MLO targets at $t = 2$, then these k nodes have views of u^* at $t = 2$ instead of $t = 1$. Additionally, the w-node of any edge

with an endpoint in the node set also receives this view after the edges departing at $t = 3$—of course, this includes every edge, since the nodes are a vertex cover. Thus, the final view sum increases by $k + |E|$ for a total of $8|E| + |V| + k$. We can show that this is an optimal MLO solution. First, adding an edge from u^* to a w-node only increases the final view sum by 1. On the other hand, adding an edge from u^* to an original node increases the final view sum by 1+ the number of edges that node covers. An MLO solution with k nodes therefore increases the final view sum by k+ the number of edges covered by the original nodes in the solution. The best possible MLO solution thus increases the final view sum by $k + |E|$ by covering all edges, which is exactly what the vertex cover solution does. This line of reasoning also implies that if there is no vertex cover of G of size k, then the optimal MLO solution is worse than $8|E| + |V| + k$, since we cannot reach the $k + |E|$ improvement in final view sum.

Finally, note that the size of the MLO instance we created is polynomial in the size of the VERTEX COVER instance: G' has $|V| + |E| + 1$ nodes and $2(|V| + |E|) + 2|E|$ temporal edges. Therefore, if we could solve MLO in polynomial time, we could determine whether there is a size k vertex cover of G by performing the construction and checking if the MLO solution reached the optimal value $8|E| + |V| + k$. MLO is therefore NP-hard by the hardness of VERTEX COVER. □

Theorem 2. MMLI *is NP-hard.*

Proof. By reduction from SET COVER. Let $(S_1, \ldots, S_n \subseteq \mathcal{U}, k)$ be an instance of SET COVER. Create nodes u_i for each $i \in \mathcal{U}$ and nodes u_S for each subset $S \in \{S_1, \ldots, S_n\}$. Also create a target node v for the MMLI instance. To make the graph strongly connected, add edges $(u, v, 0, 1)$ and $(v, u, 1, 2)$ for every node $u \neq v$. Add an edge $(u_i, u_S, 2, 3)$ for each subset $S \in \{S_1, \ldots, S_n\}$ and for each for each element $i \in S$.

We'll show that if a set cover of size k exists, then it corresponds to an optimal MMLI solution, and that we can identify from an optimal MMLI solution if no set cover of size k exists.

Consider the views in the MMLI instance we constructed, before adding any of the k edges into v. After the edges departing at $t = 0$ and $t = 1$, every node other than v has a view of v at 1 and every node has views of all the non-v nodes at 0 (disregarding self-views). After the edges departing at $t = 2$, every set node u_S has views at 2 of its element nodes u_i for each $i \in S$. If we add an edge from an element node u_i into v, we will therefore only update v's view of u_i to 2. On the other hand, if we add an edge from a set node u_S to v, we will update v's view of u_S as well as all of the element nodes u_i for $i \in S$, all to 2. There is thus always an optimal solution that only selects in-edges from set nodes, since they give at least as much benefit to the final view sum as element nodes. Moreover, if we select k set nodes, the number of view updates to 2 that we get is k+ the number of elements covered by those set nodes. The best we can possibly do for MMLI is thus to select k set nodes whose sets cover $\mathcal{U}$, which gives the final view sum

$$\underbrace{n + |\mathcal{U}|}_{t=1 \text{ edges}} + \underbrace{2\sum_{S_i} |S_i|}_{t=2 \text{ edges}} + \underbrace{2k + 2|\mathcal{U}|}_{\text{set cover edges}}.$$

If no set cover exists, then the optimal MMLI solution will be strictly smaller, since we will get fewer view updates at v by missing uncovered elements.

Finally, the construction has size polynomial in the size of the SET COVER instance, since we created $n + |\mathcal{U}| + 1$ nodes and $2n + 2|\mathcal{U}| + \sum_{S_i} |S_i|$ edges. We can therefore use a polynomial-time algorithm for MMLI to solve SET COVER in polynomial time. Thus, MMLI is NP-hard. □

Note that MMLI is a special case of MLI where $t' = t$. We therefore have the following corollary of Theorem 2:

Corollary 1. MLI *is NP-hard.*

We now prove that the greedy algorithm is a constant-factor approximation for MLO, MLI, and MMLI.

Lemma 1. *Let $f(S)$ be the final view sum of G if we add the temporal edges $(u, v, t, t+1)$ for each $(u, v) \in S$ to G. $f(S)$ is monotone and submodular.*

Proof. Let $f_{uv}(S)$ be the term of $f(S)$ corresponding to v's view of u, with $\sum_{u \in G} \sum_{v \in G} f_{uv}(S) = f(S)$. We'll show that $f(S) + f(S \cup \{(i,j),(h,\ell)\}) \leq f(S \cup \{(i,j)\}) + f(S \cup \{(h,\ell)\})$ by showing that $f_{uv}(S) + f_{uv}(S \cup \{(i,j),(h,\ell)\}) \leq f_{uv}(S \cup \{(i,j)\}) + f_{uv}(S \cup \{(h,\ell)\})$ for every u, v. Consider what happens to the final views when we add the edge $e_{ij} = (i,j,t,t+1)$ to S. Some subset of the views increase due to the new edge, those for which the latest departure time path uses e_{ij}. Similarly, when we add $e_{h\ell} = (h,\ell,t,t+1)$ to S, some subset of the views increase. However, when we add both e_{ij} and $e_{h\ell}$, notice that no path can use both of them, since they both depart at t and arrive at $t+1$. Thus, the latest departure time path from u to v either

1. uses e_{ij}, in which case $f_{uv}(S \cup \{(i,j),(h,\ell)\}) = f_{uv}(S \cup \{(i,j)\})$,
2. uses $e_{h\ell}$, in which case $f_{uv}(S \cup \{(i,j),(h,\ell)\}) = f_{uv}(S \cup \{(h,\ell)\})$, or
3. uses neither e_{ij} nor $e_{h\ell}$, in which case $f_{uv}(S \cup \{(i,j),(h,\ell)\}) = f_{uv}(S)$.

Additionally, adding an edge can never decrease a view (i.e., $f(S)$ and $f_{uv}(S)$ are monotone), meaning that $f_{uv}(S \cup \{(i,j)\}) \geq f_{uv}(S)$ (ditto for (h,ℓ)). Combining the above observations yields $f_{uv}(S) + f_{uv}(S \cup \{(i,j),(h,\ell)\}) \leq f_{uv}(S \cup \{(i,j)\}) + f_{uv}(S \cup \{(h,\ell)\})$, since $f_{uv}(S \cup \{(i,j),(h,\ell)\}) = \max(f_{uv}(S \cup \{(i,j)\}), f_{uv}(S \cup \{(h,\ell)\}))$. □

Theorem 3. *The greedy algorithm $(1-\frac{1}{e})$-approximates the optimal final view sum in* MLO*,* MLI*, and* MMLI*.*

Proof. Since $f(S)$ is monotone submodular by Lemma 1, the greedy algorithm that at each step picks the edge whose addition maximizes the final view sum is a $(1-\frac{1}{e})$-approximation to the optimal final view sum [1]. There are $O(n)$ edges whose effect we need to compute to run the greedy algorithm, and we can compute their effects in polynomial time using the single-pass latency algorithm. □

Theorem 4. *The greedy algorithm for* MMLO *is optimal.*

Proof. Unlike for MMLI, the edges we add have independent effects on the final latency: if we choose to add $(u, v_i, t, t+1)$, this can only decrease the latencies of v_i based on the views of u—it doesn't matter whether we also add $(u, v_{i'}, t, t+1)$. Thus, to maximize the latency, we just need to pick the k edges which most decrease the final latency, which is exactly what greedy does. □

Theorem 5. MMTII *is NP-hard.*

Proof. By reduction from SET COVER (the construction is very similar to the proof for MMLI). Let $(S_1, \ldots, S_n \subseteq \mathcal{U},\ k)$ be an instance of SET COVER. We construct a temporal network G for the MMTII instance. Create nodes u_i for each $i \in \mathcal{U}$ and nodes u_S for each subset $S \in \{S_1, \ldots, S_n\}$. Also create a target node v for the MMTII instance. Add a temporal edge $(u_i, u_S, 0, 1)$ with weight 1 for each subset $S \in \{S_1, \ldots, S_n\}$ and for each for each element $i \in S$.

Now consider the instance of MMTII with graph G, recipient v, budget k, $\lambda = 1$, and edge weight 1. The objective is to choose k in-edges for v $(u, v, 1, 2)$ whose addition maximizes the total information in G at $t = 2$. We'll show that if a set cover of size k exists, then it corresponds to an optimal MMTII solution, and that we can identify from an optimal MMTII solution if no set cover of size k exists.

As in the proof for MMLI, there is no reason to add in-edges coming from an item node u_i, which only adds to v's information about u_i, rather than in-edges coming from a set node u_S with $i \in S$, which adds just as much to v's information about u_i as well as u_S and all other u_j for $j \in S$. Note that information never decays (since we have $\lambda = 1$) and that a single edge suffices to saturate total information about the source, since all edge weights are 1. By adding in-edges into v, we can only increase v's information about item and set nodes. At most, we can only cause it to have information 1 about k set nodes. For each set node u_S we add, we also cause v to get information 1 about all item nodes u_i with $i \in S$. Thus, the maximum possible information gain we can hope for is $k + |\mathcal{U}|$ by picking k set nodes that cover $|\mathcal{U}|$. If a set cover exists, the MMTII solution can select in-edges from set nodes corresponding to the cover and we can verify that we get

information gain $k + |\mathcal{U}|$. If the optimal MMTII solution get strictly less information gain than $k + |\mathcal{U}|$, then no set cover of size k exists (if one did, then we would have a better MMTII solution).

Finally, this construction is polynomial in the size of the SET COVER instance. Thus, if we could efficiently solve MMTII, we could also efficiently solve SET COVER. □

As before, MMTII is a special case of MTII with $t' = t$. Thus:

Corollary 2. MTII *is NP-hard.*

Theorem 6. MTIO *is NP-hard.*

Proof. By reduction from SET COVER. Let $(S_1, \ldots, S_n \subseteq \mathcal{U}, k)$ be an instance of SET COVER. We construct a temporal network G for the MTIO instance. Create nodes u_i for each $i \in \mathcal{U}$ and nodes u_S for each subset $S \in \{S_1, \ldots, S_n\}$. Also create a source node u^* for the MTIO instance. Add a temporal edge $(u_S, u_i, 1, 2)$ with weight 1 for each subset $S \in \{S_1, \ldots, S_n\}$ and for each for each element $i \in S$.

Consider the instance of MTIO with temporal network G, source node u^*, time $t' = 0$, budget k, edge weight 1, and $\lambda = 1$. The edges departing at $t = 1$ cause each item node u_i to get information 1 about the set nodes u_S for which $i \in S$. By adding edges departing from u^* at $t = 0$, we can give nodes information 1 about u^*. Note that there is no point adding out-edges to item nodes, since that only adds to the information at that item node. If we instead add out-edges going to a set node whose set contains the item, we give both the set node and all of its contained item nodes information about u^*. Thus, an optimal MTIO solution only sends edges to set nodes. The best we can do with k out edges is to give k set nodes information about u^*, which then propagate this information to the items they cover through the edges departing at $t = 1$. If a set cover exists, we can therefore increase information by $k + |\mathcal{U}|$ by sending out-edges from u^* to the set nodes corresponding to the set cover. If the optimal MTIO solution increases total information by less than $k + |\mathcal{U}|$, then no set cover exists.

Since the construction has size polynomial in the size of the SET COVER instance, if we could efficiently optimally solve MTIO, then we could efficiently solve SET COVER. □

Theorem 7. *The greedy algorithm for* MMTIO *is optimal.*

Proof. As with MMLO, the choices we make for an out-edge only influences the information values at the out-edge's target, independently of any other out-edges selected. Thus, the optimal solution is to select the k out-edges that maximally improve total information, which is exactly greedy. □

Lemma 2. *Let $g(S)$ be the final total information of G (at t') if we add the temporal edges $(u, v, t, t+1, w)$ for each $(u, v) \in S$. $g(S)$ is monotone and submodular.*

Proof. First, g is clearly monotone: adding edges can only increase the total information at those edges' destinations, as well as all other nodes who eventually hear from those destinations.

Recall the definition of total information, using τ to avoid confusion with the departure time t of edges we're adding:

$$\text{totalInf}(u, v, \tau) = \min \left\{ 1, \lambda \,\text{totalInf}(u, v, \tau - 1) + \sum_{(z,v,d,a,w) \in E_\tau(v)} w\lambda^{\tau - d} \,\text{totalInf}(u, z, d) \right\}. \tag{1}$$

We'll modify this notation to define the total information v has about u at time τ if we add the edges in S to G:

$$\text{totalInf}_{uv\tau}(S) = \min \left\{ 1, \lambda \,\text{totalInf}_{uv(\tau-1)}(S) + \sum_{(z,v,d,a,w) \in E^S_\tau(v)} w\lambda^{\tau - d} \,\text{totalInf}_{uzd}(S) \right\}, \tag{2}$$

where $E^S_\tau(v)$ denotes the set of incoming edges to v arriving at τ, including those added in S (these only appear in the sum when $\tau = t + 1$). We want to show that $\sum_{u,v \in V} \text{totalInf}_{uv\tau}(S)$ is submodular. It suffices

to show that $\text{totalInf}_{uv\tau}(S)$ is submodular for all u, v, since linear combinations of submodular functions are submodular.

First, $\text{totalInf}_{uv\tau}(S)$ is submodular for all $\tau \leq t$, since edges added in S have no effect on total information values before $t+1$.

Now consider $\text{totalInf}_{uv\tau}(S)$ for $\tau = t+1$, the crucial timestep where added edges arrive. We need to show that

$$\text{totalInf}_{uv\tau}(S \cup \{(i,j),(h,\ell)\}) \leq \text{totalInf}_{uv\tau}(S \cup \{(i,j)\}) + \text{totalInf}_{uv\tau}(S \cup \{(h,\ell)\}). \tag{3}$$

If either $j \neq v$ or $\ell \neq v$, then this is vacuously true, since adding the edge $(i, j, t, t+1, w)$ cannot influence the total information at v at $t+1$ unless $j = v$. Therefore, consider the left hand side of (3) when $j = \ell = v$. The terms of the sum in Equation (2) corresponding to the new edges (i, v) and (h, v) are $w\lambda\,\text{totalInf}_{uit}(S \cup \{(i,v),(h,v)\})$ and $w\lambda\,\text{totalInf}_{uht}(S \cup \{(i,v),(h,v)\})$, respectively. Note that totalInf_{uit} and totalInf_{uht} are constant with respect to their arguments, since $t < t+1$, so define the constants $c_i = w\lambda\,\text{totalInf}_{uit}(\cdot)$ and $c_h = w\lambda\,\text{totalInf}_{uht}(\cdot)$. Additionally, the information from the previous timestep $\lambda\,\text{totalInf}_{uv(\tau-1)}(\cdot) = \lambda\,\text{totalInf}_{uvt}(\cdot)$ is also constant over its argument, so call it c_t. We thus have:

$$\text{totalInf}_{uv\tau}(S \cup \{(i,v),(h,v)\}) = \min\left\{1, c_t + c_i + c_h + \sum_{(z,v,d,a,w) \in E_\tau^S(v)} w\lambda^{\tau-d}\,\text{totalInf}_{uzd}(\cdot)\right\}$$

Similarly,

$$\text{totalInf}_{uv\tau}(S \cup \{(i,v)\}) = \min\left\{1, c_t + c_i + \sum_{(z,v,d,a,w) \in E_\tau^S(v)} w\lambda^{\tau-d}\,\text{totalInf}_{uzd}(\cdot)\right\}$$

and likewise for $\text{totalInf}_{uv\tau}(S \cup \{(h,\ell)\})$, but with c_h instead of c_i. We thus have the desired inequality (3), since $\min\{1, c_t + c_i + c_S\} + \min\{1, c_t + c_h + c_S\} \geq \min\{1, c_t + c_i + c_h + c_S\}$ (where c_S is the sum over $E_\tau^S(v)$), so $\text{totalInf}_{uv\tau}(S)$ is submodular for $\tau = t+1$.

All that remains to be shown is that $\text{totalInf}_{uv\tau}(S)$ is submodular for $\tau > t+1$, which we can do inductively given the base cases above. Suppose that $\text{totalInf}_{uv\tau'}(S)$ is submodular for all $\tau' < \tau$. Since $\tau > t+1$, the edges we encounter in the sum do not depend on S, so we have

$$\text{totalInf}_{uv\tau}(S) = \min\left\{1, \lambda\,\text{totalInf}_{uv(\tau-1)}(S) + \sum_{(z,v,d,a,w) \in E_\tau(v)} w\lambda^{\tau-d}\,\text{totalInf}_{uzd}(S)\right\}. \tag{4}$$

The second argument of the min is a linear combination of total informations before τ, which are submodular by inductive hypothesis. A linear combination of submodular functions is also submodular. Finally, the minimum of a submodular function and a constant is also submodular, so $\text{totalInf}_{uv\tau}(S)$ is submodular.

Summing over all pairs u, v, and taking $\tau = t'$ we then have that the final total information $g(S)$ is submodular.

□

Theorem 8. *The greedy algorithm $(1 - \frac{1}{e})$-approximates the optimal final total information in* MTIO, MTII, *and* MMTII.

Proof. Since $g(S)$ is monotone submodular by Lemma 2, the greedy algorithm that at each step picks the edge whose addition maximizes the final total information is a $(1 - \frac{1}{e})$-approximation to the optimal final total information [1]. There are $O(n)$ edges whose effect we need to compute to run the greedy algorithm, and we can compute their effects in polynomial time using the single-pass total information algorithm. □

Theorem 9. *We can compute all single-source maximum informations in time $O(m + nc\log n + nc\log c)$, where $n = |V|, m = |E|$, and $c = O(m)$ is the maximum degree of any node in G.*

Proof. We use Dijkstra's algorithm on a transformed graph (an approach inspired by [2]). The idea is to make min-weight paths in the transformed graph correspond to max-information paths in the original graph. To do so, we'll make nodes v_t in the transformed graph for each node v in the original graph and each timestamp t where v either sends or receives an edge. There are $O(nc)$ such nodes v_t, where $c = O(m)$ is the maximum number of temporal edges incident on any node in G. For each node v, let $t_1, \dots t_k$ be the k timestamps where v either sends or receives an edge. Form an edge from v_{t_i} to $v_{t_{i+1}}$ with weight $-\log(\lambda^{t_{i+1}-t_i})$ (notice that this is non-negative since $\lambda \leq 1$). This is a total of $O(nc)$ edges. Additionally, for each weighted temporal edge (u, v, d, a, w), form an edge from u_d to v_a with weight $-\log(\lambda^{a-d}w)$. This adds $O(m)$ edges to the transformed graph. If we want to compute information at time t, create a temporal node v_t for each original node v and form an edge from each node v's highest-timestamped temporal node before t (call it v_h) to v_t with weight $-\log(\lambda^{t-h})$. Finally, create a start node s and connect it with weight 0 edges to every temporal node of the source node. These start/end additions result in $O(n)$ extra nodes and $O(n + c)$ extra edges. The transformed graph thus has a total of $O(nc)$ nodes and $O(n + m)$ edges.

We can then run Dijkstra's algorithm to compute min-weight paths from s to every other node. The weights of the paths from s to each of the v_t nodes then correspond to the information v has about the source at time t. To see this, consider the weight of a path from s to w_t. Edges along this path either go from u_{t_i} to $u_{t_{i+1}}$ (information passing time at a node), from u_d to v_a (information traveling between nodes), or (once) from s to x_{t_i} with weight 0. The total weight of a path P in the transformed graph is then

$$\begin{aligned}
&\sum_{(u_{t_i}, u_{t_{i+1}}) \in P} -\log(\lambda^{t_{i+1}-t_i}) + \sum_{(u_d, v_a) \in P} -\log(\lambda^{a-d} w_{u_d, v_a}) \\
&= -\log\left(\prod_{(u_{t_i}, u_{t_{i+1}}) \in P} \lambda^{t_{i+1}-t_i} \prod_{(u_d, v_a) \in P} \lambda^{a-d} w_{u_d, v_a} \right),
\end{aligned}$$

where the first sum/product is over time-passing edges in P and the second sum/product is over the inter-node edges in P. Let x_{t_1} be the first node in the path after s. The total elapsed time of the path is then $t - t_1$. The total elapsed time of the path is the sum of elapsed times over each edge: $\sum_{(u_{t_i}, u_{t_{i+1}}) \in P}(t_{i+1} - t_i) + \sum_{(u_d, v_a) \in P}(a - d) = t - t_1$. We can combine the λ terms using this fact:

$$\begin{aligned}
&-\log\left(\prod_{(u_{t_i}, u_{t_{i+1}}) \in P} \lambda^{t_{i+1}-t_i} \prod_{(u_d, v_a) \in P} \lambda^{a-d} w_{u_d, v_a} \right) \\
&= -\log\left(\lambda^{\sum_{(u_{t_i}, u_{t_{i+1}}) \in P}(t_{i+1}-t_i) + \sum_{(u_d, v_a) \in P}(a-d)} \prod_{(u_d, v_a) \in P} w_{u_d, v_a} \right) \\
&= -\log\left(\lambda^{t-t_1} \prod_{(u_d, v_a) \in P} w_{u_d, v_a} \right) \qquad (*)
\end{aligned}$$

Notice that $(*)$ is exactly $-\log(\text{information transmitted across } P)$. Since the path can begin for free at any temporal version of w, the min-weight path from s to x_t therefore corresponds to the max-information path from w to x ending at time t. Taking $\exp(-\text{ path weight})$ converts back to information. Note that all weights in the transformed graph are non-negative, so we can use Dijkstra's algorithm to compute all single-source shortest paths. By looking at shortest paths to intermediate nodes u_{t_i}, we can actually compute the best information every node has about w at every timestep using a single call to Dijkstra's algorithm. The total running time on the transformed graph is $O(n + m + nc\log(nc)) = O(m + nc\log n + nc\log c)$. □

References

[1] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, 1978.

[2] Huanhuan Wu, James Cheng, Silu Huang, Yiping Ke, Yi Lu, and Yanyan Xu. Path problems in temporal graphs. *Proceedings of the VLDB Endowment*, 7(9):721–732, 2014.