Smooth Routing in Decaying Trees

Till Fluschnik
Humboldt-Universität zu Berlin,
Department of Computer Science,
Algorithm Engineering Group
Berlin, Germany
till.fluschnik@hu-berlin.de

Amela Pucic
Technische Universität Berlin,
Algorithmics and Computational
Complexity Group
Berlin, Germany
amela.pucic@student.tu-berlin.de

Malte Renken
Technische Universität Berlin,
Algorithmics and Computational
Complexity Group
Berlin, Germany
malte.renken@mailbox.de

ABSTRACT

Motivated by evacuation scenarios arising in extreme events such as flooding or forest fires, we study the problem of smoothly scheduling a set of paths in graphs where connections become impassable at some point in time. A schedule is smooth if no two paths meet on an edge and the number of paths simultaneously located at a vertex does not exceed its given capacity. We study the computational complexity of the problem when the underlying graph is a tree, in particular a star or a path. We prove that already in these settings, the problem is NP-hard even with further restrictions on the capacities or on the time when all connections ceased. We provide an integer linear program (ILP) to compute the latest possible time to evacuate. Using the ILP and its relaxation, we solve sets of artificial (where each underlying graph forms either a path or star) and semi-artificial instances (where the graphs are obtained from German cities along rivers), study the runtimes, and compare the results of the ILP with those of its relaxation.

KEYWORDS

Decaying graphs; Temporal paths; Scheduling; NP-hardness; ILP

ACM Reference Format:
Till Fluschnik, Amela Pucic, and Malte Renken. 2026. Smooth Routing in Decaying Trees. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 9 pages. https://doi.org/10.65109/BBUZ3605

1 INTRODUCTION

In evacuation scenarios facing extreme events like flooding or forest fires, the underlying routing network may suffer under ceasing connections. Evacuations plans may include *fixed* evacuation routes, i.e., each route is fully described as the sequence of all its vertices. However, since the way how extreme events impact the network can differ, a fixed evacuation schedule, i.e., an assignment of departure times for each evacuation route on each of its connections, could fail. Given a set of *fixed* evacuation routes in a network with ceasing connections and vertex capacities, we wonder what is the computational complexity of finding a *smooth* feasible evacuation schedule. Herein, we call an evacuation schedule smooth when, on the one hand, at each vertex the number of routes that simultaneously gather is at most the capacity of that vertex, and, on the other hand, every two routes start to traverse a connection at different times and traverse no two-way connection in opposite directions such that they meet at some time on it. We are particularly interested in the special case when the underlying network forms a tree. Herein, as subclasses of trees, we focus on underlying networks that form paths (like a main street) or stars (like larger crossings or simplified roundabouts).

Our Model and Central Decision Problem. Our model uses *decaying* graphs. A decaying graph $\mathcal{G} = (V, E, A, c, \theta, d, \tau)$ consist of a static (mixed) graph $G = (V, E, A)$ with edge set $E \subseteq \binom{V}{2}$ and arc set $A \subseteq V \times V$, vertex capacities $c\colon V \to \mathbb{N}$, and lifetime $\tau \in \mathbb{N}$, and each connection (i.e., edge or arc) is equipped with a traversal time $\theta\colon E \cup A \to \{0, \dots, \tau - 1\}$ and a deadline $d\colon E \cup A \to \{1, \dots, \tau\} =: T$. Undirected edges model segments which can be traversed in both directions but not simultaneously (e.g., narrow streets). For every static (mixed) graph $G = (V, E, A)$ we assume that every two adjacent vertices are connected either by an edge, an arc, or two antiparallel arcs. Intuitively, $\mathcal{G}$ models the street network with ceasing connections. The static (mixed) graph together with θ represents the street network in the usual way (vertices correspond to junctions or dead ends, connections to street segments, and θ specifies traversal times), but each connection is associated with a deadline d until which it is available for any evacuation route. Moreover, each vertex has a capacity c which limits the number of evacuation routes present at the vertex at the same time.

Each fixed evacuation route is modelled as a path in G and is then realized over time in $\mathcal{G}$. A path P can be represented as a sequence $P = (v_1, v_2, \dots, v_{k+1})$ of at least two vertices all of which are mutually distinct or as a sequence $P = (e_1, \dots, e_k)$ of connections, where $e_i = \{v_i, v_{i+1}\}$ or $e_i = (v_i, v_{i+1})$. Let $V(P)$ denote the set of vertices of P and $X(P) = \{(v_i, v_{i+1}) \mid i \in \{1, \dots, k\}\}$ be the set of all pairs of consecutive vertices in P. Let $s(P) = v_1$ denote the source and $t(P) = v_{k+1}$ the sink of P. For a vertex $v \in V$ and set $\mathcal{P}$ of paths over V, let $\mathcal{P}(v)$ denote the set of all paths in $\mathcal{P}$ containing v.

Altogether, an instance $I = (\mathcal{G}, \mathcal{P})$ of our problem consists of a decaying graph $\mathcal{G} = (V, E, A, c, \theta, d, \tau)$ and a set $\mathcal{P} = \{P_1, \dots, P_p\}$ of paths in $G = (V, E, A)$ (see Figure 1(left) for an example).

Next we model the temporal realization, or schedule, of the evacuation routes, specifying where the evacuation unit is located along its route at each time. By this, each path in $\mathcal{P}$ becomes a *temporal* path. A *temporalization* π of an instance I is an assignment $\pi(P, e_i) \in T$ for each path $P = (e_1, \dots, e_k) \in \mathcal{P}$ such that $\pi(P, e_i) + \theta(e_i) \leq \pi(P, e_{i+1})$ and $\pi(P, e_i) + \theta(e_i) \leq d(e_i)$ (we refer to this as *adequacy*); $\pi(P, e_i)$ corresponds to the departure time of P on edge e_i.[1] Given a temporalization, two paths P, P' are *temporally*

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus. © 2026 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). https://doi.org/10.65109/BBUZ3605

[1] Such temporalizations can be equivalently defined as assignments of departure times for paths from their vertices; We will make use of this equivalence throughout.

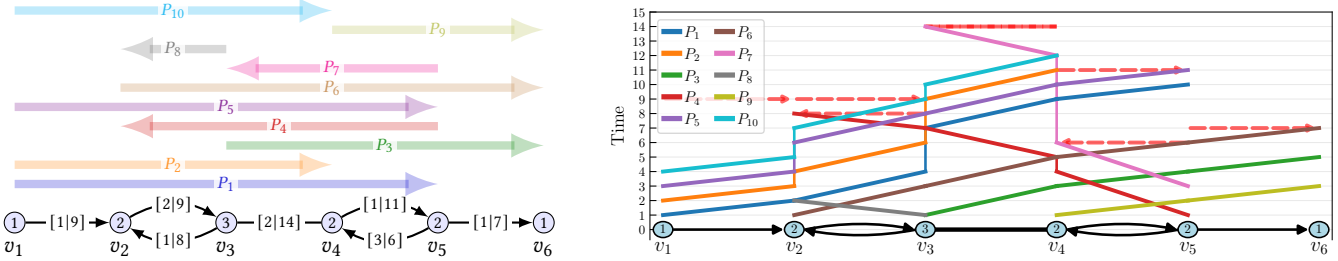

Figure 1: Example instance of SRDG on a decaying path with 6 vertices (left) and a solution witnessing feasibility (right). The capacity of each vertex v_i is encircled. (Left) On each connection e we indicate its traversal time and deadline as $[\theta(e)|d(e)]$. Each of the paths $P_1, \ldots, P_{10}$ is described by the vertically aligned source and sink and by an arc for its direction. (Right) For each path its timely location on any vertex or connection is drawn. E.g., the path P_1 (blue) starts at time step 1 from its source v_1, arrives at and departs from v_2 at time step 2, and so on. Red dashed lines/arrows indicate the deadline on the respective connection.

edge-disjoint when the following hold: (i) if $(v, w) \in X(P) \cap X(P')$, then $\pi(P, \{v, w\}) \neq \pi(P', \{v, w\})$ (resp., $\pi(P, (v, w)) \neq \pi(P', (v, w))$), and (ii) if $e = \{v, w\} \in E$, $(v, w) \in X(P)$, and $(w, v) \in X(P')$, then $|\pi(P, e) - \pi(P', e)| \geq \max\{1, \theta(e)\}$ (in the special case of zero traversal time, it forbids departing on that edge simultaneously). Intuitively, when two paths are temporally edge-disjoint, (i) they depart at different times on common connections in the same direction, and (ii) they do not meet at any point in time on common edges when traversing them in opposite directions. A temporalization is *temporally edge-disjoint* if every two paths $P, P' \in \mathcal{P}$ are temporally edge-disjoint. A path $P = (v_1, \ldots, v_{k+1}) \equiv (e_1, \ldots, e_k)$ is (located) at an inner vertex v_i, $1 < i \leq k$, at the time steps $\pi(P, e_{i-1}) + \theta(e_{i-1}), \ldots, \pi(P, e_i)$. As special cases, a path is (located) only at time $\pi(P, e_1)$ at $s(P) = v_1$ and only at time $\pi(P, e_k) + \theta(e_k)$ at $t(P) = v_{k+1}$. A temporalization *respects* the (vertex) capacities if for every vertex $v \in V$ and every time step $t \in T$, at most $c(v)$ many paths are located on v at t. A decaying graph is *uncapacitated* when $c(v) \geq |\mathcal{P}(v)|$ for all $v \in V$, i.e., when all paths containing v can be located at v at the same time, and *capacitated* otherwise. Altogether, a temporalization is *valid* if it is temporally edge-disjoint and respects the vertex capacities. Finally, we seek to solve the following problem (see Figure 1(right) for a witnessing solution):

Problem (Smooth Routing in Decaying Graphs (SRDG)). **Given** an instance $I = (\mathcal{G}, \mathcal{P})$ with decaying graph $\mathcal{G} = (V, E, A, c, \theta, d, \tau)$ and set $\mathcal{P} = \{P_1, \ldots, P_p\}$ of paths in $G = (V, E, A)$, the **question** is whether there is a valid temporalization π of I.

Our Contributions. We contribute both on the theoretical and experimental side. For all settings of the deadlines and capacities being constant or scaling (where scaling means that the value may grow with the instance size), we prove whether SRDG on capacitated decaying paths and stars is NP-hard or polynomial-time solvable (see Table 1 for an overview). Our main result is that SRDG is NP-hard on uncapacitated decaying trees with constant deadlines. We provide an integer linear program (ILP) to compute the minimum addition to every deadline to achieve a feasible instance, i.e., the latest possible time to evacuate. With experiments on artificial (decaying stars and paths) and semi-artificial instances (street networks from German cities along rivers for which we simulated flooding), we prove the practicality of the ILP and its relaxation.

Table 1: Overview on our complexity-theoretic results. Each hardness result still holds when the respective decaying graph is exogenous. Results shown in bold are discussed in greater detail in this short version of the paper.

	deadlines	capacitated	
		constant capacities	scaling capacities
path	constant	**P** (Thm. 2)	
	scaling	NP-hard (Thm. 1)	
star	constant	P (Thm. 3)	**NP-hard** (Thm. 4)
	scaling	NP-hard (Thm. 5)	NP-hard
tree	constant	**NP-hard** (Thm. 6)	
	scaling		

Due to the space restrictions, proof details of results marked with ★ are deferred to a full version of the paper.[2]

Related Work. Issues caused by human behavior through evacuation like congestion [18] are problematic. A common framework is evacuation via origin-destination (OD) pairs [14, 23, 24, 26], suitable for centralized planning e.g. when evacuating inhabitants with no access to private automobiles. Computing and testing evacuation paths are mostly done for simultaneously initiated evacuation via, e.g., time-expanded networks and flow models [1, 8, 19, 22, 27].

Closest to us is the work by Klobas et al. [16] and Kunz et al. [17], studying the problem Temporally Disjoint Paths (TDP). Their model yet differs in several aspects: They study temporal graphs where all connections are edges that can also reappear, they have no vertex capacities and travel times (translated into our model, all vertex capacities are one and all travel times are zero), and their paths are given only by OD pairs. Due to the latter, TDP is NP-hard already if $\tau = 1$ [16]; SRDG is (trivially) polynomial-time solvable in this case. TDP is hard [17] for different graph restrictions and parameters, e.g., NP-hard (and W[1]-hard w.r.t. the number of vertices) on temporal stars (without constantly-bounded lifetime).

[2]Code and data for the experiments, as well as the full version of this paper, are available at: https://github.com/buhtig-tf/Smooth-Routing-in-Decaying-Trees.

Both [16, 17] study temporal trees with temporal lines [16, 17] and stars [17] (note that OD pairs fully describe paths in decaying trees).

Waiting-time restrictions for temporal paths are studied [5], even with no waiting time for a set of paths all with the same OD pair [9]. Also studied is the effect of ceasing connections on shortest temporal paths [4]. Finally, there is work related to successive (as opposed to simultaneous) evacuation in temporal graphs [6, 10].

Multi-agent pathfinding [3, 11, 25] coordinates multiple agents sharing limited network resources while e.g. avoiding safety-critical conflicts such as collisions [2, 21] or overloading [24]. It appears in dynamic settings [20], where edge availability or travel times vary, and formulated on graphs [28, 29], which encode feasible routes.

2 PRELIMINARIES

We denote by $\mathbb{N}$ and $\mathbb{N}_0$ the natural numbers excluding and including zero, respectively. For a graph $G = (V, E, A)$, we denote by $\mathcal{U}(G)$ the graph obtained from replacing each arc with an edge and then deleting duplicated edges. A *decaying tree* $\mathcal{G} = (V, E, A, c, \theta, d, \tau)$ is a decaying graph with $\mathcal{U}(G)$ being a *tree* (a connected, cycle-free graph), where $G = (V, E, A)$. If $\mathcal{U}(G)$ is a path (star), then we call $\mathcal{G}$ a decaying path (star). A *star* with n vertices is a graph with center vertex v^* and vertices $v_1, \dots, v_{n-1}$, each of which is adjacent only with v^*. In a decaying tree with vertices s, z, we also write (s, z)-path for the unique path with source s and sink z. A decaying tree is *exogenous*, when for every $t \in \{1, \dots, \tau\}$, $\mathcal{U}(G_t)$ is a tree plus isolated vertices, where $G_t = (V, E_t, A_t)$ such that $e \in E_t$ (resp. $a \in A_t$) if and only if $e \in E$ and $d(e) \geq t$ (resp. $a \in A$ and $d(a) \geq t$).

3 COMPUTATIONAL COMPLEXITY

We distinguish between decaying paths (Section 3.1), decaying stars (Section 3.2), and decaying trees (Section 3.3).

3.1 On Capacitated Paths

In Section 3.1.1 we show that SRDG on decaying paths is NP-hard even for unit capacities. In Section 3.1.2, we show that if the lifetime is constant, then SRDG is polynomial-time solvable.

3.1.1 *On Paths with Capacity One.*

Theorem 1 (★). SRDG *is* NP*-hard even on exogenous decaying paths where every vertex has capacity one.*

Our reduction builds on the reduction from Klobas et al. [16, Theorem 6]. On a high level, one can represent colored intervals as paths going from "left to right" on the decaying path such that they all cross a common vertex v^* and can only start and finish at time steps according to the respective interval's "left and right" endpoints. Then, with unit capacities, there is a valid temporalization (that in particular respects the capacity of 1 of v^*) if and only if there is a set of pairwise disjoint intervals of each color, where the latter corresponds to an NP-hard task.

3.1.2 *On Paths with Constant Deadlines.*

Next we show that presumably, Theorem 1 does not hold for constant deadlines, even if we allow scaling capacities. In fact, we show an even stronger result that if the maximum *vertex load* $\mathrm{vl}(I) := \max_{v \in V} |\mathcal{P}(v)|$, i.e., the maximum number of paths containing the same vertex, is constant, then SRDG is solvable in polynomial-time.

Theorem 2 (★). SRDG *on decaying paths is solvable in* $O(n \cdot |\mathcal{P}| + \tau^{4\,\mathrm{vl}(I)} \cdot \mathrm{vl}(I)^2 \cdot n)$ *time, where* I *denotes any input instance with* n *vertices, lifetime* τ*, and set* $\mathcal{P}$ *of paths.*

For an instance I on a decaying path with lifetime τ, we can assume that $\mathrm{vl}(I) \leq 4\tau$, as otherwise I is a no-instance: at most four paths can arrive and leave at a time step in a vertex. Thus, SRDG on decaying paths is polynomial-time solvable for constant deadlines.

We prove Theorem 2 via dynamic programming over the vertices of the decaying path from "left to right". On a high level, the dynamic program at vertex v_{i+1} considers all possible ways paths that traverse to or from "the left" are scheduled to leave or arrive from or at v_i and v_{i+1}. Thereby, it takes care only about the capacities of v_i and v_{i+1} as well as about monotonicity and temporally edge-disjointness regarding the connection(s) between v_i and v_{i+1}.

Let $\mathcal{G} = (V, E, A, c, \theta, d, \tau)$ be a decaying path with vertex set $V = \{v_1, \dots, v_n\}$ and connections exactly between v_i and v_{i+1} for all $i \in \{1, \dots, n-1\}$. Let c denote the capacities and $\mathcal{P}$ the set of paths. For every $i \in \{1, \dots, n\}$, let $\mathcal{G}_i = (V_i, E_i, A_i, c_i, \theta_i, d_i, \tau)$ denote $\mathcal{G}$ restricted to $V_i := \{v_1, \dots, v_i\}$, where E_i and A_i contains every connection from E and A with both endpoints in V_i, $c_i \colon V_i \to \mathbb{N}$ with $c_i(v) = c(v)$, $d_i \colon E_i \cup A_i \to \{1, \dots, \tau\}$ with $d_i(e) = d(e)$, $\theta_i \colon E_i \cup A_i \to \{0, \dots, \tau - 1\}$ with $\theta_i(e) = \theta(e)$. Let $\mathcal{P}_i$ denote the set of all paths $P \in \mathcal{P}$ with $|V(P) \cap V_i| \geq 2$ restricted to V_i; that is, a path $P = (v_j, \dots, v_k)$ with $j < k$ and $j < i$ restricted to V_i is the path $P_i = (v_j, \dots, v_{\min\{k,i\}})$, and a path $P = (v_j, \dots, v_k)$ with $j > k$ and $k < i$ restricted to V_i is the path $P_i = (v_{\min\{i,j\}}, \dots, v_k)$. Let $R_i \subseteq \mathcal{P}_i$ be the set of paths that arrive at v_i and let $L_i \subseteq \mathcal{P}_i$ be the set of paths that depart from v_i.

Construction 1. Let $I = (\mathcal{G}, \mathcal{P})$ be an instance of SRDG consisting of a decaying path $\mathcal{G} = (V, E, A, c, \theta, d, \tau)$ with $V = \{v_1, \dots, v_n\}$ and a set $\mathcal{P} = \{P_1, \dots, P_p\}$ of paths in $G = (V, E, A)$. For all $i \in \{1, \dots, n\}$, for all assignments $\rho_i \colon R_i \to \{1, \dots, \tau\}$ of the arrival times of paths in R_i at v_i, and for all assignments $\lambda_i \colon L_i \to \{1, \dots, \tau\}$ of the departure times of paths in L_i from v_i, define $T[i, \rho_i, \lambda_i] \in \{\top, \bot\}$ as follows. Set $T[1, \rho_1, \lambda_1] = \top$ (since $R_1 = L_1 = \emptyset$, as convention, there is only one ρ_1 and λ_1 mapping R_1 and L_1 to 1 and τ, respectively). Iteratively for $i = 1, \dots, n-1$, we set $T[i+1, \rho_{i+1}, \lambda_{i+1}] = \top$ if and only if there exist ρ_i and λ_i such that $T[i, \rho_i, \lambda_i] = \top$ and each of (1)–(9) holds true, which are defined as follows. Herein, with $\overrightarrow{e_i}$ we refer to (v_i, v_{i+1}) if $(v_i, v_{i+1}) \in A$ and to $\{v_i, v_{i+1}\}$ otherwise; similarly, with $\overleftarrow{e_i}$ we refer to (v_{i+1}, v_i) if $(v_{i+1}, v_i) \in A$ and to $\{v_i, v_{i+1}\}$ otherwise.

First, we address monotonicity and deadlines.

$$\forall P \in R_i \cap R_{i+1} : \quad \rho_i(P) \leq \rho_{i+1}(P) - \theta(\overrightarrow{e_i}) \tag{1}$$

$$\forall P \in L_i \cap L_{i+1} : \quad \lambda_i(P) \geq \lambda_{i+1}(P) + \theta(\overleftarrow{e_i}) \tag{2}$$

$$\forall P \in R_{i+1} : \quad d(\overrightarrow{e_i}) \geq \rho_{i+1}(P) \tag{3}$$

$$\forall P \in L_{i+1} : \quad d(\overleftarrow{e_i}) \geq \lambda_{i+1}(P) + \theta(\overleftarrow{e_i}) \tag{4}$$

Next, we address temporal disjointness.

$$\forall P, Q \in R_{i+1},\ P \neq Q : \quad \rho_{i+1}(P) \neq \rho_{i+1}(Q) \tag{5}$$

$$\forall P, Q \in L_{i+1},\ P \neq Q : \quad \lambda_{i+1}(P) \neq \lambda_{i+1}(Q) \tag{6}$$

$$e_i = \{v_i, v_{i+1}\} \in E \implies \big(\forall P \in R_{i+1}, Q \in L_{i+1} : \max\{1, \theta(e_i)\} \leq |\rho_{i+1}(P) - \theta(e_i) - \lambda_{i+1}(Q)|\big) \tag{7}$$

Finally, we address the capacities. First for v_{i+1}, second for v_i:

$$c(v_{i+1}) = 1 \implies \forall P \in R_{i+1}, Q \in L_{i+1} : \rho_{i+1}(P) \neq \lambda_{i+1}(Q) \quad (8)$$

$$\forall t \in \{1, \dots, \tau\} : c(v_i) \geq |\{P \in \mathcal{P}(v_i) \mid t \in S_i(P)\}|, \quad (9)$$

where for all $P \in \mathcal{P}(v_i)$ we have

$$S_i(P) = \begin{cases} [\rho_i(P), \rho_{i+1}(P) - \theta(\overrightarrow{e_i})], & \text{if } P \in R_i \cap R_{i+1}, \\ [\lambda_{i+1}(P) + \theta(\overleftarrow{e_i}), \lambda_i(P)], & \text{if } P \in L_i \cap L_{i+1}, \\ [\rho_i(P), \rho_i(P)], & \text{if } P \in R_i \setminus R_{i+1}, \\ [\rho_{i+1}(P) - \theta(\overrightarrow{e_i}), \rho_{i+1}(P) - \theta(\overrightarrow{e_i})], & \text{if } P \in R_{i+1} \setminus R_i, \\ [\lambda_i(P), \lambda_i(P)], & \text{if } P \in L_i \setminus L_{i+1}, \\ [\lambda_{i+1}(P) + \theta(\overleftarrow{e_i}), \lambda_{i+1}(P) + \theta(\overleftarrow{e_i})], & \text{if } P \in L_{i+1} \setminus L_i. \end{cases} \quad \diamond$$

Intuitively, the dynamic program is correct since at vertex v_{i+1}, to extend paths coming from "left" to v_{i+1} or leaving to the "left" from v_{i+1}, all that matters is whether this extension respects the deadline of the connection(s) between v_i and v_{i+1} (see (3) and (4)), is temporally edge-disjoint between v_i and v_{i+1} (see (5)–(7)), respects the capacities of v_i and v_{i+1} (see (8) and (9)), and fits to a valid temporalization of the set $\mathcal{P}_i$ of paths in $\mathcal{G}_i$ (in the sense of monotonicity, see (1) and (2)). Whether such an extension fits depends mainly on when the paths to extend arrive or leave v_i, which is stored in $T[i, \rho_i, \lambda_i]$ for all possible entries ρ_i and λ_i.

For each of the at most $\tau^{2|\mathcal{P}(v_{i+1})|}$ entries $T[i+1, \cdot, \cdot]$, we have to check at most $\tau^{2|\mathcal{P}(v_i)|}$ entries $T[i, \cdot, \cdot]$ with respect to (1)–(9). Assuming all of the path subsets like R_i and L_i are precomputed in $O(n \cdot |\mathcal{P}|)$ time, each of such a check takes time $O(|\mathcal{P}(v_{i+1})|^2)$ (for (1)–(8)) and $O(|\mathcal{P}(v_{i-1})| \cdot \log(|\mathcal{P}(v_{i-1})|)$ (for (9)).

3.2 On Capacitated Stars

For every instance I on a decaying star with lifetime τ and set $\mathcal{P}$ of paths, since every path contains center v^*, we have $|\mathcal{P}| = |\mathcal{P}(v^*)| = \mathrm{vl}(I)$ (vl is defined in Section 3.1.2). By guessing all paths' arrival and departure times at v^*, we get the following.

Theorem 3 (★). SRDG *on decaying stars is solvable in* $O(n \cdot \mathrm{vl}(I) + \tau^{2 \cdot \mathrm{vl}(I)} \cdot \mathrm{vl}(I)^2)$ *time, where* I *denotes any input instance with* n *vertices and lifetime* τ.

In contrast to decaying paths, we cannot upper bound vl by the lifetime (see Theorem 4). Yet, with maximum capacity c^* and deadline d^*, there can be at most $c^* \cdot d^*$ many paths (in each of the d^* time steps, at most c^* paths can be located at v^*). Since $|\mathcal{P}| = \mathrm{vl}(I)$ on decaying stars, it follows that SRDG is linear-time solvable on decaying stars with constant capacities and deadlines.

3.2.1 *Scaling Capacities and Constant deadlines.*

Theorem 4. SRDG *is* NP*-hard even on capacitated decaying stars with constant lifetime.*

We give a reduction from the NP-hard [12] problem Cubic Independent Set, where, given an undirected graph G, where each vertex has exactly three neighbors, and an integer k, the question is whether there are at least k vertices that are pairwise not adjacent.

Construction 2. Let $I = (G = (V, E), k)$ be an instance of Cubic Independent Set with m edges and n vertices (recall that $2m = 3n$). We construct an instance I' of SRDG on star $G' = (V', E')$ with center vertex v^* as follows (see Figure 2 for an illustration).

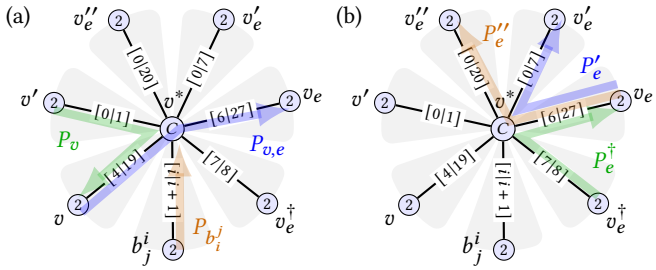

Figure 2: Illustration to Construction 2. For each edge e we indicate the traversal time and deadline as $[\theta(e)|d(e)]$. Here, we picked v and e with $v \in e$ for illustration.

Table 2: The values for n_i in Construction 2. Since each n_i is of the form $n_i = C - n'_i$, we give the n'_i values that correspond to the number of paths that can be located on v^* at time step i next to the paths starting in all vertices from B_i.

i	$C - n_i$	i	$C - n_i$	i	$C - n_i$
1	n	7	$4k + m$	18	$2(n-k)$
2, 3, 4	k	8	m	19	$3(n-k)$
5	$2k$	$9, \dots, 16$	0	20	$3(n-k)+m$
6	$3k$	17	$n-k$	$21, \dots, 27$	$3(n-k)$

Let $\tau = 27$. For each vertex $v \in V$, add a vertex v and v' to V'. Make v adjacent with v^* with traversal time 4 and deadline 19. Make v' adjacent with v^* with traversal time 0 and deadline 1. For every edge $e \in E$, add the vertex v_e to V' and make it adjacent with v^* with travel time 6 and deadline 27, add the vertex v'_e to V' and make it adjacent with v^* with travel time 0 and deadline 7, add the vertex v''_e to V' and make it adjacent with v^* with travel time 0 and deadline 20, add the vertex $v^\dagger_e$ to V' and make it adjacent with v^* with travel time 7 and deadline 8, For each $i \in \{1, \dots, \tau\}$, add the set $B_i := \{b^i_1, \dots, b^i_{n_i}\}$ of *blocker* vertices to V', where n_i is defined in Table 2, and make each vertex from B_i adjacent with v^* with travel time i and deadline $i+1$ (thus, every path with source from B_i must start at time step 1 and arrive at time step $i+1$ at vertex v^*). Set $B := \bigcup_{i=1}^{\tau} B_i$. Set $c(v^*) = C$ with $C := 2m+3n+4k$ and for every node $x \in V' \setminus \{v^*\}$, set $c(x) = 2$ (thus, every temporal edge-disjoint temporalization respects each leaf's capacity constraint, since at most one path departs from or arrives at a leaf at each time step).

For the path set, for each $v \in V$, add the *vertex* path $P_v = (v', v^*, v)$. For each $e \in E$, add paths $P'_e = (v_e, v^*, v'_e)$, $P''_e = (v_e, v^*, v''_e)$, and $P^\dagger_e = (v^\dagger_e, v^*, v_e)$, and for each $v \in e$, add the *verifier* path $P_{v,e} = (v, v^*, v_e)$. For each $b \in B$, add the *blocker* path $P_b = (b, v^*)$. $\diamond$

We get the following directly from the construction.

Observation 1. *Let* I' *be a* yes*-instance. Then, for every solution, the following hold: (i) Path* P_v *for every* $v \in V$ *arrives at time step 1 at* v^* *and at least* $n-k$ *of the vertex paths leave* v^* *also on time step 1. (ii) Path* $P^\dagger_e$ *for every* $e \in E$ *starts at time step* 1 *and arrives and leaves* v^* *at time step* 8. *(iii) Path* P'_e *for every* $e \in E$ *starts at time step* 1 *and arrives and leaves* v^* *at time step* 7. *(iv) Every path that arrives on* v^* *latest at time step* 7 *departs from* v^* *latest on time step* 7.

Assume I' to be a yes-instance and consider an arbitrary solution. For every $e \in E$, path P''_e must start latest at time step 14 by construction. Indeed, due to Observation 1(ii) & (iii) and the fact that only m paths can be located at v^* at time step 8, path P''_e must start exactly at time step 14.

Lemma 1 (★). *Let I' be a* yes*-instance. Then, for every solution and for every $e \in E$, path P''_e starts at time step* 14 *and arrives and leaves v^* at time step* 20.

By Observation 1(i), verifier paths for at most k different vertices from V can arrive at v^* latest at time step 7. Now, basically due to Lemma 1, it follows that these are exactly $3k$ verifier paths: If not, then more than $3(n-k)$ verifier path must be located at v^* at time step 19, which would violate the capacity constraint of v^*.

Lemma 2 (★). *Let I' be a* yes*-instance. Then, for every solution, from exactly k different vertices all $3k$ verifier paths arrive at v^* latest at time step* 7.

Proof of Theorem 4. Let I' be the instance obtained from instance $I = (G, k)$ with $G = (V, E)$ of Cubic Independent Set using Construction 2, which can be done in time polynomial in $|I|$. We prove that I is a yes-instance if and only if I' is a yes-instance.

($\Rightarrow$) Let $W \subseteq V$ be an independent set of size k. We route all paths addressed in Observation 1(iii)–(ii) and Lemma 1 as described therein. Path P_v for every $v \in V \setminus W$ arrives and leaves v^* at time step 1, and for every $v \in W$ arrives at v^* at time step 1 and leaves v^* at time step 7. Also, for every $v \in W$, the verifier paths for v start (in arbitrary order) at time steps 1, 2, and 3, and stay at v^* until time step 7. Since W is an independent set, for every two paths $P_{v,e}$ and $P_{w,e'}$ with $v, w \in W$, $v \in e$, and $w \in e'$, we have that $e \neq e'$. Thus, all verifier paths for vertices in W leave at time step 7 (to a different sink). The remaining $3(n-k)$ verifier paths start at time steps 13–15, arrive at v^* at 17–19, and from time step 20 onwards, successively and earliest possible leave to their sinks.

($\Leftarrow$) Let π be a valid temporalization to I'. Due to Lemma 2, exactly $3k$ verifier paths from a set $W \subseteq V$ of exactly k vertices arrive latest at time step 7 at v^*. Due to Observation 1(iv), they all leave v^* at time step 7. We claim that W is an independent set. Suppose towards a contradiction that there are $u, v \in W$ such that $e = \{u, v\} \in E$. Then the verifier paths $P_{v,e}$ and $P_{u,e}$ depart v^* towards v_e at the same time step 7, a contradiction. □

3.2.2 Constant Capacities and Scaling Deadlines.

Theorem 5 (★). SRDG *is* NP*-hard even on decaying stars with maximum capacity one.*

For Theorem 5 we reduce from the NP-hard [15] problem Vertex Cover, where, given an undirected graph $G = (V, E)$ and an integer k, the question is whether there is a set $W \subseteq V$ with $|W| \leq k$ such that every edge $e \in E$ has an endpoint in W, i.e., $e \cap W \neq \emptyset$. Similar to the proof of Theorem 4, from the input instance of Vertex Cover, for each $v \in V$ and $e \in E$ there are leaves v and v_e, respectively, in the constructed instance. Yet, in contrast, now there are large traversal times: for every edge in E there is a unique traversal time. Then, on a high-level, for every edge $e \in E$ there are exactly two time windows for the *verifier* path that starts at v_e and terminates at v with $v \in e$: one window of size $|E|$ that they all share, and another unique window of size one. By this, for each edge one of its verifier paths must arrive in the first window (this corresponds to the covering). Additionally constructed leaves and paths make only k vertices available as endpoints of these verifier paths in the first window, witnessing a vertex cover of size k.

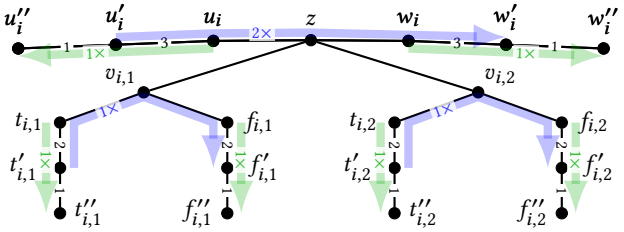

Figure 3: Illustration to the construction regarding V_i and its blocker paths. Edges with deadlines < 4 are labeled.

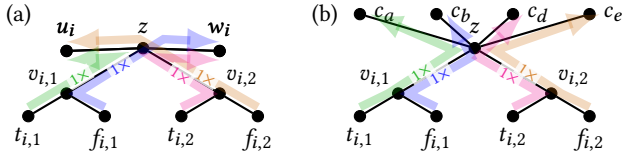

Figure 4: Illustration to the construction for (a) validation paths and (b) verifier paths.

3.3 On Uncapacitated Trees

Theorem 6 (★). SRDG *is* NP*-hard even on uncapacitated, exogenous decaying trees with lifetime four.*

We give a reduction from the NP-hard [7] problem (2,2)-3SAT: Given a set X of variables and a boolean CNF-formula $\phi = \bigwedge_{j=1}^{M} C_j$ over X with clauses $C_1, \ldots, C_M$, each consisting of exactly three distinct variables, such that every variable appears exactly twice negated and exactly twice unnegated in ϕ, the question is whether there is a truth assignment $\beta\colon X \to \{\top, \bot\}$ that satisfies ϕ?

Construction 3. Let $I = (X, \phi)$ be an instance of (2,2)-3SAT with $X = \{x_1, \ldots, x_N\}$ and $\phi = \bigwedge_{j=1}^{M} C_j$. Construct an instance $I' = (\mathcal{G}, \mathcal{P})$ with $\mathcal{G} = (V, E, \emptyset, c, \theta, d, \tau)$ of SRDG with $\tau = 4$ and uncapacitated c as follows (see Figures 3 and 4 for an illustration).

Let $V = \{z\} \cup \bigcup_{i=1}^{N} V_i \cup \bigcup_{j=1}^{M} \{c_j\}$, where $V_i = \{u_i, u'_i, u''_i\} \cup \{w_i, w'_i, w''_i\} \cup \bigcup_{j \in \{1,2\}} \{\{v_{i,j}\} \cup \{t_{i,j}, t'_{i,j}, t''_{i,j}\} \cup \{f_{i,j}, f'_{i,j}, f''_{i,j}\}\}$ for each $i \in \{1, \ldots, N\}$. Construct edge set E with deadlines d and $\theta(e) = 0$ for every edge $e \in E$ as follows: For each $i \in \{1, \ldots, N\}$ (see Figure 3), add edges $\{z, u_i\}$, $\{z, w_i\}$ with deadline 4, $\{u_i, u'_i\}$, $\{w_i, w'_i\}$ with deadline 3, and $\{u'_i, u''_i\}$, $\{w'_i, w''_i\}$ with deadline 1; For each $j \in \{1, 2\}$, add $\{z, v_{i,j}\}$ with deadline 4, $\{t_{i,j}, t'_{i,j}\}$, $\{f_{i,j}, f'_{i,j}\}$ with deadline 2, and $\{t'_{i,j}, t''_{i,j}\}$, $\{f'_{i,j}, f''_{i,j}\}$ with deadline 1. For each $j \in \{1, \ldots, M\}$ (see Figure 4(b)), add edge $\{z, c_j\}$ with deadline 4.

Since G is a decaying tree, we represent each path as $P(s, t)$ with source s and sink t. For each $i \in \{1, \ldots, N\}$, add the following *blocker* paths (see Figure 3). Add $P(u_i, u''_i)$, $P(w_i, w''_i)$, $P_1(u'_i, w'_i)$ and $P_2(u'_i, w'_i)$. For all $j \in \{1, 2\}$, add $P(t_{i,j}, t''_{i,j})$, $P(f_{i,j}, f''_{i,j})$, and $P(t'_{i,j}, f'_{i,j})$. Next (see Figure 4(a)), add the *validity* paths $P(t_{i,1}, u_i)$,

$P(t_{i,2}, w_i)$, $P(f_{i,1}, w_i)$, and $P(f_{i,2}, u_i)$. Finally add the following *verifier* paths (see Figure 4(b)). When variable x_i appears unnegated in clauses C_a and C_d with $a < d$, add the *verifier* paths $P(t_{i,1}, c_a)$ and $P(t_{i,2}, c_d)$. When variable x_i appears negated in clauses C_b and C_e with $b < e$, add the verifier paths $P(f_{i,1}, c_b)$ and $P(f_{i,2}, c_e)$. ⋄

By construction, we immediately get the following.

Observation 2. *Let I' be a* yes*-instance. For every solution and every $i \in \{1, \dots, N\}$ we have: (i) on their edges, $P(u_i, u_i'')$, $P(w_i, w_i'')$, take exactly the time step* 1*, and $P_1(u_i', w_i')$ and $P_2(u_i', w_i')$ take exactly the time steps* 2 *and* 3*. (ii) for every $j \in \{1, 2\}$, on their edges, $P(t_{i,j}, t_{i,j}'')$, and $P(f_{i,j}, f_{i,j}'')$, take exactly the time step* 1*, and $P(t_{i,j}', f_{i,j}')$ take exactly the time step* 2*.*

Assume I' is a yes-instance and consider an arbitrary solution. For each $i \in \{1, \dots, N\}$, by construction, only one of the two validity paths starting at $t_{i,j}$ or $f_{i,j}$, $j \in \{1, 2\}$, can occupy z at time step 1. Moreover, exactly two validity paths with sinks u_i and w_i must be at z at time step 1, since by Observation 2(i), each edge $\{z, u_i\}$ and $\{z, w_i\}$ is available only at time steps 1 and 4 for exactly two of the four validity paths for i. We thus have the following.

Lemma 3 (★). *Let I' be a* yes*-instance. For every solution and every $i \in \{1, \dots, N\}$, on time step* 1*, edges $\{z, u_i\}$, $\{z, w_i\}$, $\{z, v_{i,1}\}$, and $\{z, v_{i,2}\}$, are taken either by the two validity paths $P(t_{i,1}, u_i)$ and $P(t_{i,2}, w_i)$ or by the two validity paths $P(f_{i,1}, w_i)$ and $P(f_{i,2}, u_i)$.*

Combining Lemma 3 with Observation 2(ii), we now get that if there are verifier paths at z at time step 2, then they departed at time step 1 either from $t_{i,1}$ and $t_{i,2}$ or from $f_{i,1}$ and $f_{i,2}$.

Lemma 4 (★). *Let I' be a* yes*-instance. For every solution and every $i \in \{1, \dots, N\}$, there are at most two verifier paths for i located at z at time step 2, and if there are exactly two, then they start either from $t_{i,1}$ and $t_{i,2}$ or from $f_{i,1}$ and $f_{i,2}$.*

Intuitively, verifier paths arriving at z at time step 2 induce a satisfying truth assignment. By Lemma 4, no two validity paths at z at time step 2 correspond to different assignments of the same variable. By Lemma 3 and Observation 2, verifier paths can reach z only at time steps 2, 3, and 4. Since each clause vertex is the sink of three verifier paths, at least one must be at z at time step 2.

Proof of Theorem 6. Let instance $I' = (\mathcal{G}, \mathcal{P})$ of SRDG with $\mathcal{G} = (V, E, \emptyset, c, \theta, d, \tau)$ and $\tau = 4$ be obtained from instance $I = (X, \phi)$ of (2,2)-3SAT with $X = \{x_1, \dots, x_N\}$ and $\phi = \bigwedge_{j=1}^{M} C_j$ using Construction 3 in time polynomial in $|I|$. We claim that I is a yes-instance if and only if I' is a yes-instance (for the forward direction, see a full version of the paper).

($\Leftarrow$) Let π be a valid temporalization. Construct $\beta\colon X \to \{\top, \bot\}$ as follows. For each $i \in \{1, \dots, N\}$, set $\beta(x_i) = \top$ if there is no verifier path for i at z at time step 2 or there is a verifier path at z at time step 2 starting from $t_{i,j}$, $j \in \{1, 2\}$. Set $\beta(x_i) = \bot$ if there is a verifier path at z at time step 2 starting from $f_{i,j}$, $j \in \{1, 2\}$. Note that β is well-defined due to Lemma 4. We claim that β is a satisfying truth assignment. Suppose towards a contradiction that there is clause C_j not satisfied by β. Then none of the three verifier paths ending in c_j are located at z at time step 2. Thus, each of the three verifier paths arrives at z earliest at time step 3 (recall that no verifier path is at z at time step 1). Hence, only two time steps remain for the three paths to proceed to c_j, a contradiction. □

4 INTEGER LINEAR PROGRAMMING

We give an integer linear program (ILP) for SRDG. Let

$$\forall P \in \mathcal{P}, (v, w) \in X(P): \qquad x^P_{v,w} \in \{1, \dots, \tau\} \tag{10}$$

be the time when P leaves v towards w. We first ensure adequacy. We write $\theta(v, w)$ short for $\theta((v, w))$ and $\theta(\{v, w\})$ (same for d).

$$\forall P \in \mathcal{P}, (u, v), (v, w) \in X(P): \quad x^P_{u,v} + \theta(u, v) \le x^P_{v,w} \tag{11}$$

$$\forall P \in \mathcal{P}, (v, w) \in X(P): \quad x^P_{v,w} + \theta(v, w) \le d(v, w) \tag{12}$$

Next we ensure that any two paths are temporally edge-disjoint. Let $\overleftarrow{X}(P) \coloneqq \{(w, v) \mid (v, w) \in X(P)\}$.

$$\begin{aligned} &\forall P, P' \in \mathcal{P}, (v, w) \in X(P) \cap X(P'): |x^P_{v,w} - x^{P'}_{v,w}| \ge 1 \\ &\forall P, P' \in \mathcal{P}, \{v, w\} \in E \text{ s.t.} \\ &(v, w) \in X(P) \cap \overleftarrow{X}(P'): |x^P_{v,w} - x^{P'}_{w,v}| \ge \max\{1, \theta(\{v, w\})\}. \end{aligned} \tag{13}$$

For the vertex-capacity constraints, we follow the idea of a "sweeping line" to find the largest clique in an interval graph. Observe that the time steps at which a path is located at some vertex v forms an interval. We introduce the following variables.

$$\forall v \in V, P, Q \in \mathcal{P}(v): \qquad \alpha^{P,Q}_v, \beta^{P,Q}_v, \gamma^{P,Q}_v \in \{0, 1\} \tag{14}$$

We define the arrival and departure times for a path P at its vertex v with predecessor u and successor w (if existent): $t^P_{\text{arr}}(v) = x^P_{u,v} + \theta(u, v)$ if $v \neq s(P)$, and $t^P_{\text{arr}}(v) = x^P_{v,w}$ otherwise; $t^P_{\text{dep}}(v) = x^P_{v,u}$ if $v \neq t(P)$, and $t^P_{\text{dep}}(v) = x^P_{u,v} + \theta(u, v)$ otherwise. We ensure next that $\gamma^{P,Q}_v = 1$ if Q is located at v when P arrives at v, i.e., when $t^P_{\text{arr}}(v) \in [t^Q_{\text{arr}}(v), t^Q_{\text{dep}}(v)]$. Let M be a sufficiently large number.

$$\begin{aligned} \forall v \in V, P, Q \in \mathcal{P}(v): \; t^P_{\text{arr}}(v) &\le t^Q_{\text{arr}}(v) - 1 + M \cdot (1 - \alpha^{P,Q}_v) \\ t^P_{\text{arr}}(v) &\ge t^Q_{\text{dep}}(v) + 1 - M \cdot (1 - \beta^{P,Q}_v) \\ 1 &= \gamma^{P,Q}_v + \alpha^{P,Q}_v + \beta^{P,Q}_v \end{aligned} \tag{15}$$

Note that if $\gamma^{P,Q}_v = \gamma^{P,Q'}_v = 1$, then also Q and Q' must be intersecting (at least in $t^P_{\text{arr}}(v)$). Observe that the arrival times are sufficient: For every vertex v consider any two paths arriving at v at time steps t_1 and t_2 such that no other path arrives between t_1 and t_2. Then the number of paths located at vertex v is not increasing in $\{t_1, \dots, t_2 - 1\}$, since other paths can only depart at these time steps. Hence, to ensure that the capacity constraints are respected, we only need to count all the overlaps for each path via γ.

$$\forall v \in V, P \in \mathcal{P}(v): \qquad \sum\nolimits_{Q \in \mathcal{P}(v)} \gamma^{P,Q}_v \le c(v) \tag{16}$$

Our final ILP for SRDG is composed of (10)–(16) and provides a valid temporalization for every feasible instance.

Optimization goal. When an extreme event like flooding is predicted to happen, practically we wish to know the minimum in-advance starting time d^* to obtain a feasible evacuation schedule. To this end, we only need to adjust (12) (and possibly M) in our ILP.

$$\begin{aligned} &\min d^* \text{ s.t.} \quad (10), (11), (13)\text{–}(16), \text{ and} \\ &\forall P \in \mathcal{P}, (v, w) \in X(P): \quad x^P_{v,w} + \theta(v, w) \le d^* + d(v, w) \end{aligned} \tag{17}$$

Table 3: Overview on `osm` for the top five cities. For the connections we give the percentages of one-way arcs (O), antiparallel arcs (T), and undirected edges (U). diam denotes the diameter of G. For θ, d, and c, we give the mean along with [min, max].

City	# Vertices (in zones 0, A, B)		# Connections (O, T, U)		diam	θ	d	c
Gera	2487	(223, 243, 485)	3197	(17.8%, 21.0%, 61.2%)	88	[1.0, 527.0] 14.2	[13.0, 9465.0] 1958.2	[2.0, 10.0] 4.7
Kitzingen	707	(83, 85, 188)	904	(14.3%, 16.8%, 68.9%)	56	[1.0, 117.0] 12.4	[50.0, 3576.0] 1051.7	[2.0, 8.0] 4.8
Koblenz	2961	(278, 621, 864)	3886	(35.5%, 14.4%, 50.2%)	100	[1.0, 463.0] 11.5	[22.0, 4658.0] 1035.1	[1.0, 8.0] 4.4
Trier	2777	(252, 331, 719)	3680	(31.0%, 19.7%, 49.3%)	124	[1.0, 254.0] 11.6	[63.0, 6201.0] 1590.9	[1.0, 8.0] 4.5
Zell (Mosel)	246	(135, 47, 58)	334	(20.4%, 10.2%, 69.5%)	48	[1.0, 349.0] 13.7	[82.0, 1070.0] 322.0	[2.0, 8.0] 4.9

5 EXPERIMENTS

We implemented the ILP (17) using Gurobi(py) 12.0. in two ways, where we implemented (15) as outlined, referred to as `bigM`, and using Gurobi indicator constraints instead, referred to as `indic`. We set M for `bigM` as follows. Greedily color paths such that every two paths of the same color have disjoint vertex sets. Let $\mathcal{P}_1, \dots, \mathcal{P}_\chi$ be the resulting partition of the paths with χ colors. Set $M = \sum_{x \in \{1,\dots,\chi\}} \max_{P \in \mathcal{P}_x} (1 + \sum_{(u,v) \in X(P)} \theta(u,v))$, which corresponds to iteratively, one color after the other, initiating a traversal without waiting for all paths from the same color starting at the same time when the latest path from the preceding color finished. For both implementation we give a warm start regarding d^*: for `bigM`, we use the ILP relaxation, and for `indic`, we take the largest violated deadline when every path starts at time step 1 without waiting to its sink. For each instance, we let both implementations compete in parallel and only recorded the runtime of the winner.

Our experiments ran on Intel® Xeon® Silver 4310 CPU at 2.10GHz (12 cores), 125 GB RAM, Ubuntu 22.04.3 LTS (x86_64).

5.1 Data

We conducted experiments on artificial decaying paths and stars and semi-artificial decaying graphs where the networks are obtained from ten German cities each located at at least one river.

Artificial Data. For each number $n \in \{8, 12, 16\}$ of vertices, we create decaying stars and decaying paths. Here, for every two vertices that are supposed to be adjacent (e.g., any leaf and the center for stars, and v_i and v_j with $|i - j| = 1$ for paths), we equally likely connect them either with a single arc, two bidirectional arcs, or an undirected edge (a single arc's direction is also equally likely drawn). The traversal times are picked uniformly at random from the set $\{5, \dots, 20\}$ (corresponding to street segments of roughly 70 m to 280 m at 50 km h^{-1}). We add $p = p^* \cdot n$ many paths to each decaying path and decaying star, where $p^* \in \{0.5, 0.67, 0.83, 1\}$ for decaying paths and $p^* \in \{0.5, 1, 1.5, 2\}$ for decaying stars.[3] Herein, a source is picked uniformly at random, and then extended. For decaying paths, we additionally restrict the paths length $\ell = \ell^* \cdot n$ with $\ell^* \in \{0.33, 0.44, 0.55, 0.66\}$. From the source, two auxiliary paths start to the left and right, each of which stops latest when having length ℓ. Then, we only add the path of larger length, and a fair coin decides when both have the same length. For decaying stars, when the source is a leaf, a fair coin decides whether the end vertex is the center or another leaf picked uniformly at random from all reachable leaves. For each vertex $v \in V$, we set $c(v) = \max\{1, \lceil c^* \cdot |\mathcal{P}(v)| \rceil\}$ with $c^* \in \{0.1, 0.4, 0.7, 1\}$ (i.e., $c^* = 1$ yields an uncapacitated instance). For each connection e we set the deadline $d(e) = d^* \cdot d_{\text{lb}}(e)$ with $d^* \in \{0.2, 0.47, 0.73, 1\}$, where $d_{\text{lb}}(e)$ is $\theta(e)$ plus the maximum of the number of paths using connection e and the latest arrival time of any path using connection e when each paths is routed independently and earliest without waiting. We constructed 1920 instances of the form $I\_n\_p^*\_c^*\_d^*$-x for decaying stars (referred to by `stars`) and 7680 instances of the form $I\_n\_p^*\_c^*\_d^*\_\ell^*$-x for decaying paths (referred to by `paths`), with $x \in \{0, 1, \dots, 9\}$ (i.e., 10 instances per constellation). See a full version of the paper for more details on our constructed instances.

[3] Throughout, we round to the next integer if required and not specified otherwise.

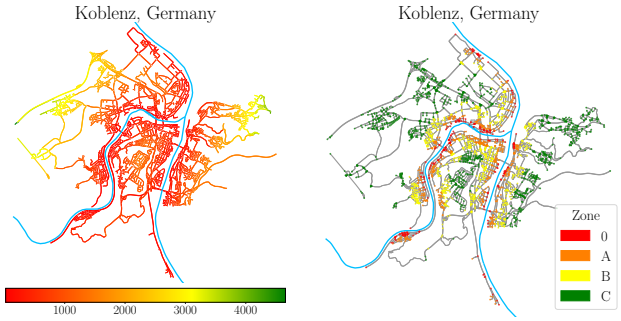

Figure 5: Illustration for one `osm` with waterways depicted blue and color-codings for (left) deadlines and (right) zones.

Semi-artificial Data. We chose the ten German cities that have, according to [13], the highest ratio of addresses under severe threat of flooding (see Figure 5 and Table 3 for the top five). From OSM data, we extracted the street network and the main river(s) of each city (assuming these to cause flooding). For the network, if two nodes are connected by a single lane traversable both-ways, we made this connection an undirected edge. For any connection, we set its travel time (in seconds) to its length divided by the maximum allowed speed of 50 km h^{-1}; Its deadline we set to the smallest distance of its two endpoints to the river(s) divided by a flood speed of 1 m s^{-1} (i.e., the farther, the later it ceases). For each vertex, we set the capacity to the number of incident connections (each serves as a waiting spot). Moreover, we assigned each vertex v to a zone, determined by v's closest distance $\text{dist}(v)$ to the river(s): vertex v is in zone 0 if $\text{dist}(v) \leq 250$ m, in zone A if 250 m $< \text{dist}(v) \leq 500$ m, in zone B if 500 m $< \text{dist}(v) \leq 1000$ m, and in zone C if $\text{dist}(v) > 1000$ m. Zone 0 will always be evacuated. Let $\zeta = (0, A, B, C)$. The path set is constructed in the following way, where $p^* \in [0, 1]$ determines

Table 4: Average runtime [sec] for `osm` of top five cities along with [min, max] runtimes.

City	(0.1, A)	(0.2, A)	(0.3, A)	(0.1, B)	(0.2, B)	(0.1, C)
Gera	[0.06, 0.18] 0.13	[0.31, 1.32] 0.59	[0.76, 6.82] 2.34	[0.33, 0.88] 0.49	[1.80, 10.06] 3.97	[1.39, 7.84] 2.97
Kitzingen	[0.02, 0.05] 0.03	[0.02, 0.09] 0.06	[0.03, 0.19] 0.11	[0.02, 0.06] 0.04	[0.03, 0.27] 0.16	[0.22, 1.61] 0.42
Koblenz	[0.07, 0.49] 0.26	[0.79, 7.33] 2.29	[1.59, 7.56] 3.82	[1.15, 7.08] 2.32	[7.66, 40.80] 18.02	[12.32, 97.08] 35.87
Trier	[0.17, 0.88] 0.31	[0.54, 3.21] 1.01	[1.09, 10.71] 3.63	[0.68, 2.31] 1.10	[3.49, 18.73] 7.15	[5.40, 29.56] 10.92
Zell (Mosel)	[0.06, 1.11] 0.33	[0.41, 11.07] 2.32	[2.04, 170.05] 18.49	[0.33, 98.06] 7.62	[3.33, 347.86] 25.05	[0.30, 105.16] 9.65

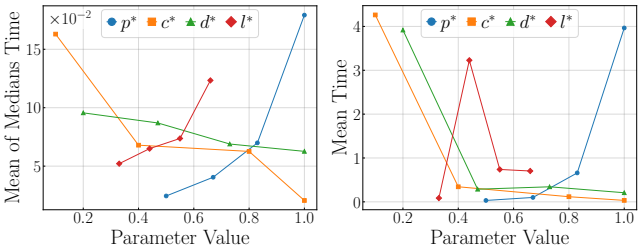

Figure 6: Mean of medians (left) and mean (right) over all runtimes [sec] for solving `paths` when a parameter is fixed.

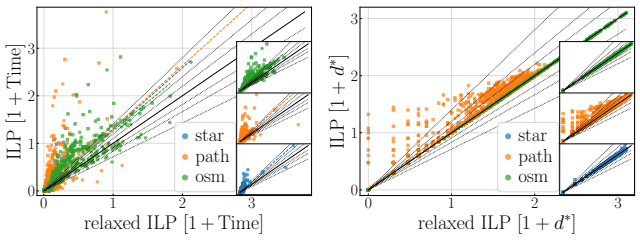

Figure 7: Scatter log-log plot for relaxed ILP versus ILP regarding runtime (left) and d^* (right) on `stars`, `paths`, and `osm`.

the number of paths: For a zone $\zeta_i \in \{A, B, C\}$ with $i \geq 2$, for $p^* \cdot |\bigcup_{j=1}^{i-1} \zeta_j|$ times we pick a random source v from zones $\bigcup_{j=1}^{i-1} \zeta_j$, then pick a random sink w from the 20% of the vertices closest to v in zones $\bigcup_{j=i}^{4} \zeta_j$, and add a shortest (v, w)-path. We constructed 600 instances of the form *I_i_p*_z-x* (referred to by `osm`), where $x \in \{0, 1, \ldots, 9\}$ and $i \in \{0, 1, \ldots, 9\}$ corresponds to the index of the German city at hand, for each tuple (p^*, z) in $\{(0.1, X) \mid X \in \{A, B, C\}\} \cup \{(0.2, Y) \mid Y \in \{A, B\}\} \cup \{(0.3, A)\}$.

5.2 Runtime Comparisons

We analysed the mean over all constellations, where for each constellation we either took the median or the mean over all $x \in \{0, 1, \ldots, 9\}$. See Figure 6 for our results for `paths` (for `stars` we have similar conclusions; see a full version of the paper). Our results for the median indicate that runtime increases strongly with higher p or smaller c, and moderately with higher ℓ (for `paths`). We detect no correlation with d. The correlation with p and ℓ can be explained since the number of variables depends quadratically and linearly on p and ℓ, respectively. The correlation with c follows our intuition that d^* is monotonically increasing with decreasing c. Considering the mean reveals severe outliers, showing that the random assignments of the traversal times and paths can have a severe impact on the ILP's performance. In fact, for some few instances the computation time is very large (up to two hours). Implementation `indic` is faster than `bigM` on 54.8% of `stars` and on 62.7% of `paths`.

For our results on `osm`'s top five, see Table 4. Our results show that our ILP can handle, e.g., 10%-zone-C evacuation in less than 40 seconds on average (for, e.g., Koblenz, these 10% correspond to 176 evacuation paths). Our results for zone-A and zone-B evacuation indicate that increasing the number of paths severely impacts the running time (for zone-C evacuation, even for 15% our machines did not terminate in reasonable time). Implementation `indic` performed faster than `bigM` on 73.3% of `osm`.

5.3 Performance of the ILP Relaxation

The ILP relaxation performs surprisingly good for `osm`, see Figure 7: On average, the ILP relaxation needs 98.2% (91.8% in the median) of the running time of the best exact ILP and yet computes on almost all instances the optimal d^* from the exact ILP. Also for `paths` and `stars` the solution quality is quite high with 97.7% and 99.8%, respectively. Yet, the standard deviation for `paths` is significantly larger than for `stars`. On both `paths` and `stars`, the average runtime of the ILP relaxation is even larger than the one of the best exact ILP, which means that in many cases, `indic` performs fast.

6 EPILOGUE

We proved that smooth evacuation is a computationally hard task even in well-structured decaying trees under quite restrictive settings. Given the complexity, we formulated an ILP. Our experiments demonstrate the practicality of our ILP. Facing larger cities, larger path sets, or longer paths, our recommendations are as follows: (1) Start both of our implementations and the relaxed ILP in parallel on separate machines. (2) Use powerful clusters (our machine already failed to handle instances slightly larger than our largest).

Our work opened several future pathways. On the theoretical side, we wonder about the computational complexity on uncapacitated decaying paths and stars. Further restrictions on the path set such as only few vertices being eligible as sinks is also well-motivated. On the experimental side, we wish to handle larger instances. For this, reduction rules may be beneficial. For both sides, decaying caterpillars—generalizing both stars and paths—may be of interest, in particular in terms of small deadlines.

ACKNOWLEDGMENTS

Till Fluschnik acknowledges support by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project PACS (FL 1247/1-1, 522475669).

REFERENCES

[1] Eleni C. Akrida, Jurek Czyzowicz, Leszek Gasieniec, Lukasz Kuszner, and Paul G. Spirakis. 2019. Temporal flows in temporal networks. *J. Comput. Syst. Sci.* 103 (2019), 46–60.

[2] Shaull Almagor, Justin Kottinger, and Morteza Lahijanian. 2024. Temporal segmentation in multi agent path finding with applications to explainability. *Artif. Intell.* 330 (2024), 104087.

[3] Dor Atzmon, Roni Stern, Ariel Felner, Glenn Wagner, Roman Barták, and Neng-Fa Zhou. 2020. Robust Multi-Agent Path Finding and Executing. *J. Artif. Intell. Res.* 67 (2020), 549–579.

[4] Jan Boeckmann, Clemens Thielen, and Alina Wittmann. 2023. Complexity of the Temporal Shortest Path Interdiction Problem. In *Proceedings of the 2nd Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2023) (LIPIcs, Vol. 257)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 9:1–9:20.

[5] Arnaud Casteigts, Anne-Sophie Himmel, Hendrik Molter, and Philipp Zschoche. 2021. Finding Temporal Paths Under Waiting Time Constraints. *Algorithmica* 83, 9 (2021), 2754–2802.

[6] Markus Chimani and Niklas Troost. 2023. Multistage Shortest Path: Instances and Practical Evaluation. In *Proceedings of the 2nd Symposium on Algorithmic Foundations of Dynamic Networks SAND (LIPIcs, Vol. 257)*, David Doty and Paul G. Spirakis (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:19.

[7] Andreas Darmann and Janosch Döcker. 2021. On simplified NP-complete variants of Monotone 3-Sat. *Discrete Applied Mathematics* 292 (2021), 45–58.

[8] Tanka Nath Dhamala. 2015. A survey on models and algorithms for discrete evacuation planning network problems. *Journal of Industrial & Management Optimization* 11, 1 (2015), 265.

[9] Till Fluschnik, Marco Morik, and Manuel Sorge. 2019. The complexity of routing with collision avoidance. *J. Comput. System Sci.* 102 (2019), 69–86.

[10] Till Fluschnik, Rolf Niedermeier, Carsten Schubert, and Philipp Zschoche. 2023. Multistage s-t Path: Confronting Similarity with Dissimilarity. *Algorithmica* 85, 7 (2023), 2028–2064.

[11] Jianqi Gao, Yanjie Li, Xinyi Li, Kejian Yan, Ke Lin, and Xinyu Wu. 2024. A review of graph-based multi-agent pathfinding solvers: From classical to beyond classical. *Knowl. Based Syst.* 283 (2024), 111121.

[12] M.R. Garey, D.S. Johnson, and L. Stockmeyer. 1976. Some simplified NP-complete graph problems. *Theoretical Computer Science* 1, 3 (1976), 237–267.

[13] Gesamtverband der Deutschen Versicherungswirtschaft (GDV). 2024. Amtliche Zahlen zeigen: Mehr als 300 000 Adressen in Deutschland sind von Hochwasser bedroht. GDV Pressemitteilung. https://gdv.de/gdv/medien/medieninformationen/amtliche-zahlen-zeigen-mehr-als-300000-adressen-in-deutschland-sind-von-hochwasser-bedroht-168828 Last accessed on August 29, 2025.

[14] Yuya Higashikawa, Naoki Katoh, Junichi Teruyama, and Yuki Tokuni. 2024. Faster algorithms for evacuation problems in networks with a single sink of small degree and bounded capacitated edges. *J. Comb. Optim.* 48, 3 (2024), 18.

[15] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA (The IBM Research Symposia Series)*. Plenum Press, New York, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

[16] Nina Klobas, George B. Mertzios, Hendrik Molter, Rolf Niedermeier, and Philipp Zschoche. 2023. Interference-free walks in time: temporally disjoint paths. *Auton. Agents Multi Agent Syst.* 37, 1 (2023), 1.

[17] Pascal Kunz, Hendrik Molter, and Meirav Zehavi. 2023. In Which Graph Structures Can We Efficiently Find Temporally Disjoint Paths and Walks?. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China*. ijcai.org, 180–188.

[18] Todd Litman. 2006. Lessons from Katrina and Rita: What major disasters can teach transportation planners. *Journal of Transportation Engineering* 132, 1 (2006), 11–18.

[19] Gopinath Mishra, Subhra Mazumdar, and Arindam Pal. 2018. Improved algorithms for the evacuation route planning problem. *Journal of Combinatorial Optimization* 36, 1 (2018), 280–306.

[20] Aniello Murano, Giuseppe Perelli, and Sasha Rubin. 2015. Multi-agent Path Planning in Known Dynamic Environments. In *Proceedings of the 18th International Conference on Principles and Practice of Multi-Agent Systems (PRIMA 2015) (Lecture Notes in Computer Science, Vol. 9387)*. Springer, 218–231.

[21] Arseni Pertzovskiy, Roni Stern, Roie Zivan, and Ariel Felner. 2025. Multi-Agent Corridor Generating Algorithm. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2025)*. ijcai.org, 240–247.

[22] Urmila Pyakurel, Hari Nandan Nath, Stephan Dempe, and Tanka Nath Dhamala. 2019. Efficient Dynamic Flow Algorithms for Evacuation Planning Problems with Partial Lane Reversal. *Mathematics* 7, 10 (2019).

[23] Hermann Schichl and Meinolf Sellmann. 2015. Predisaster Preparation of Transportation Networks. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'15)*. AAAI Press, 709–715.

[24] Kaveh Shahabi and John P. Wilson. 2014. CASPER: Intelligent capacity-aware evacuation routing. *Comput. Environ. Urban Syst.* 46 (2014), 12–24.

[25] Roni Stern, Nathan R. Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T. Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, T. K. Satish Kumar, Roman Barták, and Eli Boyarski. 2019. Multi-Agent Pathfinding: Definitions, Variants, and Benchmarks. In *Proceedings of the 12th International Symposium on Combinatorial Search (SOCS 2019)*, Pavel Surynek and William Yeoh (Eds.). AAAI Press, 151–158.

[26] XiaoJian Wu, Daniel Sheldon, and Shlomo Zilberstein. 2016. Optimizing Resilience in Large Scale Networks. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI'16)*. AAAI Press, 3922–3928.

[27] Jingjin Yu and Steven M. LaValle. 2012. Multi-agent Path Planning and Network Flow. In *Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics (WAFR 2012) (Springer Tracts in Advanced Robotics, Vol. 86)*. Springer, 157–173.

[28] Jingjin Yu and Steven M. LaValle. 2013. Structure and Intractability of Optimal Multi-Robot Path Planning on Graphs. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*. AAAI Press, 1443–1449.

[29] Jingjin Yu and Steven M. LaValle. 2016. Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics. *IEEE Trans. Robotics* 32, 5 (2016), 1163–1177.