The Complexity of Routing with Collision Avoidance

Fakultät IV
Fachgebiet Algorithmik und Komplexitätstheorie
Bachelor Thesis
2016

Marco Morik

Erstgutachter: Prof. Dr. Rolf Niedermeier
Zweitgutachter: Prof. Dr. Stephan Kreutzer

Betreuer: Till Fluschnik, Manuel Sorge

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.
Berlin, den

Unterschrift

Zusammenfassung

In dieser Arbeit untersuchen wir die Komplexität der Bestimmung von Routen mit Kollisionsvermeidung. Eine Kollision entsteht, wenn zwei geplante Pfade in einem Graphen zur selben Zeit die gleiche Kante benutzen. Dafür definieren wir das Minimum Time-Shared Edges-Problem. Gegeben wird ein Startknoten s und ein Zielknoten t, ein gerichter oder ungerichteter Graph G, eine Anzahl p von Pfaden und eine maximale Anzahl k von Kollisionen bzw. geteilten Kanten. Die Frage ist, ob es k Pfade von s nach t in G gibt, die maximal k Kanten teilen. Eine Kante ist geteilt, wenn es mehr als einen Pfad gibt, die diese Kante zum selben Zeitpunkt nutzen. Wurde eine Kante geteilt, kann sie von beliebig vielen Pfaden auch gleichzeitig benutzt werden. Wir betrachten in unserem Problem ein diskretes Zeitmodell, in dem jeder Pfad für jede benutze Kante genau einen Zeitschritt braucht.

In unserer Komplexitätsanalyse unterscheiden wir zwischen drei verschiedenen Definitionen von Pfaden: Der „*path*“, welcher jeden Knoten nur maximal einmal besuchen darf, der „*trail*“, welcher jede Kante maximal einmal benutzen darf und der „*walk*“, welcher sowohl Kanten als auch Knoten mehrfach benutzen kann.

Da Routing Probleme Anwendungen im Straßenverkehr haben und Straßennetze als planare Netzwerke gesehen werden können, untersuchen wir die Komplexität auf sowohl gerichteten als auch ungerichteten planaren Graphen. Auch für Abwassersysteme kann Minimum Time-Shared Edges genutzt werden, um herauszufinden, welche Rohre ausgebessert werden müssen. In Kanalsystemen haben Rohre oft eine Fließrichtung bergab, daher untersuchen wir gerichtete azyklische Graphen.

Des weiteren untersuchen wir die Komplexität, wenn wir keine Kollision zulassen, also keine geteilte Kante erlaubt ist. Wir zeigen, dass wir Minimum Time-Shared Edges mit „walks“ so sowohl auf ungerichteten als auch gerichteten Graphen in polynomieller Zeit lösen können. Für „paths“ und „trails“ zeigen wir, dass Minimum Time-Shared Edges aber sogar auf planaren Graphen NP-vollständig ist.

Auf gerichteten azyklischen Graphen ist jeder Pfad ein „path“, da keine Knoten und Kanten doppelt verwendet werden können. Während Minimum Time-Shared Edges auch auf diesen Graphen mit geteilten Kanten NP-vollständig ist, können wir für keine geteilte Kante einen Polynomialzeitalgorithmus präsentieren. Als letztes führen wir das Short Minimum Time-Shared Edges-Problem ein, in welchem die Länge des längesten Pfades beschränkt ist. Wir beweisen, dass dieses Problem sowohl für „paths“ als auch „trails“als auch „walks“ auf ungerichteten Graphen NP-vollständig ist.

Summary

In this work, we analyze the complexity of Routing with Collision Avoidance. A collision occurs when two s-t connections in a graph use the same edge at the same time. To tackle this problem, we introduce the Minimum Time-Shared Edges problem. Given a source s and a sink t, a directed or undirected Graph G, a number p of connections and an upper bound k on collision edges, we are looking for p s-t connections in G with at most k edges with collision. We later refer to a collision as a shared edge, since once a collision occurs, this edge can be used by an unrestricted number of connections. An edge is shared when there are at least two s-t connections using this edge at the same time. We use a discrete time model, where each connection needs one step per used edge.

We differentiate between three different versions of s-t connections. A *path* can contain each vertex at most once, a *trail* can use each edge at most once, and a *walk* can use both edges and vertices an unrestricted number of times.

Since routing problems have an application in road traffic and street networks can be seen as planar graphs, we analyze the complexity of Minimum Time-Shared Edges both on directed and undirected planar graphs. We can use Minimum Time-Shared Edges also for wastewater systems to determine which drains need to be enlarged. In canal systems, drains have often a direction of flow downhill, therefore, we analyze the complexity of Minimum Time-Shared Edges on directed acyclic graphs (DAGs).

Moreover, we study the complexity for the case that we do not allow any collision, that is $k = 0$ shared edges. For $k = 0$, we show how to solve Walk Minimum Time-Shared Edges both on undirected and directed graphs in polynomial time. For paths and trails, we prove that Minimum Time-Shared edges remains NP-complete even with $k = 0$ on directed and undirected planar graphs.

On DAGs, each s-t connection is a path, since neither vertices nor arcs can appear more than once. We show that with shared arcs Minimum Time-Shared Edges is NP-complete. However, for $k = 0$ shared edges, we provide a polynomial-time algorithm.

Last, we introduce the Short Minimum Time-Shared Edges problem. Here, the length of the longest s-t connections is upper bounded. We prove that this problem is NP-complete for trails and paths as well as walks on undirected graphs.

Contents

1 Introduction

Some everyday problems like logistic problems, traffic problems, or security problems can be considered routing problems. Think, for instance, of the maintenance of wastewater systems. For the maintenance of wastewater networks, we cannot flood all drains at the same time, but have to take into account when water reaches each drain. The overall flow of the wastewater system should be constructed such that the flow never exceeds the maximal capacity. We can look for the minimal amount of drains to enlarge to provide a given flow.

A first approach to solve the wastewater maintenance is to describe the problem as a maximum flow problem. We characterize locations where a drain flows into another one by vertices and all drains by edges. However, the maximum flow does not consider the time an edge is used. Each drain would only be used once and stay empty the rest of the time. A more appropriate model which also considers the time is that of a dynamic maximum flow; it can be solved in polynomial time [FF58]. However, in this model, water could stay still, which is quite unrealistic. Moreover, both maximal flow models characterize the network as it is. If we want to detect bottlenecks in order to enhance the network, we need a different model. There, we do not want to constrain edges which we want to enhance, but consider them having infinite capacity.

For another example, consider security problems. For instance, a very important person arrives at a location, e.g., an airport, and needs to travel to a goal, e.g., the parliament. Several actors play the role of that VIP and are each using a different limousine so that no assassin can be sure to target the VIP when attacking a car surrounded by police forces. Of course, two such cars should not be at the same place at the same time! This kind of *collision* must be avoided. A similar example is that of guiding fans of opposing football clubs to the stadium or train station. They should not *share* a street, because they could start a fight.

These problems can be described in a uniform manner. In all examples, we see a network. The drains or the streets are edges and the junctions or intersections are the vertices. An example is shown in Figure 1. When an event ends, people want to go from the stadium to the subway station. Each edge represents a street and each vertex represents an intersection. We want to route p opposing groups of fans from the stadium s to the subway station t in a way that they share at most k edges. Without the aspect of time, we can use the Minimum Shared Edges problem introduced by Omran et al. [OSZ13]. It was originally designed for the guidance of VIPs. For each street used by more than one limousine, one has to hire a guard to protect this street which can afterwards be used by an unlimited number of limousines without extra costs. The problem is to route p paths from a start s to a goal t and minimize the number k of shared streets, in order to minimize the additional security personal. Nevertheless, the security problem needs to include time aspects for a realistic model. The limousines might drive on the same road, but not at the same time. This would not increase the risk of a successful attack, since an assassin cannot attack both cars. Also the opposing groups of fan can use the same road, as long as they are not on this road at the same time. We introduce the Minimum Time-Shared Edges (MTSE) problem to add the

Figure 1: A map of the Berliner Olympiastadium and its closest subway station with the underlying graph [Goo16].

dimension of time to this model. The definition is similar to Minimum Shared Edges, we also route p paths from s to t with at most k shared edges. But we call an edge shared only when at least two paths use this edge at the same time. For simplicity reasons, we consider a discrete time model where each path takes exactly one *step* per edge. An edge is shared if there are at least two paths which use the edge after the same amount of steps. In this work, we investigate the complexity of MTSE not only for paths, but also for trails and walks on different types of graphs. Moreover, we study the complexity if we restrict the number of shared edges to zero.

1.1 Preliminaries

Before we define our problem, we need to clearify some notations used in this work.

First we state some basic mathematical conventions. With $\mathbb{N}$ we describe the natural numbers without zero and with $\mathbb{N}_0$ the natural numbers including zero: $\mathbb{N}_0 := \mathbb{N} \cup \{0\}$. To get the positive integers up to $\ell \in \mathbb{N}$, we define $[\ell] := \{1, \ldots, \ell\} \subseteq \mathbb{N}$ and to get a range of integers between $a, b \in \mathbb{N}$ with $b \geq a$ we define $[a, b] := \{a, a+1, \ldots, b-1, b\}$.

Notation of Graph Theory. A graph is defined as a tuple $G = (V, E)$, where V are the vertices and E are the edges. If we compare multiple graphs, then we write V_i for the vertices in graph G_i and E_i for the edges in graph G_i. In this work, we consider both,

directed and undirected graphs. For directed graphs, we call an edge arc and each arc $e \in E$ is defined as a tuple $e = (v_i, v_j)$. For the undirected graph each edge e is defined as a set of size two: $e = \{v_i, v_j\}$. A planar embedding of a graph is a representation of the graph on a plane. A vertex is on the *outer face* if it is not surrounded by edges and belongs to the unbounded face of the embedding.

- An *s-t walk* is a sequence of vertices $P = (v_1, \ldots, v_q)$ with $v_1 = s$ and $v_q = t$ such that for all $i \in [q-1]$ it holds that $(v_i, v_{i+1}) \in E$ for the undirected graph and $\{v_i, v_{i+1}\} \in E$ for the directed graph respectively. We call the indices in P, the use of one edge, *steps*. So, $v_i \in P$ is the vertex after i steps.
- An *s-t trail* is an *s-t* walk which can contain vertices several times, but contains each edge at most once.
- An *s-t path* is an *s-t* walk which contains both edges and vertices at most once.
- We define an *s-t connection* as a placeholder for either a path, trail or walk: connection $\in$ {path, trail, walk}.
  - The *length* of an *s-t* connection P is defined as $|P| - 1$, the number of edges in P.
  - An edge $e \in E$ is called *shared* if there exist at least two *s-t* connections $P_x = (v_1, \ldots, v_a)$, $P_y = (u_1, \ldots, u_b)$ in a solution for MTSE such that $\exists i \in [\min(|P_x|, |P_y|) - 1]$: $e = \{v_i, v_{i+1}\} = \{u_i, u_{i+1}\}$ in the undirected graph or $e = (v_i, v_{i+1}) = (u_i, u_{i+1})$ in the directed graph respectively. In words, an edge is shared if and only if at least two *s-t* connections use that edge after the same amount of steps.
  - If an edge is already used by an *s-t* connection after the ith step, then we call this edge *blocked* for this time. It can only be used by another *s-t* connection in this ith step if it is a shared edge.
  - Vertex-disjoint *s-t* connections are *s-t* connections which have no common vertex except s and t.
- The *degree* of a vertex in an undirected graph is defined as the number of incident edges degree(v)=$|\{\{v, v_i\} \in E \mid \forall v_i \in V\}|$. In a directed graph, we can distinguish between *in-degree* and *out-degree* of a vertex. The in-degree of a vertex v is defined as all incoming arcs, in-degree$(v) = |\{(v_i, v) \in E \mid \forall v_i \in V\}|$, and the out-degree is defined as all outgoing arcs, out-degree$(v) = |\{(v, v_i) \in E \mid \forall v_i \in V\}|$. The degree of a vertex in a directed graph is the sum of in-degree and out-degree.
- *Subdivision* is the expansion of an edge $e = \{v, u\}$ by adding a new vertex w and replace e with $\{v, w\}$ and $\{w, u\}$.
- We call a path of length x between two vertices, containing only vertices with degree two, an *x-chain* or *chain of length x*. An edge $\{v, u\}$ between two vertices can be expanded to an x-chain by subdividing $x - 1$ times.

- The distance between two vertices v, u is defined as the length of the shortest path P_{short} from v to u: $\text{dist}(v,u) := \text{length}(P_{\text{short}})$

A *flow network* consists of a directed graph $G = (V, E)$, a source $s \in V$, a sink $t \in V$, and a capacity function $u : E \to \mathbb{R}_{\geq 0}$ which assign every edge a capacity. An *s-t flow* is a function $f : E \to \mathbb{R}_{\geq 0}$ which satisfy the following two properties. First, the flow capacity condition which requires that for each edge the flow is upper bounded by the capacity of this edge: $\forall e \in E : 0 \leq f(e) \leq c(e)$. Second, the flow conservation condition which requires for each vertex except s and t that the incoming flow equals the outgoing flow: $\forall v \in V \backslash \{s,t\} : \sum_{(u,v)\in E} f((u,v)) = \sum_{(v,u)\in E} f((v,u))$. The value of an *s-t* flow is defined by the outgoing flow of the source s: $|f| := \sum_{(s,u)\in E} f((s,u)) - \sum_{(u,s)\in E} f((u,s))$. An *s-t* flow f is a maximum *s-t* flow if there is no *s-t* flow f' with $|f'| > |f|$. An *s-t cut* $C \subseteq E$ is a set of edges such that the graph $G \backslash C$ is partitioned in two disconnected sets $S, V \backslash S$ where $s \in S$ and $t \in V \backslash S$. The size of an *s-t* cut is defined by the sum of capacities of the edges in the cut: $\text{val}(C) := \sum_{e\in C} c(e)$. An *s-t* cut C is a minimum *s-t* cut if there is no cut C' separating s and t with $\text{val}(C') < \text{val}(C)$. The size of a minimum *s-t* cut is equal to the size of the maximum *s-t* flow. For more about network flows we refer to the work of Kleinberg and Tardos [KT05].

An extension to the static network flow is the *network flow over time*, also called *dynamic flow*. There is an additional *time horizon* T and each edge has a transit time $d : E \to \mathbb{N}_0$. The flow over time is a Lebesgueintegrable function for each edge $f_e : [0,T) \to \mathbb{R}_{\geq 0}$. An *s-t* flow over time $f_e(x)$ denotes the flow starting at the edge e at time step x. It arrives at the other vertex of the edge at time step $x + d(e)$. Flow over time is both defined over a discrete time model as well as a continuous time model. In this work, we only work with the discrete time model. The capacity condition requires that for each edge and each time step the capacity is not exceeded: $\forall e \in E, \forall x \in [0,T) : f_e(x) \leq c(e)$. The *excess for vertex v at time x* is the net amount of flow that enters the vertex up to time x: $\text{ex}(v,x) := \sum_{(u,v)\in E} \int_0^{x-d((u,v))} f_{(u,v)}(\sigma)d\sigma - \sum_{(v,u)\in E} \int_0^x f_{(v,u)}(\sigma)d\sigma$. The weak flow conservation condition allows flow to wait in a vertex and requires for each vertex $v \in V \backslash \{s\}$ and all time steps $x \in [0,T)$ that $\text{ex}(v,x) \geq 0$. Moreover, $\text{ex}(v,T) = 0$ must hold for each $v \in V \backslash \{s,t\}$. For strict flow conservation $\text{ex}(v,x) = 0$ must hold for all $v \in V \backslash \{s,t\}$ and all $x \in [0,T]$, here flow can not be stored in intermediate nodes. The *value* of an *s-t* flow over time is defined by $|f| := \text{ex}(t,T)$. For more about network flow over time we refer to the work of Skutella [Sku08].

In this work, we investigate our problem on different types of graphs. First we introduce *bipartite* graphs. In a bipartite graph one splits the vertices into two disjoint partitions U and V such that each edge connects a vertex in U with a vertex in V. Another simple graph is the *complete graph*. There every pair of vertices is adjacent.

A *planar* graph is a graph where an embedding in the plane, without intersections of edges exists. An alternative definition follows from Kuratowskis Theorem [Tho80]. A graph is planar if and only if it does not contain a subgraph which is a subdivision of K_5 or $K_{3,3}$. Here K_5 is the complete graph with five vertices and $K_{3,3}$ is the complete bipartite graph with three vertices in each partition and each vertex is adjacent to the three vertices in the other partition. The above graphs are defined both on undirected and

directed graphs. *Directed acyclic graphs* (DAGs) are directed graphs without directed cycles. This means that, for every vertex $v \in V$, there is no walk starting and ending in v, which uses at least one edge. The last graph we will use in this work is the *cubic* graph, it is an undirected graph where all vertices have degree three. For more about graph theory, we refer to the work of Diestel [Die00] and West et al. [Wes+01].

Computational Complexity Theory. There are basically two different kind of problems in computational complexity. First, there are the *decision problems*. A decision problem is a problem defined over some formal system with a yes-or-no answer depending on the input. On the other hand, there are *optimization problems*. Instead of a yes-or-no answer, a optimization problem has a cost function and tries to minimize or maximize this cost function. In this work, we use decision problems for complexity analysis. A *problem instance* is a specific input to a problem. We measure the complexity of a problem depending on the input size instead of some scalar value. In computational complexity, we analyze the asymptotic behavior of algorithms solving a problem depending on the input size. To measure this we use the Big O notation. An algorithm f running in $\mathcal{O}(g)$ denotes that the asymptotic running time of f is not essential faster than g. Formally, there exists a x_0 and a positive constant M such that $\forall x \geq x_0 : |f(x)| \leq M|g(x)|$.

Problems are classified in complexity classes. Classes are defined by their model of computation, in this thesis deterministic or nondeterministic Turing machines, their bounded resources, in most cases time or space, and their requirements to this resources. In this work, we handle the complexity classes P and NP. Both require the calculation time bounded by a polynomial depending on the input size, but for a problem in P there exists a polynomial-time algorithm on a deterministic Turing machine and for a problem in NP on a nondeterministic Turing machine. Clearly, P$\subseteq$NP, but whether P=NP or P$\subset$NP is an open problem. In this work, we assume that P$\neq$NP. Since all existing computers are deterministic, we call a problem polynomial-time solvable only if it is in P.

A problem is contained in NP if there is a deterministic polynomial-time algorithm that can verify a "yes"-answer to this problem. To show that a problem A is *NP-hard*, we can reduce an already known NP-hard problem B to the problem A in polynomial-time. A polynomial-time reduction from A to B is an algorithm running in polynomial-time which transform each instance of problem A to a corresponding instance of problem B. If there is an algorithm solving problem B efficiently, this algorithm can be used as a subroutine to solve problem A efficiently as well. When problem A is NP-hard, we assume that there is no algorithm solving A in polynomial-time. Consequently, if there is a polynomial-time reduction from A to B, then there cannot be an algorithm solving B in polynomial-time. If an NP-hard problem is also contained in NP, then we call this problem *NP-complete*.

There are several methods to deal with an NP-hard problem. Optimization problems can be solved with approximation algorithms which may not provide an optimal solution, but a solution upper-bounded by a certain function depending on the optimum. Other problems can be solved in polynomial-time when some input parameters are fixed. We

speak of *parameterized complexity*. If there is an algorithm solving a problem with running time in $f(k) \cdot n^{\mathcal{O}(1)}$, where k is an parameter, n the size of the input, and $f(x)$ an arbitrary function only depending on k, then this problem is called *fixed-parameter tractable* with respect to the parameter k. If there is an algorithm solving a problem with running time in $n^{f(k)}$ for some computational function $f(k)$ depending only on the parameter k, then this problem is in XP. For more about Computational Complexity, we refer to the book of Arora et al. [AB09]. For more information about parameterized complexity, we refer to the work of Downey and Fellows [DF13] and Niedermeier [Nie06].

1.2 Problem Definition

We define our main problem of interest as follows:

Problem: Minimum Time-Shared Edges (MTSE)
Input: A directed or undirected graph $G = (V, E)$, $s, t \in V$, $p \in \mathbb{N}$, and $k \in \mathbb{N}_0$.
Question: Are there p s-t connections in G that share at most k edges?

As defined in the preliminaries, an edge is shared when there are at least two connections using that edge at the same time. Since we consider a discrete time model where a connection needs exactly one step per edge, an edge is shared when it appears at the same position in at least two connections. The parameters k and p allow us to apply the definition to varying problems. The number k of edges that are allowed to be shared can be between zero and the overall number of edges. The amount p of s-t connections needs to be at least one. The definition covers graphs, be they directed or undirected. It also refers to s-t connections, which represent walks, trails, or paths. Hence, the definition is general. To differentiate between walks, trails, or paths, we refer to Walk-MTSE, Trail-MTSE, and Path-MTSE, respectively. If we refer to MTSE, then a statement is valid for both walks, trails and paths. We explicitly state whether we consider directed or undirected graphs. In this work, we address simple graphs without loops or multiple edges between vertices.

1.3 Related Work

MTSE was inspired by Minimum Shared Edges (MSE) introduced by Omran et al. [OSZ13]. The difference to Minimum Time-Shared Edges lies in the definition of a shared edge. While we consider an edge as shared when at least two paths use an edge after the same amount of steps, in MSE an edge is shared when it is contained in more than one path regardless of time. MSE is NP-complete on directed [OSZ13] and undirected graphs [Flu15] and admits no polynomial-time $2^{\log^{1-\epsilon} k}$-factor approximation[1] for any constant $\epsilon > 0$ [OSZ13]. Fluschnik et al. [Flu+15] showed that MSE is NP-complete on graphs with maximum degree at least five but can be solved in constant time

[1] An approximation algorithm outputs a solution which size is upper bounded by a certain factor of the optimal solution size.

on unbounded undirected $\mathcal{Z} \times \mathcal{Z}$-grid graphs. They also showed that MSE is W[2]-hard relating to the maximum number k of shared edges but fixed-parameter tractable with respect to the number p of paths. Ye et al. [Ye+13] showed that Minimum Shared Edges can be solved in $\mathcal{O}(|V| \cdot (p+1)^{2^{\omega \cdot (\omega+1)/2}} + |V| \cdot (p+1)^{(\omega+4)^{2 \cdot \omega + 8}})$ time, where ω is the treewidth and p is the number of desired s-t paths.

Minimum Vulnerability is a generalization of MSE introduced by Assadi et al. [Ass+14]:

Problem: Minimum Vulnerability (MV)
Input: Graph $G = (V, E)$, $s, t \in V$, edge costs $c_e : E \to \mathbb{R}_{\geq 0}$, edge capacity $u_e : E \to \mathbb{N}$, $r \in \mathbb{N}_0$, $p \in \mathbb{N}$, $k \in \mathbb{R}_{\geq 0}$
Question: Are there p s-t paths in G with costs of edges used in more than r s-t paths are at most k.

Assadi et al. showed that MV can be approximated with a $\lfloor \frac{k}{r+1} \rfloor$ factor which results in a $\lfloor \frac{k}{2} \rfloor$ approximation for MSE. Moreover, they showed that, for any constant k, MV can be solved exactly in polynomial time. Aoki et al. [Aok+14] showed that MV is NP-hard even on undirected bipartite series-parallel graphs[2] and undirected threshold graphs[3]. They showed that MV can be solved in polynomial time on graphs with bounded treewidth. The running time is $|V| \cdot (p+1)^{\mathcal{O}(\omega^{(\omega+1)})}$, where ω is the treewidth and p the number of desired s-t paths. Additionally, they proved that there is a fixed-parameter algorithm with respect to the number of paths p for MV on chordal graphs[4]. Another related concept is that of Dynamic Flow, also called Flow over Time, introduced by Ford and Fulkerson [FF58; FF62]. It generalizes the network flow by introducing the component of time. The dynamic flow in each arc is now stated per discrete time interval and there is a time horizon which determines when the dynamic flow has to reach the target t.

Problem: Maximum Dynamic Flow
Input: Graph $G = (V, E)$, capacities $c : E \to \mathbb{R}_{\geq 0}$, transit times $d : E \to \mathbb{N}_0$, $s, t \in V$, time horizon $T \in \mathbb{N}_0$
Task: Find a feasible s-t flow over time f with time horizon T and maximum value $|f|$.

There are a few important differences between Maximum Dynamic Flow and Minimum Time-Shared Edges. The biggest difference is clearly that in MTSE we can share edges which then have unlimited capacities. Furthermore, in Dynamic flow flow

[2] A series-parallel graph is obtained by a combination of series and parallel compositions starting with a K_2. Refer to the book of Brandstädt et al. [BLS99].
[3] A threshold graph is obtained by a combination of adding an isolated vertex or adding a dominating vertex starting with a single vertex. Refer to the book of Brandstädt et al. [BLS99].
[4] In a chordal graph, every cycle of length four or more has an edge not part of the cycle connecting two vertices of the cycle. Refer to the book of Brandstädt et al. [BLS99].

	Path-MTSE	Trail-MTSE	Walk-MTSE
$k = 0$ Undirected	NP-c (Theorem 2.4)	NP-c (Theorem 2.7)	P (Theorem 2.9)
$k = 0$ Directed	NP-c (Theorem 2.4)	NP-c (Theorem 2.6)	P (Theorem 2.10)
$k = 0$ DAGs	P (Theorem 2.2)	P (Theorem 2.2)	P (Theorem 2.2)
Undirected	NP-c (Theorem 2.4)	NP-c (Theorem 2.7)	P (Theorem 2.9)
Directed	NP-c (Theorem 2.4)	NP-c (Theorem 2.6)	NP-c (Theorem 2.1)
DAGs	NP-c (Theorem 2.1)	NP-c (Theorem 2.1)	NP-c (Theorem 2.1)
Short MTSE	NP-c (Theorem 3.1)	NP-c (Theorem 3.1)	NP-c (Theorem 3.2)

Table 1: Overview on the results of the complexity analysis. NP-c denotes that a problem is NP-complete, P denotes that a problem is contained in P

is allowed to stop and wait at vertices including the source s; it is only important that the flow reaches t within the time horizon. While we introduce in Section 3 the SMTSE problem which provides MTSE also with a time horizon, we strictly forbid the waiting in vertices. Finally, there is no restriction for the dynamic flow to use vertices and edges at most once as we have in Path-MTSE or Trail-MTSE. These differences lead to the different complexity of the two problems, since Maximum Dynamic Flow can be solved in polynomial time using standard flow algorithms on time-expanded graphs [Sku08], while Path-MTSE and Trail-MTSE are NP-complete.

1.4 Overview

In this work, we characterize the complexity of Walk-MTSE, Trail-MTSE, and Path-MTSE.

- In Section 2.1 we prove that MTSE is NP-complete on directed acyclic graphs for all walks, trails, and paths. However, if the number of shared edges is fixed, then we can solve MTSE in polynomial time.
- In Section 2.2 we prove that Path-MTSE is NP-complete on both directed and undirected planar graphs, even if we share no edge.
- In Section 2.3 we prove that Trail-MTSE is also NP-complete on both directed and undirected planar graphs, even if we share no edge.
- In Section 2.4 we provide a polynomial-time algorithm for Walk-MTSE on directed and undirected graphs, if we share no edge. For unrestricted k, we provide a polynomial-time algorithm on undirected graphs.
- In Section 3 we analyze the complexity of a modified version of MTSE (S-MTSE), where the longest s-t connection is restricted by an additional parameter α. We determine that SMTSE is at least as hard as MTSE. While Walk-MTSE was polynomial-time solvable on undirected graphs, SWalk-MTSE is NP-complete on undirected graphs.

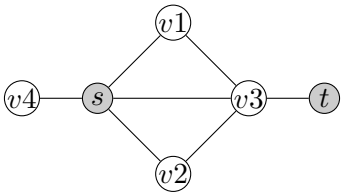

Figure 2: A graph G

Table 1 shows the theorems that, together, give a clear picture of the complexity of the MTSE problem for various graph types and k.

1.5 Basic Observations

In this work, we differentiate between walks, trails, and paths. We show the differences between the types of s-t connections using the example graph showed in Figure 2. For $k = 0$ shared edges, the highest value of p so that the problem is a yes-instance is for Path-MTSE $p = 2$ (P_1, P_2), for Trail-MTSE $p = 3$ (P_1, P_2, P_3), and for Walk-MTSE $p = 4$ (P_1, P_2, P_3, P_4):

$$\begin{aligned} P_1 &= (s, v_3, t), \\ P_2 &= (s, v_1, v_3, t), \\ P_3 &= (s, v_2, v_3, v_1, s, v_3, t), \\ P_4 &= (s, v_4, s\ , v_3, t). \end{aligned}$$

For higher values of p, each of the problems is a no-instance. The example also illustrates the difference to the Minimum Shared Edge problem, since there is only one path for $k = 0$.

Even if the size c of a minimum s-t cut is smaller than p, there could be a solution without a shared edge. For a general graph G, given a minimum s-t cut $C = (S, T)$ of capacity c. Let $V_{\text{cut}} := \{v \in V \mid (v, u) \in E \wedge v \in S \wedge u \in T\}$ be the vertices in S incident to this minimum cut. Let long_v be the longest path from s to a vertex $v \in V_{\text{cut}}$ and short_v be the shortest path from s to a vertex $v \in V_{\text{cut}}$. We can set the upper bound of s-t paths p that share no edge to the sum of the differences between the longest and shortest path to each vertex $v \in V_{\text{cut}}$, $p \leq \sum_{v \in V_{\text{cut}}} |\text{long}_v| - |\text{short}_v|$. However, this upper bound is not only weak, it is also NP-hard to determine the longest path on general graphs.

If the number of shared edges equals the shortest path, there is a solution for every p, where each path is equal to the shortest path. Furthermore, for every yes-instance of MTSE, there is a solution containing a shortest path, since always one path can follow a shortest path, without blocking an edge for any other path than another shortest path.

A lot of tricks used on other graph problems do not work on MTSE, because we added the dimension of time. We can neither subdivide all edges, nor contact an edge to simulate a shared edge. If we subdivide an edge $\{v, u\}$ shared by two s-t connections P_1, P_2, this results either in two shared edges or no shared edge. If both connections

first use v or u and then u or v, then there are two shared edges in the graph after subdivision. If one connection uses first v and the other one first u, then there is no shared edge in the graph after subdivision.

All variants of MTSE presented in this work are contained in NP, since each solution can be verified in polynomial time.

Lemma 1.1. Walk-MTSE, Trail-MTSE , *and* Path-MTSE *are contained in NP both on directed and on undirected graphs.*

Algorithm 1 Verify a given solution $\mathcal{P}$

```
1:  procedure VERIFY(P = {P_1, ..., P_p}, V, E, s, t, p, k)
2:      for i = 1 to i = p do
3:          if P_i[0] ≠ s ∨ P_i[length(P_i)] ≠ t then return False
4:      Shared = ∅
5:      for i = 1 to p do
6:          for l = 0 to length(P) do
7:              e = {P_i[l], P_i[l + 1]}
8:              if e ∉ E then return False
9:              for j = i + 1 to p do
10:                 if length(P_j) ≥ l then
11:                     if e = {P_j[l], P_j[l + 1]} ∧ e ∉ Shared then
12:                         Shared = Shared ∪ {e}
13:                         k = k − 1
14:     if k ≥ 0 then return True
15:     else return False
```

We will prove Lemma 1.1 by providing a polynomial verifier Algorithm 1. For the verifier, we neither assume that a solution contains walks, trails, or paths nor that the graph is directed or undirected. Therefore, the verifier holds for all variants of MTSE.

Proof. Given a solution $\mathcal{P}$ of MTSE, we will verify it in polynomial time as shown in Algorithm 1. Note that with $P[i]$ we access the vertex in P after i steps. If $P = (v_0, v_1, \ldots, v_{l-1}, v_l)$, then $P[i] := v_i$. The verifier handles s-t walks, therefore, for path we have to check beforehand that each vertex appears at most once in each path, for trails we have to check that each edge appears at most once in each trail.

First Algorithm 1 checks, whether each walk is an s-t walk. Next, for each walk P_i, the algorithm iterates step wise through the vertices. For each step l, it first checks whether the edge $e = (P_i[l], P_i[l+1])$ between consecutive vertices is in E. Afterwards, for each walk P_j with $j > i$, it compares if P_j uses the same edge at step i. If the edge was not shared before, then we add e to the set of shared edges and decrease k by one. Note that we compare P_i only with paths P_j for $j > i$, since sharing an edge is a symmetric operation and we compared P_i with all paths P_x for $x < i$ in iterations before. If $k \geq 0$ after all iterations, then the solution shared at most k edges and is therefore a suitable solution for the instance of MTSE. Otherwise, more than k edges

were shared. The algorithm has two loops over p and one loop over the length of the longest walk $l_{\max}$. This results in a running time of $\mathcal{O}(l_{\max}p^2)$. For trails $l_{\max} \leq |E|$, since each edge can appear at most once in a trail. For paths $l_{\max} \leq |V|$, since each vertex can appear at most once. In Lemma 2.11, we will show that the length of the longest walk is upper-bounded by a polynomial in $|E|$ and p. Therefore, we can verify each solution in polynomial time for walks, trails, and paths. Hence, Walk-MTSE, Trail-MTSE, and Path-MTSE are contained in NP. □

2 Complexity of Minimum Time-Shared Edges

In this section we analyze the computational complexity of MTSE for several different variants of input graphs. We first consider directed acyclic graphs, since for DAGs every walk and trail is also a path, since no vertex can be revisited. After that, we analyze the complexity of paths, trails, and walks separately. We consider the general case as well as the case where no edge may be shared.

2.1 Directed acyclic graphs

In directed acyclic graphs, every walk or trail is also a path, since no vertex can be revisited. By definition, DAGs do not contain any directed cycle, hence each vertex can appear at most once in an s-t connection. Therefore, each s-t walk and s-t trail is also an s-t path. In the following proofs, we consider s-t paths.

2.1.1 DAGs with shared arcs

Theorem 2.1. Minimum Time-Shared Edges *on directed acyclic graphs is NP-complete.*

For the proof, we slightly modify the reduction from Set Cover to Minimum Shared Edges given by Omran et al.[OSZ13]. Set Cover is a well-known NP-complete problem defined as follows.

Problem: Set Cover (SC)
Input: A set X, a set of sets $\mathcal{C} \subseteq 2^X$, and $\ell \in \mathbb{N}$.
Question: Are there sets $C_1, \ldots, C_{\ell'} \in \mathcal{C}$ with $\ell' \leq \ell$ such that $X = \bigcup_{i=1}^{\ell'} C_i$?

The basic idea of the reduction from SC to MTSE is to create a vertex for every set in $\mathcal{C}$ and for every element in X. A vertex representing an element $x \in X$ has an edge to a vertex representing a set $C \in \mathcal{C}$ if and only if $x \in C$. Each of these vertices is connected to s and all the vertices representing a set are adjacent to t. In each solution of MTSE on our constructed graph, each vertex corresponding to an element of X appears in an s-t path, which afterwards visits a vertex c_i representing a set C_i in $\mathcal{C}$ and then has to share the arc from c_i to t. To determine a solution of Set Cover, we simply take the sets C_i,

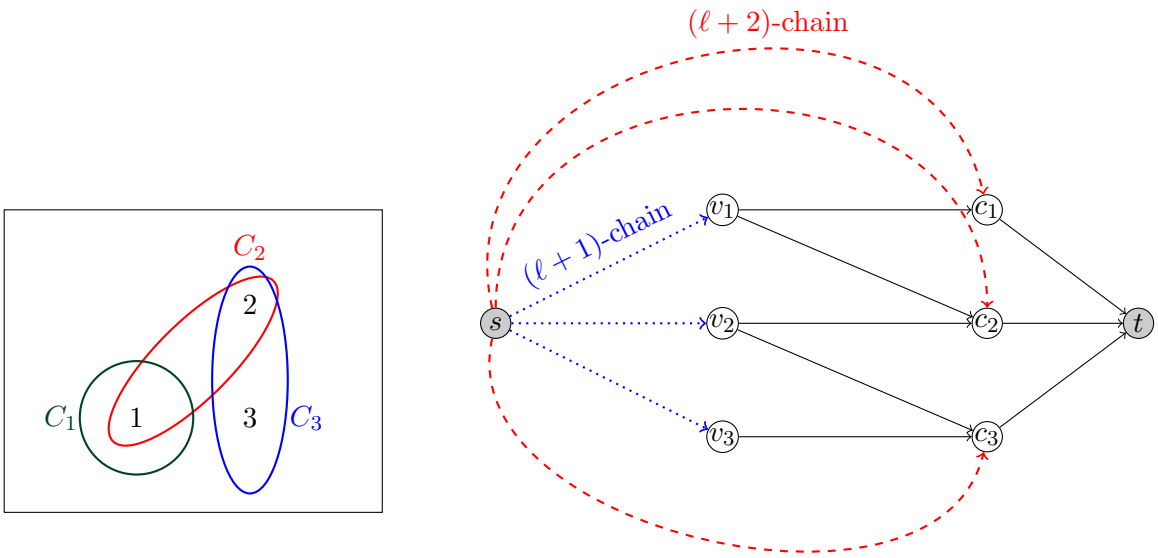

Figure 3: Left an instance of Set Cover with $X = \{1, 2, 3\}$, on the right side a graph of MTSE according our construction. Each red dashed line stands for an $(\ell + 2)$-chain and each blue dotted line stands for an $(\ell + 1)$-chain.

where the corresponding vertices c_i are incident to a shared arc. Our modification to the proof of Omran et al. [OSZ13] is the length of chains connecting s with the vertices, to force all paths arriving at the vertices representing the sets after the same amount of steps.

Proof of Theorem 2.1. To prove that MTSE on DAGs is NP-hard, we reduce Set Cover to MTSE in polynomial time. Let $(X, \mathcal{C}, \ell)$ be an instance of SC. We construct an instance (G, s, t, p, k) of MTSE as follows.

Construction: See Figure 3 for an example of the following construction. Initially, G is the empty graph. For every element $x_i \in X$ we add a vertex v_i to V_X, $V_X = \{v_i \mid x_i \in X\}$, and for every set $C_i \in \mathcal{C}$ we add a vertex c_i to $V_\mathcal{C}$, $V_\mathcal{C} = \{c_i \mid C_i \in \mathcal{C}\}$. At last we add the source s and sink t to our graph. The vertices in G so far are $V_X \cup V_\mathcal{C} \cup \{s, t\}$.
We connect s with each vertex $c_i \in V_\mathcal{C}$ via an $(\ell + 2)$-chain directed from s towards c_i and with each vertex $v_i \in V_X$ via an $(\ell + 1)$-chain directed from s towards v_i. For each vertex $v_i \in V_X$, we add an arc directed towards $c_j \in V_\mathcal{C}$ if and only if the corresponding element $x_i \in X$ in our instance of Set Cover is in the set $C_j \in \mathcal{C}$ corresponding to c_j, that is $\{(v_i, c_j) \mid i \in [|X|] \wedge j \in [|\mathcal{C}|] \wedge x_i \in C_j\}$. Last, we add an arc from every $c_i \in V_C$ directed to t. Note that the constructed graph is a directed acyclic graph. We set the

upper bound k of shared arcs to ℓ and the number p of desired paths to $|X| + |\mathcal{C}|$.

Correctness: We show that, given p s-t paths in G that share at most k arcs, we can construct a set cover $\mathcal{C}' \subseteq \mathcal{C}$ for X with $|\mathcal{C}'| \leq \ell$ and vice versa.
First, we state some basic observations about the graph that we created. The vertex s has $p = |\mathcal{C}| + |X|$ outgoing arcs, each is part of a chain of length at least $(\ell + 1)$, which is exceeding the budget of $k = \ell$ and can therefore not be shared. Therefore, in any solution, every outgoing arc of s must appear in a distinct s-t path. Thus, every vertex $v_i \in V_X$ appears in exactly one path and also every vertex $c_i \in V_\mathcal{C}$ appears in at least one s-t path. Note that all s-t paths arrive at the vertices in $V_\mathcal{C}$ after exactly $\ell + 2$ time-steps. Moreover, all s-t paths are vertex-disjoint up to the vertices in $V_\mathcal{C}$, and it follows that the arcs between vertices in $V_\mathcal{C}$ and t are the only arcs which can be shared in any solution.
Suppose we have p s-t paths in G that share at most k arcs. As stated before, the only arcs that can be shared are arcs between $V_\mathcal{C}$ and t. Let $V' = \{c_i \in V_\mathcal{C} \mid (c_i, t)$ is a shared arc$\}$ be the vertices in $V_\mathcal{C}$ incident to shared arcs. We claim that the set $\mathcal{C}' = \{C_i \in \mathcal{C} \mid c_i \in V'\}$ is a set cover of X. Each vertex $c_i \in V_\mathcal{C}$ has out-degree one, so each ingoing path has to take the arc to t. Since every vertex $c_i \in V_\mathcal{C}$ appears in at least one s-t path using the $(\ell + 2)$-chain from s, for every s-t path using an arc (v_j, c_i) between $v_j \in V_X$ and $c_i \in V_\mathcal{C}$ the arc (c_i, t) must be shared. We say that a set of vertices *covers* a set of paths if each of the paths contains at least one of the vertices. Therefore, the set V' covers all s-t path containing vertices in V_X. Since every vertex in V_X is visited by a distinct s-t path, the set V' covers an s-t path for all the vertices in V_X. By construction of G, the arc (v_i, c_i) is contained in E if and only if $x_i \in C_j$. Since every vertex in V_X is adjacent to a vertex in V', the set $\mathcal{C}'$ covers every element in X. Thus, the set $\mathcal{C}'$ is a set cover of X of size $|V'| \leq \ell$.

Conversely, assume that we have a set cover $\mathcal{C}' \subseteq \mathcal{C}$ of X with $|\mathcal{C}'| \leq \ell$. Let $V' := \{c_i \in V_C \mid C_i \in \mathcal{C}'\}$ be the vertices corresponding to the sets in the set cover, we show that there is a solution of our constructed instance of Path-MTSE where $E_s := \{(c_i, t) \mid c_i \in V'\}$ are the shared arcs. By definition $|E_s| = |\mathcal{C}'| \leq \ell$, thus we need to show that there are $p = |\mathcal{C}| + |X|$ s-t paths in G that do not share any arc beside those in E_s. First we construct $|\mathcal{C}|$ different s-t paths. Each of those s-t paths contains exactly one $(\ell+2)$-chain going from s to a vertex in $V_\mathcal{C}$ and the arc to t. Since there are $|\mathcal{C}|$ different $(\ell+2)$-chains outgoing from s to a distinct vertex in $V_\mathcal{C}$, those s-t paths share no arc.
The remaining $|X|$ s-t paths contain each a different $(\ell + 1)$-chain to a distinct vertex in V_X. Since $\mathcal{C}'$ is a set cover of X, every vertex in V_X is adjacent to at least one vertex in V'. Therefore, those s-t paths in V_X use an arc to a vertex in V' and the thereby shared arc to t. In total, we constructed $p = |\mathcal{C}| + |X|$ s-t paths which share at most $k = \ell$ arcs.

NP-completeness: So far, we proved that MTSE on directed acyclic graphs is NP-hard. MTSE on directed acyclic graphs is also contained in NP as proven in Lemma 1.1. Hence, the problem is NP-complete. □

2.1.2 DAGs without shared arcs

In this section, we investigate the complexity of MTSE on DAGs for the case that no arc is allowed to be shared.

Theorem 2.2. Minimum Time-Shared Edges *on directed acyclic graphs can be solved in* $\mathcal{O}(|V|^3|E|)$ *time if no arc is allowed to be shared.*

Unlike the general case, if we restrict the parameter k to be equal to zero, MTSE is polynomial-time solvable on directed acyclic graphs, as we argue in the following. As mentioned before, there is no difference between a walk, trail, or path in DAGs, so we consider s-t paths in the following algorithm. We make use of the concept of time expansions introduced by Ford and Fulkerson [FF62]. *Time-expanded graphs* are mainly used to reduce dynamic flow problems to static flow problems. In a time-expanded graph, there is a copy of the whole static graph for each discrete time step in the time horizon T. Other than in dynamic flow problems, we have no fixed time horizon T. However, the maximal length of a path from s to t in a DAG is equal to the number of vertices in $G = (V, E)$, so $T = n$ for $n := |V|$. The arcs are connected with the layer according to their transit time, in our case always one.
More precisely, let $G = (V, E)$ be the graph for an instance of MTSE with $k = 0$. We construct a capacitated time-expanded graph $G' = (V', E')$ where we can calculate the p s-t paths of our solution by searching edge-disjoint paths in the time-expanded graph using a maximum-flow algorithm. A sample construction is shown in Figure 4.

Lemma 2.3. *There are p s-t paths without sharing an arc in a directed acyclic graph G, if and only if there is a maximum s_0-t_{T-1} flow of size at least p in the time-expanded graph G'.*

Proof. Construction: For every vertex $v \in V$, there are $T = n$ copies $v_0, \dots, v_{T-1} \in V'$. For each arc $e = (u, v) \in E$, there are $T - 1$ arcs $e_0, \dots, e_{T-2} \in E'$, where $e_i = (u_i, v_{i+1})$ for every $i \in [0, T - 2]$. Those arcs have capacity $c = 1$. Additionally, there are $T - 1$ arcs $(t_0, t_1), \dots, (t_{T-2}, t_{T-1})$ connecting the copies of t with capacity $c = p$. The source of G' is s_0 and the sink is t_{T-1}.

Correctness: We show that there are p s-t paths without sharing an arc in a directed acyclic graph G if and only if there is a maximum s_0-t_{T-1} flow of size at least p in the time-expanded graph G'. Beside the capacities of the arcs between the different vertices t_i, all capacities are equal to one. Therefore, a maximum flow of size p in our constructed time-expanded graph G' is equivalent to p arc-disjoint (except arcs $e \in \{(t_i, t_{i+1}) \mid i \in [0, T-2]\}$) s-t paths in G' [KT05]. We prove that given p arc-disjoint s-t paths in G', we can construct p s-t paths in G which do not share an arc, and vice versa.

Given p arc disjoint s_0-t_{T-1} paths $\mathcal{P}' = \{P'_0, \dots, P'_{p-1}\}$ in G', we construct p s-t paths $\mathcal{P} = \{P_0, \dots, P_{p-1}\}$ in G which do not share an arc. By construction, each arc is directed from layer x to the layer $x + 1$. Since s_0 is in the first layer and t_{T-1} in the Tth layer, each path P' has exactly length T containing exactly one vertex in each layer. If

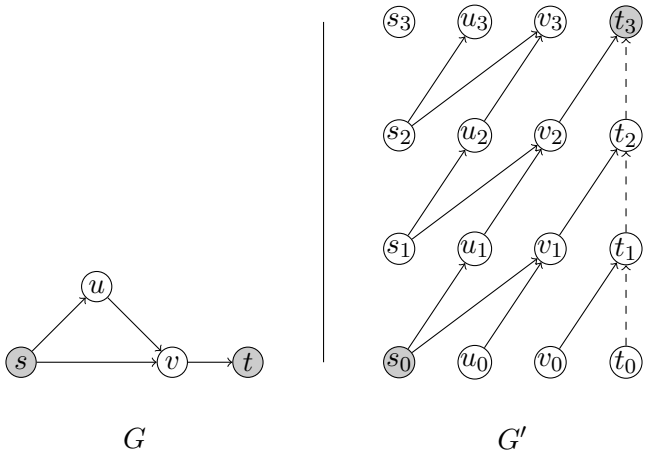

Figure 4: A sample graph G with its time-expanded graph G'. The solid arcs have capacity $c = 1$, the dashed arcs capacity $c = p$

$P'_i = (s_0, x_1, \ldots, y_{l-1}, t_l, t_{l+1}, \ldots, t_{T-1})$ is an s_0-t_{T-1} path in G', then the corresponding s-t path in G is constructed as $P_i = (s, x, \ldots, y, t)$. Since all arcs (x_i, y_{i+1}) in G' have an underlying arc (x, y) in G, the constructed s-t paths $\mathcal{P}$ are valid. Each arc in G' represents an arc in G at a specific step. Since the paths in G' are arc-disjoint, the paths in G share no arc. Therefore, $\mathcal{P}$ contains p s-t paths in G sharing no edge.

Conversely, we can construct p arc-disjoint s_0-t_{T-1} paths in G', given p s-t paths in G sharing no edge. For each s-t path $P_i = (s, v, \ldots, u, t)$ with length ℓ_i, the corresponding s_0-t_{T-1} paths will be $P'_i = (s_0, v_1, \ldots, u_{\ell_i - 1}, t_{\ell_i}, \ldots, t_{T-1})$. Since the paths in G share no arc—which means that they are not using an arc after the same number of steps—the constructed paths in the time-expanded graph G' will be arc-disjoint till t_{ℓ_i}. In G' every s_0-t_{T-1} path has exactly length T. Therefore, each path P'_i contains the vertices $\{t_{\ell_i+1}, t_{\ell_i+2}, \ldots, t_{T-1}\}$ to cover the difference from the sink in G to the sink in G'. Since the arcs (t_j, t_{j+1}) have capacity p, we can construct from the paths $\mathcal{P}'$ a valid s_0-t_{T-1} flow of size p. □

Above we showed that we can construct p s-t paths for each instance $(G, s, t, p, k = 0)$ using a time-expanded graph G'. We use this, to prove the complexity of MTSE on DAGs with $k = 0$.

Proof of Theorem 2.2. Since we can expand a graph G with time expansion in polynomial time to an expanded graph G' and solve the Maximum Flow problem also in polynomial time on G', which is equivalent to solving MTSE for $k = 0$ on G as proven in Lemma 2.3, we can solve MTSE on directed acyclic graphs for $k = 0$ in polynomial time. More precisely, the time-expanded graph G' has n times more vertices and arcs, so $\mathcal{O}(|V'|) = \mathcal{O}(|V|^2)$ and $\mathcal{O}(|E'|) = \mathcal{O}(|E| \cdot |V|)$. We can compute the maximum flow on a static graph in $\mathcal{O}(|V||E|)$ time [Orl13], so we get an overall complexity of $\mathcal{O}(|V|^2 \cdot |V||E|) = \mathcal{O}(|V|^3|E|)$ time.

□

We remark without proof, that given a shared arc $e = (u, v) \in E$, we can include this shared arc in the time-expanded graph G' by setting the capacity of each expanded arc $e' \in \{(u_i, v_i + 1) \mid i \in [0, T - 2]\}$ to $c_{e'} = p$. By this, we find s-t walks sharing specific arcs. Testing each combination of shared arcs results in a brute-force algorithm with running time in $\mathcal{O}((|V|^3|E|) \cdot |V|^k)$. This is an XP algorithm with respect to the number of shared edges k. Therefore, MTSE on DAGs has polynomial running-time for a fixed value of k.

2.2 Paths

In Section 2.1, we showed that Path-MTSE is NP-complete on DAGs with an unbound number k of shared edges, but polynomial-time solvable for $k = 0$ and constant k. In this section, we show that Path-MTSE is NP-complete, both, on planar directed and planar undirected graphs, even for $k = 0$.

Theorem 2.4. Path-Minimum Time-Shared Edges *on planar graphs is NP-complete for every number $k \geq 0$ of shared edges.*

To prove Theorem 2.4 we set the amount of shared edges $k = 0$, so we first prove that the problem with general k is at least as hard as the problem with $k = 0$. Afterwards we proof, that Path-MTSE is NP-complete for $k = 0$ shared edges.

Lemma 2.5. Path-Minimum Time-Shared Edges *and* Trail-Minimum Time-Shared Edges *are at least as hard with $k > 0$ shared edges as with $k = 0$ shared edges.*

Proof. We can reduce every instance of Path-MTSE and Trail-MTSE $(G, s, t, p, k = 0)$ with no shared edge to an instance of Path-MTSE or Trail-MTSE $(G', s', t' = t, p' = p, k')$ with any value for k'. To construct G', we add a new source s' to G, which is connected to s via a k'-chain. Since the degree of the new source s' is one, every path or trail has to use and share the k'-chain to s and will reach s after k' steps with no edge left to share in the subgraph G.

Given p' s-t paths or s-t trails in G' sharing k' edges, they all share the chain between s' and s as mentioned above. Therefore, there is no shared edge in the subgraph G and hence the paths or trails from s are a feasible solution for the instance of MTSE without a shared edge.

Conversely, given p s-t paths or s-t trails in G without sharing an edge, we can clearly construct the p' paths or trails in G' by concatenating the k'-chain with each of the paths or trails. Since the paths or trails share no edge in G, the only shared edges are the edges contained in the k'-chain between s' and s.

Since we did not restrict the graph to be directed or undirected, this proof holds for both, directed and undirected graphs. □

For proving Theorem 2.4 we give a reduction from the NP-complete Hamiltonian Circuit problem on planar, cubic graphs [GJT76], defined as follows:

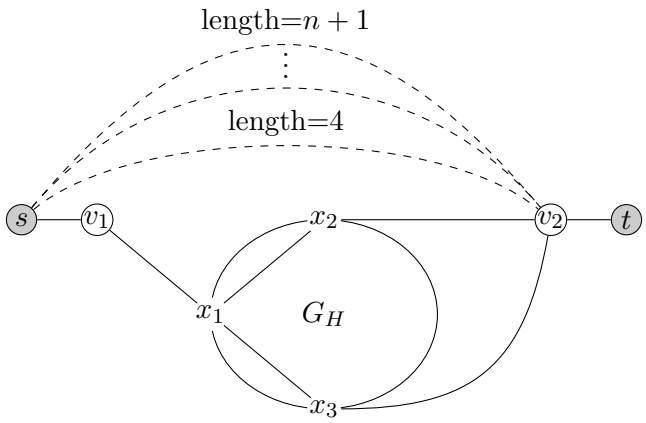

Figure 5: Illustration of the reduction of Hamiltonian Cycle to Path-MTSE. The dashed lines represent chains of length 4 to $n+1$ and the ellipse G_H represents the graph of PHC, where x_1, x_2, x_3 are vertices on the outer face of G_H.

Problem: Planar Hamilton Cycle (PHC)
Input: Graph $G = (V, E)$, where G is undirected, planar, and cubic.
Question: Is there a cycle in G visiting each vertex exactly once?

Proof of Theorem 2.4. To prove that Path-MTSE on undirected planar graphs remains NP-hard for $k = 0$, we reduce every instance of PHC to an instance of Path-MTSE in polynomial time. The basic idea is to force exactly one path to follow a Hamiltonian cycle by blocking the only edge to t with the other paths. In particular, given an instance $G_H = (V_H, E_H)$ of PHC, we construct an instance $(G_M = (V_M, E_M), s, t, p, 0)$ of Path-MTSE as illustrated in Figure 5.

Construction: Let $n := |V_H|$ be the number of vertices in G_H. Note that $n \geq 4$ since G_H is a cubic graph. We set the number of desired paths $p = n - 1$. To construct the planar graph G_M, we copy the graph G_H and add the vertices $\{s, t, v_1, v_2\}$. Next, we connect s with v_1 via a single edge and with v_2 via $n - 2$ chains of length 4 to $n + 1$. We select any vertex $x_1 \in G_H$ and connect it with v_1. Since G_H is planar and cubic, we can select two neighbours x_2, x_3 of x_1 and connect them with v_2 by a single edge each. Last, we connect v_2 with t via a single edge. The constructed graph G_M is clearly planar, since G_H is planar and we provided a planar embedding of the rest graph with x_1, x_2, x_3 on the outer face of G_H.

Correctness: We show that, given p s-t paths in G_M, there is a Hamiltonian cycle in G_H and vice versa.
Suppose we have p s-t paths in G_M that do not share any edge. Since s has degree $n - 1 = p$, every incident edge must be used by exactly one path. The $p - 1$ paths containing the chains arrive at v_2 after 4 to $n + 1$ steps, respectively, and must contain the edge to t because they are not allowed to visit v_2 more than once by our definition of

a path and v_2 is the only adjacent vertex of t. Therefore, the edge $e = \{v_2, t\}$ is blocked between the fifth and $n + 2nd$ step. The remaining path containing v_1 can arrive at v_2 in not less than four steps. Since the edge $e = \{v_2, t\}$ is blocked between the fifth and $(n + 2)$nd step, it must arrive at v_2 after $n + 2$ steps. Hence, the last path must stay in the subgraph G_H for n steps. This is only possible if it visits every vertex at exactly once, since a path can visit every vertex at most once and there are n vertices in G_H. The path enters the subgraph G_H at vertex x_1 and can only leave it at vertex x_2 or x_3, which are adjacent to x_1. Consequently, the last path contains a Hamiltonian path of G_H starting in x_1 and ending in x_2 or x_3 which can be extended to a Hamiltonian cycle of G_H by adding the edge to x_1 at the end. Therefore, this path is a feasible solution for the instance of PHC.

Conversely, suppose that we have a Hamiltonian cycle P_c in G_H. We construct $p = n - 1$ s-t paths in G_M as follows. By construction of G_M, we can construct $n - 2$ s-t paths using the chains between s and v_2. The remaining path is going over v_1 to the subgraph G_H. Here it follows the Hamiltonian cycle P_c starting in x_1. Since every vertex in G_H has degree three, in the Hamiltonian cycle either x_2 or x_3 must be adjacent to x_1. The path follows the Hamiltonian cycle starting at x_1 in the direction such that one of x_2 or x_3 will be the last vertex on the cycle not equal to x_1. Therefore, the path arrives at x_2 or x_3 after $n + 1$ steps and then using the edge to v_2, so that it arrives there after $n + 2$ steps. Since the first $p - 1$ paths have reached t after at most $n + 2$ steps, the remaining path can use the edge to t without sharing it. By this, we provided p paths in G_M sharing not a single edge.

Directed: The proof handles undirected graphs, for the directed graphs we replace every edge $\{a, b\}$ in our construction by two arcs (a, b) and (b, a). Obviously, every path in the undirected graph is also valid for the directed graph. Since no edge is shared, the split of one edge into two arcs does not increase the amount of shared edges. Since there is only one path in the subgraph G_H and the s-t paths using the chains to v_2 are forced to use the edge to t it cannot happen that two paths use the same edge in a different direction. Therefore, this can also not occur in the directed graph, where the edges are split in two arcs. Consequently, the proof is also valid for directed graphs.

Completeness: Since the constructed graph G_M is planar, we proved that Path-MTSE on planar graphs is NP-hard for $k = 0$. Using Lemma 2.5, we conclude that Path-MTSE is NP-hard for any value of parameter k. Path-MTSE on planar graphs is contained in NP, since we showed in Lemma 1.1 that each solution of Path-MTSE can be verified in polynomial time. Hence, the problem is NP-complete. □

2.3 Trails

In Section 2.1, we showed that Trail-MTSE is NP-complete on DAGs with k shared edges, but polynomial-time solvable if the number of shared edges $k = 0$. In this section, we distinguish between directed and undirected trails and show that Trail-MTSE is NP-complete on planar directed graphs and planar undirected graphs, even if the number of shared edges $k = 0$.

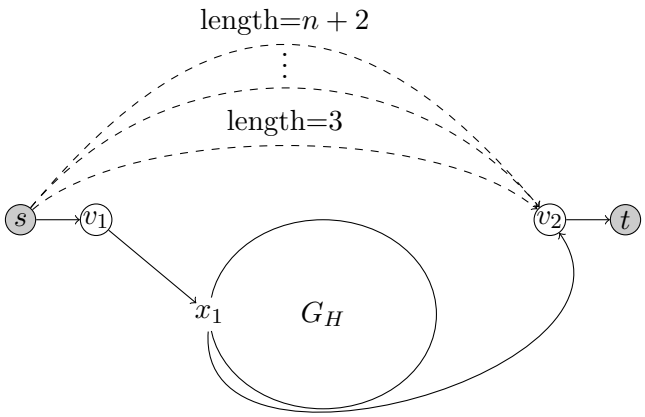

Figure 6: Illustration of the reduction of the Hamiltonian Cycle to Trail-MTSE on directed graphs. The dashed lines represent a chain of length 3 to $n+2$ and the ellipse G_H represents the graph of DPHC.

2.3.1 Directed Trails

We first prove the NP-completeness for Trail-MTSE on directed planar graphs.

Theorem 2.6. Trail-Minimum Time-Shared Edges *on planar directed graphs is NP-complete for every* $k \geq 0$.

In order to prove Theorem 2.6, we adapt the proof for Theorem 2.4 in which we showed that Path-MTSE is NP-complete. Instead of reducing from Hamiltonian Circuit on planar, cubic graphs, we reduce from the NP-complete Hamiltonian Circuit on planar, directed graphs with in-degrees and out-degrees at most two and the sum of them at most three, shortly DPHC [Ple79].

Proof of Theorem 2.6. Let $G_H = (V_H, E_H)$ be an instance of DPHC, we will construct an instance $(G_M = (V_M, E_M), s, t, p, 0)$ of Trail-MTSE as illustrated in Figure 6.

Construction: Let $n := |V_H|$ be the number of vertices in G_H. We set the number p of desired trails to $n+1$. We copy the graph G_H and add the vertices $\{s, t, v_1, v_2\}$, so far the set of vertices of G_M consists of $V_M = V_H \cup \{s, t, v_1, v_2\}$, later on we will add vertices contained in chains. We add an arc each from s to v_1, from v_1 to a vertex x_1 on the outer face of G_H, from x_1 to v_2, and from v_2 to t. Last, we connect s with v_2 via $n-1$ chains of length $3, 4, \ldots, n+2$, respectively, directed towards v_2.

Correctness: The idea is the same as in the proof of Theorem 2.4. At first, suppose we have p s-t trails in G_M sharing no arcs. The out-degree of s equals p, so every outgoing arc must be in one distinct s-t trail. Therefore, there are $p-1$ trails using the chains from s to v_2 and blocking the arc between v_2 and t till step $n+3$. Therefore, the last

trail using v_1 and G_H has to stay in G_H for at least $n+1$ steps. Since every vertex in G_H has at most two incoming and one outgoing or one incoming and two outgoing arcs, every vertex except x_1 can only occur at most once in a trail. Hence, to stay in G_H for at least $n+1$ steps, the remaining trail contains a Hamiltonian cycle starting and ending at x_1.

Conversely, suppose that we have a Hamiltonian cycle in G_H; we need to construct $p = n+1$ s-t trails in G_M that do not share any arc. We construct n different s-t trails using the chains between s and v_2. The remaining trail uses the arc to v_1 and then the arc to the subgraph G_H. Here, the trail follows the Hamiltonian cycle starting and ending in x_1 and afterwards uses the arc to v_2 and arrives there after $n+3$ time steps. Therefore, the arc to t is not blocked and the trail is using it in order to get to t.

NP-completeness: We proved that Trail-MTSE is NP-hard on directed planar graphs and $k = 0$. Using Lemma 2.5, we conclude that Trail-MTSE is NP-hard for any value of parameter k. Trail-MTSE on directed planar graphs is contained in NP as proven in Lemma 1.1. Hence, the problem is NP-complete. □

2.3.2 Undirected Trails

In the section before, we proved that Trail-MTSE on directed planar graphs is NP-complete. We cannot use that proof for undirected graphs since the trails using the chains do not have to use the edge from v_2 to t afterwards. Therefore, it is not ensured that one trail follows a Hamiltonian cycle. In the proof for undirected graphs, we will use an other concept to ensure that the edge between v_2 and t is blocked till one trail follows a Hamiltonian Cycle, to prove that Trail-MTSE is also NP-complete on undirected graphs.

Theorem 2.7. Trail-Minimum Time-Shared Edges *on undirected planar graphs is NP-complete for every* $k \geq 0$.

In the proof of Theorem 2.7, we reduce from the NP-complete Hamiltonian Circuit Problem on planar, cubic graphs [GJT76], shortly PHC (defined in Section 2.2). We first construct a multigraph and discuss the correctness. Subsequently, we doubly subdivide all edges to transform the multigraph into a simple graph. We define a *double subdivision* as a replacement of an edge $e = \{u, v\}$ by two additional vertices p, q and three edges $\{u, p\}, \{p, q\}, \{q, v\}$. We first prove that the double subdivision does not effect the correctness.

Lemma 2.8. *Every instance* $(G, s, t, p, k = 0)$ *of* Minimum Time-Shared Edges *is equivalent to the instance* $(G', s, t, p, k' = 0)$ *of* MTSE*, where* G' *is the graph after double subdividing each edge in* G.

The proof handles s-t trails, but since every path is also a trail, Lemma 2.8 holds also for paths.

Proof of Lemma 2.8. Given p s-t trails in graph G, without sharing an edge, we can construct p s-t trails in the graph G' after double subdivision of each edge, without sharing

an edge. Roughly speaking, for every s-t trail P in the solution for G, we enlarge the trail by including the vertices of the subdivisions to construct a valid s-t trail P' for the solution for G'. More precisely, for every trail $P = (s, \dots, v_i, v_{i+1}, \dots, t)$, where each edge $e = \{v_i, v_{i+1}\}$ is subdivided to $\{v_i, p_i\}, \{p_i, q_i\}, \{q_i, v_{i+1}\}$, the corresponding s-t trail in G' is $P' = (s, \dots, v_i, p_i, q_i, v_{i+1}, \dots, t)$. Clearly, also the new trails connect s and t. Recall that we call an edge shared if there exist at least two s-t trails $P_x = (v_0, \dots, v_a)$, $P_y = (u_0, \dots, u_b)$ in our solution such that there is a step $i \in [\min(\text{length}(P_x), \text{length}(P_y))]$ where both use the same edge: $e = \{v_i, v_{i+1}\} = \{u_i, u_{i+1}\}$. After doubly subdividing all edges, each vertex v after i steps in P will be the vertex after $3 \cdot i$ steps in P'. If there is an edge $e \in \{\{v_{3 \cdot i}, v_{3 \cdot i+1}\}, \{v_{3 \cdot i+1}, v_{3 \cdot i+2}\}, \{v_{3 \cdot i+2}, v_{3 \cdot (i+1)}\}\}$ shared by two trails, P'_x and P'_y, in G', then the edge $e' = \{v_i, v_{i+1}\}$ is shared by P_x and P_y in G. Since there is no shared edge in G, there cannot be a shared edge in G'.

Conversely, given p s-t trails in $G' = (V', E')$, we can construct p s-t trails in $G = (V, E)$, both without sharing an edge. By construction, every third vertex in P' is in G, since in between there are two vertices of the subdivisions. We take the corresponding vertices in G for every third vertex in P' to create P. Precisely, if $P' = (s, v_1, v_2, v_3, v_4, \dots, t)$ the constructed trail for G will be $P = (s, v_3, v_6, v_9, \dots, t)$. There is an edge in G between the consecutive vertices in P since each of the two omitted vertices of P' represents an edge in G. Since the steps are shifted equally in every trail, if there is an edge $e = \{v_i, v_{i+1}\} = \{u_i, u_{i+1}\} \in E$ shared in P, then there is an edge $e' = \{v_{3 \cdot i+1}, v_{3 \cdot i+2}\} = \{u_{3 \cdot i+1}, u_{3 \cdot i+2}\} \in E'$ shared in P', where the edge e was subdivided into $\{v_{3 \cdot i}, v_{3 \cdot i+1}\}, \{v_{3 \cdot i+1}, v_{3 \cdot i+2}\}, \{v_{3 \cdot i+2}, v_{3 \cdot (i+1)}\}$. Since there is no shared edge in G', there is also no shared edge in G. □

Lemma 2.8 is not valid for arbitrary k. To show this, we introduce the *direction* an edge is shared. The direction an edge is shared is defined by the order in which a pair of vertices appears in a trail. Let $\{u, v\}$ be an edge shared by the trails P_1 and P_2. If $P_1 = (s, \dots, u, v, \dots, t)$ and $P_2 = (s, \dots, v, u, \dots, t)$, then the edge $\{u, v\}$ is shared in opposite direction. If $P_1 = (s, \dots, u, v, \dots, t)$ and $P_2 = (s, \dots, u, v, \dots, t)$, then the edge $\{u, v\}$ is shared in the same direction. If an edge is shared by two trails in the same direction, then this would result in three shared edges in the graph after double subdivision. However, if an edge is shared in opposite direction, then this would result in only one shared edge. Therefore, Lemma 2.8 holds only for $k = 0$.

Proof of Theorem 2.7. To prove that Trail-MTSE on undirected planar graphs is NP-hard for $k = 0$, we reduce Planar Hamiltonian Cycle to Trail-MTSE. In particular, given an instance $G_H = (V_H, E_H)$ of PHC, we construct an instance $(G_M = (V_M, E_M), s, t, p, k = 0)$ of Trail-MTSE as follows.

Construction: The construction is illustrated in Figure 7. Let $n := |V_H|$ be the number of vertices in G_H. We set the desired number of s-t trails $p = 2 \cdot n$ and the number of shared edges $k = 0$. We obtain the graph G_M from G_H as follows. First, subdivide each edge in G_M once. Let G'_H denote the obtained graph. To G'_H we add the vertices $\{s, t, v_1, v_2\}$. We select a vertex x_1 on the outer face of G'_H which was also contained in G_H and connect it with s and with v_2 via a single edge. Next, we connect s with v_1

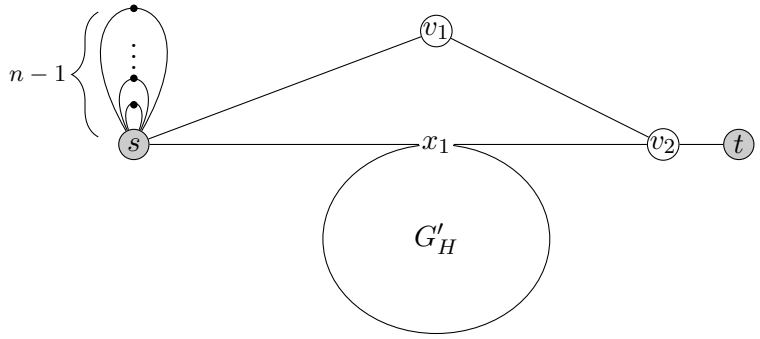

Figure 7: Illustration of the graph constructed in the reduction from Planar Hamiltonian Cycle (PHC) to Trail-MTSE. The ellipse represents the single subdivided graph G'_H of PHC.

and v_1 with v_2 via a single edge. Additionally, we connect v_2 with t via a single edge. Last, we add $n-1$ vertices $\{u_i \mid i \in [n-1]\}$ and connect these with two edges to s each. We refer to them in the following as *bows*. Since G_H is a planar graph and subdivisions keep the planarity, G'_H is also planar. Since we provided a planar embedding in Figure 7, where x_1 is on the outer face of G'_H, the constructed graph G_M is planar.

We note some basic observations about the constructed graph G_M. Since $k = 0$, no edge can be shared. We remark that a vertex of degree three can only appear once in a trail. Since v_2 is the only adjacent vertex of t and has degree three, every s-t trail reaching v_2 needs to use the edge to t afterwards. Moreover, since G_H is a cubic graph, an s-t trail can visit every vertex in G'_H except x_1 at most once. Since we subdivided each edge in G_H to obtain G'_H, each trail in G'_H arrives at a vertex in G_H every two steps. Therefore, each trail can leave G'_H only after an even number of steps in G'_H.

In the following, we state some observations on the vertex s. The incident edges are the $n-1$ bows and the edges to v_1 and x_1. Therefore, at most two trails can leave s each step, the remaining trails have to *stay* in s. By stay we mean that a trail has to use a bow and return to s after two steps. Since using a bow takes two steps, the trails arrive at s only after an even number of steps. Therefore, at most two trails can leave s every two steps. In the longest trail s can appear n times, since there are $n-1$ bows and each trail can use an edge/bow at most once by definition of a trail. Therefore, a trail can contain s between one and n times. Each bow can be used by two trails at the same time, it follows that there can be at most $n \cdot 2$ s-t trails. Since $p = n \cdot 2$ at least two trails have to leave s each even step. Therefore, exactly two trails have to leave s each even step. It is also necessary that these trails can stay in s without sharing an edge. This is possible and easy to show. First we label the bows as $l_1, \ldots, l_{n-1}$. We split the trails in two groups of size $\frac{p}{2} = n$. One group uses always first the "left" edge and then the "right" edge of a bow, the other group the other way around. By this, each bow can be used by two trails, without sharing an edge, since they are always at a vertex u_i after the same amount of steps. In each group, every trail uses a distinct bow except

one trail, which uses the edge to v_1 or x_1. For every second step when the trails arrive at s, the two trails used the bow l_{n-1} leave s via the the edges to v_1 or x_1. The other trails use a bow one rank higher in our order, so if a trail used the bow l_i in the last step, it will use the bow l_{i+1} in the next step. By this no bow is shared. Note that both the edge to v_1 and the edge to x_1 is used by exactly n trails.

Correctness: Given $2 \cdot n$ s-t trails in G_M sharing no edge, we construct a Hamiltonian cycle in G_H as follows. As mentioned before, n trails are using the edge to v_1 and n trails are using the edge to x_1, each in a frequency of two steps. Since v_1 has degree two and v_2 has degree three, the trails using the edge to v_1 have to use the edge to v_2 and then to t. Therefore, the edge (v_2, t) is blocked after all even steps till $2 \cdot n$, that is, after $i \in \{2, 4, 6, \ldots, 2 \cdot n\}$ steps. In every solution for G_M, the other n trails enter the subgraph G'_H in x_1. A trail can leave the subgraph G'_H only after an odd number of steps since each trail arrives there after an odd number of steps and, because of the subdivision of each edge, each trail has to stay in G'_H an even number of steps. Since x_1 is connected to v_2 via a single edge, every trail using the subgraph G'_H arrives at v_2 after an even number of steps. In every even step between two and $2 \cdot n$ a trail from v_1 blocks the edge between v_2 and t, thus, the first trail arriving in x_1 has to be in G'_H for at least $2 \cdot n$ steps. Since it can visit every vertex except x_1 only once, the trail has to follow a Hamiltonian cycle starting and ending in x_1 before leaving the subgraph to v_2 and arriving there after $2 \cdot n + 2$ steps. At this time, the edge is not blocked anymore. We now remove the vertices of the subdivision from this trail to obtain a Hamiltonian cycle in G_H starting and ending in x_1. Since all vertices of the subdivision have exactly degree two, they did not influenced the trail, except extending the length. Therefore, this cycle is a valid solution for PHC.

Conversely, given a Hamiltonian cycle C in G_H, we construct $2 \cdot n$ trails in G_M without sharing an edge as follows. First, we construct n trails leaving consecutively in a frequency of two steps by using the bows over v_1 to v_2 and then to t. As stated before, this is possible without sharing a bow. These trails use the "left" edges of the bows, with the order, that the trail used the bow l_{n-1} leaves to v_1. The remaining trails uses the "right" edges of the bows. The trail used the bow l_{n-1} uses the edge to x_1 afterwards. As mentioned, these trails share no edge in the bows. Each trail arriving in x_1 follows the same Hamiltonian cycle C' including the subdivisions and then uses the edge to v_2. The first trail from the subgraph G'_H arrives at v_2 after $2 \cdot n + 2$ steps, when the edge to t is not blocked anymore. Since the trails in G'_H follow the same Hamiltonian cycle in a distance of two steps, no edge is shared in G'_H. Since no edge is shared, this is a valid solution for our instance of Trail-MTSE.

Subdivision: Last, we have to reduce the multigraph to a simple graph without affecting the correctness. We double subdivide each edge in G_M to convert the double edges in the bows incident to s into cycles of length six. By Lemma 2.8, double subdivision is allowed if we have $k = 0$ no shared edge. Hence, Trail-MTSE on planar simple graphs is NP-hard.

NP-completeness: Above, we proved, that Trail-MTSE is NP-hard on undirected planar graphs even for $k = 0$. In Lemma 1.1 we proved that Trail-MTSE is contained

in NP. Hence Trail-MTSE is also NP-complete. □

2.4 Walks

In this section, we investigate walks. Unlike trails or paths, Walk-Minimum Time-Shared Edges can be solved in polynomial time on undirected graphs. A walk can use each edge an unrestricted number of times, therefore walks can alternate in an undirected graph between two vertices by using the same edge multiple times in a row. This leads to a simple polynomial time algorithm on undirected graphs. Walk-MTSE remains NP-hard for general k on directed acyclic graphs, but is polynomial time solvable on general directed graphs if the numbers of shared edges k is equal to zero.

2.4.1 Undirected Walks

Theorem 2.9. Walk-Minimum Time-Shared Edges *on undirected graphs can be solved in polynomial time.*

In the proof, we distinguish between the case of $k \geq 1$ and $k = 0$, but the idea of the algorithm is the same. The walks are alternating between s and an adjacent vertex, and then use the same path each from s to t consecutively.

Proof. Given an instance (G, s, t, p, k) of Walk-MTSE we want to find p walks from s to t sharing at most k edges. First, we need to find an s-t path, this can be done for example with Breath-First-Search in $\mathcal{O}(|V| + |E|)$ time [KT05]. This gives us a path $P_{\text{short}} = (s, v_1, \ldots, t)$ if there is a connection between s and t. Otherwise, there will be no solution for Walk-MTSE as well.

If $k \geq 1$, then all walks of our solution $P_1, P_2, \ldots, P_p$ share the edge $\{s, v_1\}$. The first walk follows the path P_{short} found in the Breath-First-Search directly. The ith path uses the edge $\{s, v_1\}$ back and forward i times before following P_{short} as well. More precisely, $P_1 = P_{\text{short}}$, $P_2 = (s, v_1, P_{\text{short}})$, $P_3 = (s, v_1, s, v_1, P_{\text{short}})$ and so on. Since all walks follow the same path, but with a time difference of two steps, no edge except $\{s, v_1\}$ is shared.

If $k = 0$, we are not allowed to share any edge. Hence, the solution depends on the degree of s. There is no solution if $\deg(s) < p$. Since all walks start at the same time, not more than $\deg(s)$ walks can leave s without sharing an edge. However, if $\deg(s) \geq p$, then in a simple graph there are at least p neighbors $v_1, \ldots, v_p$ of s, where v_1 is the neighbor in P_{short}. Each walk P_i uses the edge to v_i and alternates $2 \cdot (i-1)$ steps between s and v_i. Then the walk follows P_{short}. For example $P_1 = P_{\text{short}}$, $P_2 = (s, v_2, P_{\text{short}})$, $P_3 = (s, v_3, s, v_3, P_{\text{short}})$, and so on. In this case, no edge is shared, since the walks have each a distinct edge to alternate and then follow the path P_{short} to t with a time difference of two steps. This is illustrated in Figure 8.

□

2.4.2 Directed Walks

On directed acyclic graphs Walk-Minimum Time-Shared Edges is NP-hard as shown in Theorem 2.1. If the number of shared arcs $k = 0$, then Walk-MTSE on directed

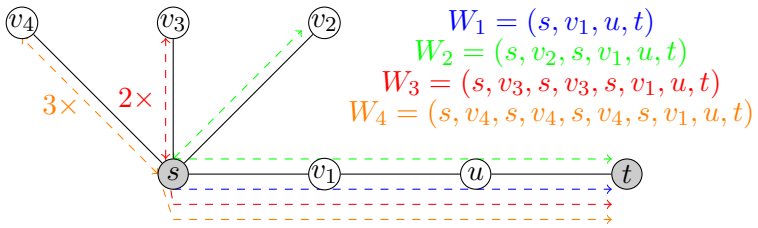

Figure 8: A sample graph G where the maximum amount of s-t walks without sharing an edge is 4. A possible solution is illustrated, where each dashed line represents a walk.

acyclic graphs becomes polynomial-time solvable, as shown in Theorem 2.2. In this section, we prove that Walk-MTSE is polynomial-time solvable also on general directed graphs for $k = 0$ by providing a polynomial-time algorithm using the concept of time-expanded graphs similarly to the proof of Theorem 2.2.

Theorem 2.10. Walk-Minimum Time-Shared Edges *on directed graphs can be solved in polynomial time with the number of shared edges* $k = 0$.

To show this, we prove that if an instance of Walk-MTSE $(G, s, t, p, k = 0)$ is a yes-instance, then there exists a solution that has the length of the longest walk restricted by the length of the longest shortest path to t, called d, and the amount of walks p. The length d of the longest shortest path to t is can be calculated by the depth of a BFS-tree starting in t with all arcs reversed. It is defined as:

$$d := \max_{v \in V; \text{dist}(v,t) < \infty} (\text{dist}(v, t))$$

We later use this length restriction to create a time-expanded graph.

Lemma 2.11. *For every finite yes-instance of* Walk-Minimum Time-Shared Edges *there exists a finite solution where the longest walk is restricted by* $p \cdot d$.

Proof. We prove Lemma 2.11 by showing that, given a solution $\mathcal{P}$ of an instance of Walk-MTSE (G, s, t, p, k) which contains at least one walk P_i with length longer than $d \cdot p$, we can construct a new solution $\mathcal{P}'$ where each walk has length of at most $d \cdot p$. More precisely, Algorithm 2 creates a solution where for each $i \in [p]$ at least i walks have length restricted by $d \cdot i$. Before proving the correctness of Algorithm 2, we explain some notation used, similar to array notation in programming. To get the vertex v after x steps in walk P, we write $P[x] := v$. To access an interval of a walk $P = (v_0, v_1, \ldots, v_\ell)$ between step i and step j, $i < j$, we write $P[i, j] := (v_i, v_{i+1}, \ldots, v_{j-1}, v_j)$. If we replace an interval of a walk $P_x[i, j]$ by another walk P_y, in order to ensure that the result is an s-t walk, $(P_x[i-1], P_y[0]), (P_y[\text{length}(y)], P_x[j+1]) \in E$ for $i \neq 0$, $i < j$ and $j \neq \text{length}(P_x)$ must hold. If $i = 0$ or $j = \text{length}(P_x)$, that is, $P_x[i] = s$ or $P_x[j] = t$, then P_y must start with the source s or end with the sink t respectively. Last, with

Algorithm 2 Reduce a solution to length $d \cdot p$

procedure ReduceLength($G = (V, E), \mathcal{P} = \{P_1, \ldots, P_p\}$)
  Let $\mathcal{P}_1 := \{P_j \in \mathcal{P} \mid |P_j| \leq 1 \cdot d\}$;
  Let $\mathcal{P}_2 := \{P_j \in \mathcal{P} \mid |P_j| > 1 \cdot d\}$;
  for $i = 1$ to $i = p$ **do**
    Let $\mathcal{W}_1 := \{P_j \in \mathcal{P}_1 \cup \mathcal{P}_2 \mid |P_j| \leq i \cdot d\}$;
    Let $\mathcal{W}_2 := \{P_j \in \mathcal{P}_1 \cup \mathcal{P}_2 \mid |P_j| > i \cdot d\}$;
    if $|\mathcal{W}_1| \geq i$ **then**
      $\mathcal{P}_1 = \mathcal{W}_1$
      $\mathcal{P}_2 = \mathcal{W}_2$
    else
      P_{temp} = Select arbitrary $P_j \in \mathcal{W}_2$
      $P_S = \text{shortestpath}(P_{\text{temp}}[(i-1) \cdot d], t)$
      $P_{\text{temp}}[(i-1) \cdot d, \text{length}(P_{\text{temp}})] = P_S$
      $\mathcal{S}$ = new shared arcs
      if $\mathcal{S} = \emptyset$ **then**
        $\mathcal{P}_1 = \mathcal{W}_1 \cup \{P_{\text{temp}}\}$
        $\mathcal{P}_2 = \mathcal{W}_2 \backslash \{P_j\}$
      else
        $(u, v) = (u, v) \in \mathcal{S} \mid \text{dist}(v, t) = \min_{(y,z) \in \mathcal{S}}(\text{dist}(z, t))$
        Select shared arc $(u, v) \in \mathcal{S}$ with least distance to t
        c = steps when P_{temp} uses and shares the arc (u, v).
        P_x = walk that shares arc (u, v) with P_{temp}
        $\mathcal{P}_2 = \mathcal{W}_2 \backslash \{P_x\}$
        $P_x[c+1, \text{length}(P_x)] = P_{\text{temp}}[c+1, \text{length}(P_{\text{temp}})]$
        $\mathcal{P}_1 = \mathcal{W}_1 \cup \{P_x\}$
  return $\mathcal{P}_1$

shortestpath(a, b) we call an algorithm which returns the shortest path from vertex a to vertex b.

Correctness: The algorithm iterates from one to p. For each $i \in [p]$, it checks whether there are already i s-t walks in our solution with length shorter than $i \cdot d$. There are at least $(i-1)$ walks with length upper-bounded by $(i-1) \cdot d$, created in the step before. Therefore, if there are less than i walks shorter than $i \cdot d$, then we need to shorten exactly one walk.

With $\mathcal{W}_2$ we denote the walks with length longer than $i \cdot d$, with $\mathcal{W}_1$ the walks shorter than or equal to $i \cdot n$. Note that if we need to shorten a walk, there is no walk with length between $(i-1) \cdot d$ and $i \cdot d$ in $\mathcal{W}_1$. We first select an arbitrary walk $P_j \in \mathcal{W}_2$. From its position after $(i-1) \cdot d$ steps, $P_j[(i-1) \cdot d]$, we calculate the shortest path to t. This path P_S is clearly shorter or equal d. We concatenate $P_j[0, (i-1) \cdot d]$ with the shortest path P_S to obtain P_{temp}. The length of P_{temp} is less equal $i \cdot d$. Therefore, if this modified walk P_{temp} has no additional shared arc with an other walk, then we add P_{temp} to our solution $\mathcal{P}_1$ and remove the unmodified walk P_j.

However, if there are new shared arcs $\mathcal{S}$, we cannot add P_{temp} to our solution. The new shared arcs $\mathcal{S}$ are shared with walks in $\mathcal{W}_2$, since, if we shorten a walk, then all walks in $\mathcal{W}_1$ have length less than $(i-1) \cdot d$ and, therefore, already arrived at t. We select the arc $e = (u, v) \in \mathcal{S}$ with the least distance to t. We can measure the distance using the shortest path P_S and check for the shared arc with the highest index c in P_{temp}. We then select the walk P_x that shares the arc $e = (u, v)$ with P_{temp}, that is $P_{\text{temp}}[c] = P_x[c] = u$ and $P_{\text{temp}}[c+1] = P_x[c+1] = v$. Since e was the shared arc with the least distance to t, there will be no further shared arcs between v and t, if P_x follows P_S. Therefore, we discard P_{temp} and concatenate $P_x[0, c]$ with the shortest path from u to t $P_{\text{temp}}[c+1, \text{length}(P_{\text{temp}}]$. We add the modified P_x to our solution, since P_x is an s-t walk with length bounded by $i \cdot d$ that does not share an additional arc with any walk.

By this, we create in each iteration i one walk with length upper bounded by $i \cdot d$. Since we iterate till $i = p$, we end up with p s-t walks sharing no additional arc and with length smaller equal $p \cdot d$. Since our algorithm works on every solution for a yes-instance of Walk-MTSE, there is a solution where the length of the longest walk is upper bounded by $p \cdot d$ for every yes-instance of Walk-MTSE. □

We showed that for every finite yes-instance (G, s, t, p, k) of Walk-MTSE, there exists a solution upper-bounded by $p \cdot d$. Therefore, we can use the concept of time-expansion with time horizon $T = p \cdot d$ to solve Walk-MTSE in polynomial time.

Proof of Theorem 2.10. Given an instance $(G = (V, E), s, t, p, k = 0)$ of Walk-MTSE, we will construct a capacitated time-expanded graph $G' = (V', E')$. In the time-expanded graph G' we will use a maximum-flow algorithm to obtain p edge-disjoint paths, which we use to calculate p s-t walks sharing no edge in G.

Construction: The construction and proof of Theorem 2.10 is similar to the proof of Theorem 2.2. Since in Lemma 2.11 we proved that for every yes-instance of Walk-MTSE there is a solution with the length of the longest path upper-bounded by $p \cdot d$,

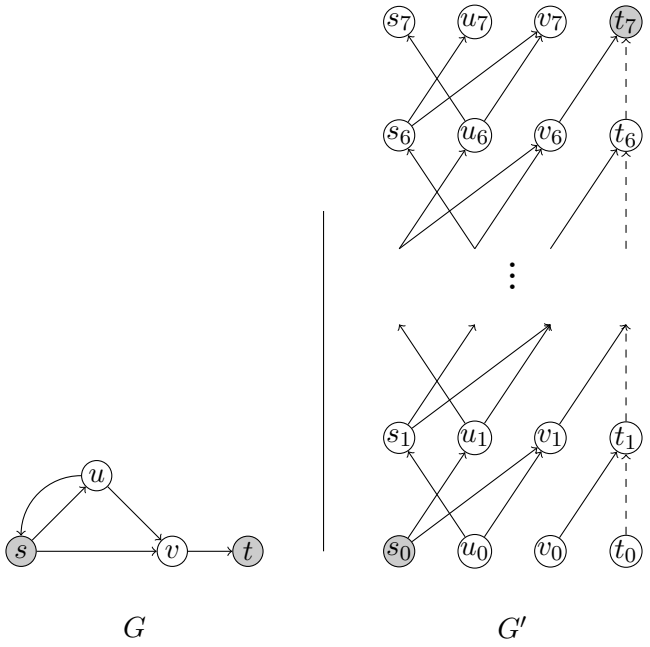

Figure 9: A sample graph G, of an instance $(G, s, t, p = 2, k = 0)$ with its time-expanded graph G'. The longest shortest path to t has length four and, therefore, $T = d \cdot p = 8$. The single arcs have capacity $c = 1$, the dashed arcs capacity $c = p$

we set the time horizon T of the time-expanded graph G' to $T = p \cdot d$. For every vertex $v \in V$, there are T copies $v_0, \ldots, v_{T-1} \in V'$. For each arc $e = (u, v) \in E$, there are $T - 1$ arcs $e_0, \ldots, e_{T-2} \in E'$, where $e_i = (u_i, v_{i+1})$ for every $i \in [0, T - 2]$. Those arcs have capacity $c = 1$. Additionally, there are $T-1$ arcs $(t_0, t_1), \ldots, (t_{T-2}, t_{T-1})$ connecting the copies of t ascending with capacity $c = p$. The source of G' is s_0 and the sink is t_{T-1}.

Correctness: Due to Lemma 2.3, we can obtain p s-t paths in a directed acyclic graph G, given an s_0-t_{T-1} flow of size p in the time-expanded graph G' of G, and vice versa. The proof is only valid if G, and hence G', is a directed acyclic graph. For general directed graphs, if we convert a flow of size p in G' to p s-t paths in G, then it is not guaranteed that these s-t paths visit each vertex and use each edge in G at most once. This contradicts the definition of an s-t path or an s-t trail and, therefore, Lemma 2.3 holds only on DAGs for paths and trails. However, a walk can use each vertex and edge any number of times. Therefore, we can use the method described in Lemma 2.3 to obtain p s-t walks in G. The algorithm runs in $\mathcal{O}(|V'||E'|)$ time. Our time-expanded graph G' has $p \cdot d$ more vertices and edges. Since d is in $\mathcal{O}(|V|)$, in the time-expanded graph we have $\mathcal{O}(|V'|) = \mathcal{O}(p|V|^2)$ vertices and $\mathcal{O}(|E'|) = \mathcal{O}(p|E||V|)$ edges. This results in a running time of $\mathcal{O}(p^2|V|^3|E|)$ for Walk-Minimum Time-Shared Edges on directed graphs. □

We remark without proof that as in DAGs, we can include a shared arc $e = (u, v) \in E$,

in the time-expanded graph, by setting the capacity of each arc $e' \in \{(u_i, v_{i+1} \mid i \in [0, T-2]\}] \subset E'$ to p. By brute-forcing every combination of shared arcs, this results in an algorithm with running time of $\mathcal{O}((p^2|V|^3|E|) \cdot |V|^k)$ which is polynomial for fixed values of k. Therefore, this is an XP algorithm and Walk-MTSE on DAGs problem is in XP with respect to the number of shared edges k.

3 Complexity of Short Minimum Time-Shared Edges

For real-world applications, it is often required to restrict the length of an s-t connection by a maximum length α. To include this into our problem defintion, we introduce a length-restricted version of MTSE named Short Minimum Time-Shared Edges. The problem is defined for s-t paths, trails, and walks, and we use s-t connection as a placeholder for the formal definition, connection $\in$ {path,trail,walk}.

Problem: Short Minimum Time-Shared Edges (S-MTSE)
Input: A graph $G = (V, E)$, $s, t \in V$, $p, \alpha \in \mathbb{N}$, and $k \in \mathbb{N}_0$.
Question: Are there p s-t connections of length at most α in G that share at most k edges?

In this section, we show that SPath-MTSE and STrail-MTSE are NP-complete by proving that they are at least as hard as Path-MTSE and Trail-MTSE , respectively. We also show that unlike Walk-MTSE, which can be solved in polynomial time, SWalk-MTSE is NP-complete on both directed and undirected graphs for general k.

3.1 Paths and Trails

Theorem 3.1. Short Path-Minimum Time-Shared Edges *and* Short Trail-Minimum Time-Shared Edges *on planar directed and undirected graphs are NP-complete.*

Both for paths and trails the number of vertices they can contain is upper bounded. Therefore, we can provide a length restriction α to reduce MTSE to S-MTSE.

Proof of Theorem 3.1. Since by definition a path contains every vertex at most once, the maximal length of an s-t path in Path-MTSE is upper-bounded by the number of vertices $|V|$. Therefore, each instance of Path-MTSE (G, s, t, p, k) is equivalent to the instance $(G, s, t, p, \alpha = |V|, k)$ of SPath-MTSE. Since in Section 2.2 we showed that Path-MTSE is NP-complete on planar directed and undirected graphs, even with the number of shared edges $k = 0$, SPath-MTSE is NP-hard on these graphs. SPath-MTSE is also contained in NP. We can adapt the verifier given in Lemma 1.1 to check whether all s-t paths have length at most α. Hence, SPath-MTSE is also NP-complete.

For trails, we can use the same argumentation as above, with the modification that instead of each vertex in a path, a trail contains each edge at most once. Therefore the length is upper-bounded by the number of edges in the graph $|E|$. We showed in

Section 2.3 that Trail-MTSE is NP-hard, hence STrail-MTSE is NP-complete on planar directed and undirected graphs, even with the number of shared edges $k = 0$. □

3.2 Walks

As shown in Section 2.4, Walk-MTSE is polynomial-time solvable on undirected graphs. However, in this section we show that SWalk-MTSE is NP-complete on undirected graphs.

Theorem 3.2. Short Walk-Minimum Time-Shared Edges *on undirected graphs is NP-complete.*

To prove Theorem 3.2, we will provide a similar reduction from Set Cover as in Theorem 2.1. The basic idea is that the shortest walk has the length of $\alpha = \ell + 3$ and so every walks has to follow one of these shortest walks.

Proof of Theorem 3.2. Construction: The construction is identical to the construction in Theorem 2.1 except that the arcs are undirected edges and in SWalk-MTSE we have the parameter α which we set to $\ell + 3$. The construction is illustrated in Figure 10. Given an instance $(\mathcal{C}, X, \ell)$ of Set Cover, the graph of our constructed instance of SWalk-MTSE contains a source s, a sink t, for every set $C_i \in \mathcal{C}$ of an instance of Set Cover a vertex c_i and for every element $x_j \in X$ a vertex v_j. The vertices c_i are connected with t via a single edge, with s via an $(\ell + 2)$-chain and with every v_j, where $x_j \in C_i$, via a single edge. Finally s is connected with every v_j via an $(\ell + 1)$-chain. The number of shared edges $k = \ell$ and the number of desired paths $p = |\mathcal{C}| + |X|$.

Correctness: We show that, given p s-t walks of length at most $\alpha = \ell + 3$ in G that share at most k edges, we can construct a set cover $\mathcal{C}' \subseteq \mathcal{C}$ for X with $|\mathcal{C}'| \leq \ell$ and vice versa. An important observation about our constructed graph is that the shortest s-t walk has length $\alpha = \ell + 3$ and, therefore, each s-t walk in our solution must have exactly length $\ell + 3$ and does not use any edge more than once. As a consequence the walks follow the same paths as in the proof for Theorem 2.1.

Suppose that we have p s-t walks in G that share at most k edges. As mentioned, each walk is a shortest path and, therefore, the walks reduce the distance to s with every step. In addition, s has degree p and every outgoing edge is connected to a chain of length at least $\ell + 1$. Therefore, each s-t walk of our solution uses a different outgoing edge of s and no edge except the edge between a node c_i and t can be shared. Let $V' := \{c_i \in V \mid (c_i, t) \text{ is a shared edge}\}$ be the vertices incident to a shared edge. We claim that the set $\mathcal{C}' := \{C_i \in \mathcal{C} \mid c_i \in V'\}$ is a set cover of X. Since every vertex $c_i \in V$ is adjacent to t and each walk has to follow a shortest path, each s-t walk arriving at a vertex c_i has to use the edge to t directly. Since every outgoing chain of s is used, in each c_i arrives at least one s-t walk after $\ell + 2$ steps and blocks the edges to t. Each vertex v_j represents an element $j \in X$ and is adjacent to all vertices $\{c_i \mid j \in C_i\}$. Since each vertex v_j is in a distinct s-t walk, one of the edges to a node c_i has to be used and the walk arrives at the vertex c_i after $\ell + 2$ steps and has to use the edge to t. But since each edge to t from a vertex c_i is blocked by the s-t walk coming over the $(\ell + 2)$-chain

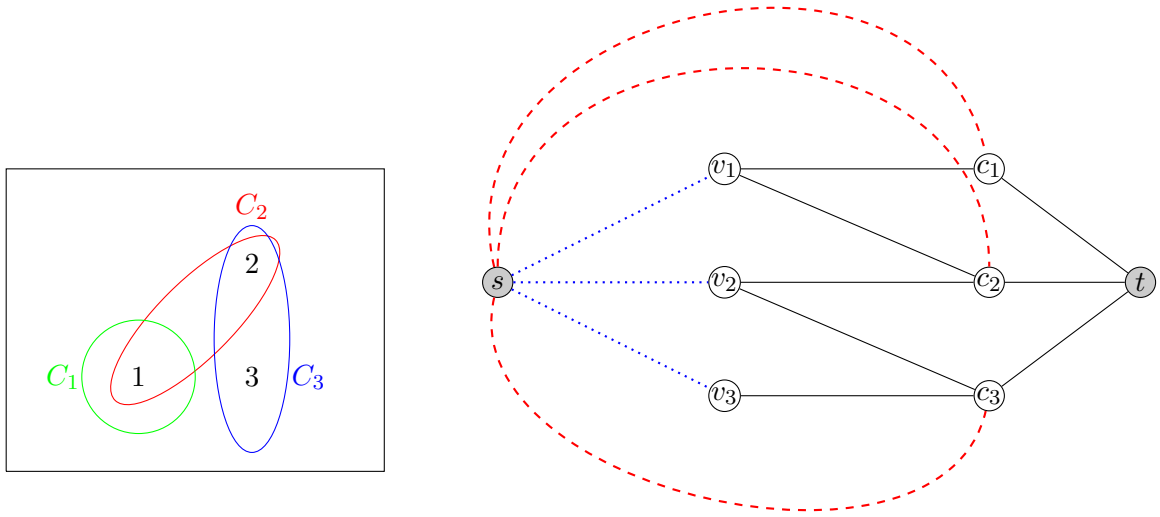

Figure 10: Left an instance of Set Cover, on the right side a graph of SWalk-MTSE according to our construction. Each red dashed line stands for an $(\ell+2)$-chain and each blue dotted line stands for an $(\ell+1)$-chain.

from s, the edge must be shared. Consequently, all s-t walks which visit a vertex v_j contain also a vertex in V'. Since all vertices v_j are in an s-t walk the set $\mathcal{C}'$ is a set cover of X and because the maximal number of shared edges is ℓ, the size $|\mathcal{C}'| \leq \ell$.

Conversely, assume that we have a set cover $\mathcal{C}' \subseteq \mathcal{C}$ with $|\mathcal{C}'| \leq \ell$. Let $V' := \{c_i \in V \mid C_i \in \mathcal{C}'\}$. We construct a solution for our constructed instance of STrail-MTSE with $E_s := \{\{c_i, t\} \mid c_i \in V'\}$ as the shared edges. By definition $|E_s| = |\mathcal{C}'| \leq k = \ell$, thus we need to show that there are $p = |\mathcal{C}| + |X|$ s-t walks of length at most $\alpha = \ell + 3$ in G that share only edges in E_s. First we construct $|\mathcal{C}|$ s-t walks. Each of these walks contains an $(\ell+2)$-chain from s to a vertex c_i and then the edge to t. Clearly, those s-t walks have length $\ell + 3$ and, since there are $|\mathcal{C}|$ outgoing $(\ell+2)$-chains in s, they all use a different chain and a different vertex c_i and so they are not sharing any edge.
The remaining $|X|$ s-t walks are each using a different $(\ell+1)$-chain to a vertex v_j. Since $\mathcal{C}'$ is a set cover of X, every vertex v_j is adjacent to at least one vertex in V'. The s-t walks each uses one of those edges to a vertex $c_i \in V'$ and then the shared edge to t. Clearly, each of these walks has also length $\ell + 3$.

Completeness: So far, we proved that SWalk-MTSE is NP-hard on undirected graphs. To show that SWalk-MTSE is also contained in NP, we can adapt the verifier

for Lemma 1.1 to check whether each walk has a length upper-bounded by α. Hence, SWalk-MTSE is NP-complete. □

4 Conclusion

In this work, we studied the computational complexity of Minimum Time-Shared Edges and ascertained that this problem is NP-complete for most variations. While most routing problems focus only on paths, we also analyzed the complexity of MTSE on trails and walks. Both Trail-MTSE and Path-MTSE are NP-complete on directed and undirected planar graphs. Even though we fixed the number of shared edges to zero, we proved that Trail-MTSE and Path-MTSE on planar graphs remain NP-complete.

On directed acyclic graphs, every walk and every trail is also a path, therefore, the complexity of Walk-MTSE, Trail-MTSE and Path-MTSE on DAGs is the same. We proved that MTSE is NP-complete on DAGs. However, for $k = 0$, we provided an algorithm running in $\mathcal{O}(|V|^3|E|)$ time. We also provided an idea how this algorithm can be extended for small values of k. This could have an usage in the maintenance of wastewater systems, when we keep the number of drains to improve low.

Walk-MTSE on undirected graphs can be solved in polynomial time. However, our algorithm abuses the fact that walks can use an edge an unlimited number of times and each walk follows the same path consecutively. Therefore, the algorithm is not applicable in real-world applications. We introduced Short Minimum Time-Shared Edges to restrict the length of the longest s-t connection. However, SMTSE is at least as hard as MTSE and for walks, SWalk-MTSE on undirected graphs is NP-complete.

Challenges for future research: We proved that Path-MTSE is NP-complete even for $k = 0$. In our proof, we used a high number of paths p and a high maximum degree. It remains an open question whether there is an FPT-algorithm with respect to the number of paths or the maximum degree. Particularly for the security problems, we can assume small values for p and k.

We only consider routing, where edges are shared, but there could be several paths at one vertex at the same time. If we want to count shared vertices as well, on directed graphs, we could replace each vertex v with two vertices v_1, v_2 connected by a single arc directed from v_1 to v_2. Every incoming arc of v is directed to v_1, every outgoing arc starts from v_2. Thus, if more than one path arrives at a vertex at the same time, then the arc between v_1 and v_2 is shared. It remains an open question, how to consider shared vertices on undirected graphs.

It is also interesting whether there is a polynomial-time algorithm on graphs with bounded treewidth. Series-parallel graphs have treewidth of at most two. When we consider Path-MTSE, if s and t are the two terminals on a series-parallel graph, then we could direct each edge of the graph to transform it into a directed acyclic graph, since a path can visit every vertex at most once. Therefore, for fixed k, Path-MTSE on series-parallel graphs is polynomial-time solvable. If there is also an algorithm for general k depending on the treewidth remains an open question for further research.

We could also extend and generalize MTSE by adding a capacity from where on an edge is shared and assign a cost of sharing for each edge. This generalized MTSE could be of special interest in the wastewater maintenance since drains have a different size and, therefore, can hold a different amount of water before they need to be extended.

Literature

[AB09] S. Arora and B. Barak. *Computational Complexity: a Modern Approach*. Cambridge University Press, 2009.

[Aok+14] Y. Aoki, B. V. Halldórsson, M. M. Halldórsson, T. Ito, C. Konrad, and X. Zhou. „The Minimum Vulnerability Problem on Graphs“. In: *Combinatorial Optimization and Applications - 8th International Conference, COCOA 2014, Wailea, Maui, HI, USA, December 19-21, 2014, Proceedings*. Springer, 2014, pp. 299–313.

[Ass+14] S. Assadi, E. Emamjomeh-Zadeh, A. Norouzi-Fard, S. Yazdanbod, and H. Zarrabi-Zadeh. „The minimum vulnerability problem“. In: *Algorithmica* 70.4 (2014), pp. 718–731.

[BLS99] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: A Survey*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999.

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Vol. 4. Springer, 2013.

[Die00] R. Diestel. *Graph Theory*. Vol. 173. Graduate Texts in Mathematics. Springer, 2000.

[FF58] L. R. Ford Jr and D. R. Fulkerson. „Constructing maximal dynamic flows from static flows“. In: *Operations Research* 6.3 (1958), pp. 419–433.

[FF62] L. R. Ford Jr and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[Flu+15] T. Fluschnik, S. Kratsch, R. Niedermeier, and M. Sorge. „The Parameterized Complexity of the Minimum Shared Edges Problem“. In: *35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2015)*. Ed. by P. Harsha and G. Ramalingam. Vol. 45. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 448–462.

[Flu15] T. Fluschnik. „The Parameterized Complexity of Finding Paths with Shared Edges“. Master Thesis (Masterarbeit). Berlin, Germany: TU Berlin, Apr. 2015.

[GJT76] M. R. Garey, D. S. Johnson, and R. E. Tarjan. „The planar Hamiltonian circuit problem is NP-complete“. In: *SIAM Journal on Computing* 5.4 (1976), pp. 704–714.

[Goo16] Google. *Berlin Olympiastadium*. Accessed: 2016-06-03. 2016. URL: %5Curl%7Bhttp://maps.googleapis.com/maps/api/staticmap?size=640x420&style=element:labels%7Cvisibility:off&zoom=15&center=52.5160499,13.2443673%7D.

[KT05] J. Kleinberg and E. Tardos. *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005.

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[Orl13] J. B. Orlin. „Max flows in $O(nm)$ time, or better“. In: *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*. ACM. 2013, pp. 765–774.

[OSZ13] M. T. Omran, J.-R. Sack, and H. Zarrabi-Zadeh. „Finding paths with minimum shared edges“. In: *Journal of Combinatorial Optimization* 26.4 (2013), pp. 709–722.

[Ple79] J. Plesnik. „The NP-completeness of the Hamiltonian cycle problem in planar diagraphs with degree bound two“. In: *Information Processing Letters* 8.4 (1979), pp. 199–201.

[Sku08] M. Skutella. „An Introduction to Network Flows over Time“. In: *Research Trends in Combinatorial Optimization, Bonn Workshop on Combinatorial Optimization, November 3-7, 2008, Bonn, Germany*. 2008, pp. 451–482.

[Tho80] C. Thomassen. „Planarity and duality of finite and infinite graphs“. In: *Journal of Combinatorial Theory, Series B* 29.2 (1980), pp. 244–271.

[Wes+01] D. B. West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, 2001.

[Ye+13] Z.-Q. Ye, Y.-M. Li, H.-Q. Lu, and X. Zhou. „Finding paths with minimum shared edges in graphs with bounded treewidth“. In: *Proceedings of FCS*. 2013, pp. 40–46.