The Complexity of Optimizing Atomic Congestion

Cornelius Brand[1], **Robert Ganian**[2], **Subrahmanyam Kalyanasundaram**[3], **Fionn Mc Inerney**[2]

[1]Algorithms & Complexity Theory Group, Regensburg University, Germany
[2]Algorithms and Complexity Group, TU Wien, Austria
[3]Department of Computer Science and Engineering, IIT Hyderabad, India
cornelius.brand@ur.de, rganian@gmail.com, subruk@cse.iith.ac.in, fmcinern@gmail.com

Abstract

Atomic congestion games are a classic topic in network design, routing, and algorithmic game theory, and are capable of modeling congestion and flow optimization tasks in various application areas. While both the price of anarchy for such games as well as the computational complexity of computing their Nash equilibria are by now well-understood, the computational complexity of computing a *system-optimal* set of strategies—that is, a centrally planned routing that minimizes the average cost of agents—is severely understudied in the literature. We close this gap by identifying the exact boundaries of tractability for the problem through the lens of the parameterized complexity paradigm. After showing that the problem remains highly intractable even on extremely simple networks, we obtain a set of results which demonstrate that the structural parameters which control the computational (in)tractability of the problem are not vertex-separator based in nature (such as, e.g., treewidth), but rather based on edge separators. We conclude by extending our analysis towards the (even more challenging) min-max variant of the problem.

1 Introduction

Congestion games are a by-now classic and widely studied model of network resource sharing. Introduced by Rosenthal (**?**), congestion games and their innumerable variants and extensions have been the focus of a vast body of literature, spanning fields from algorithmic game theory (**?**) over routing (**?**) and network design (**?**), to diverse contexts within artificial intelligence (**????**), both applied and theoretical.

The basic setup of congestion games comprises a network (modeled as a directed graph) and a set of agents that each have an origin and a destination.[1] The agents need to decide which routes to take in order to reach their destination in a way that minimizes the cost of their route, where the cost can capture, e.g., the amount of time or resources required. The eponymous *congestion* enters the scene as follows: the cost accrued by a single agent when traversing a link in the network depends on the number of agents using that link, as described by the link's *latency function*. In essence, the latency function captures how the cost of using each link changes depending on the number of agents using it; depending on the context, more agents using a link could lead to each of them paying a greater cost (e.g., when dealing with traffic congestion) or a lower cost (e.g., when dealing with logistical supply chains), up to a maximum capacity for that link.

It is well-known that selfish strategies in congestion games may not lead to optimal outcomes for all agents, let alone a *system-optimal* outcome[2] (that is, one achieving the minimum average cost) (**?**). In fact, the existence of Nash equilibria for these games was the seminal question investigated by Rosenthal (**?**), and is still of interest in economics and game theory today. Notably, the *price of anarchy* for these games has by now been determined (**??**). The price of anarchy in this context is defined as the ratio

$$\sup_S \frac{\mathrm{cost}(S)}{\mathrm{cost}(S_{\mathrm{sys}})}, \qquad \text{(PoA)}$$

where the supremum ranges over all Nash equilibria, $\mathrm{cost}(S)$ is the cost of a set of strategies S for the agents, and S_{sys} is a *system optimum*, minimizing the average cost over all agents. In addition, the computational complexity of computing Nash equilibria is equally well-studied (**??**); see also the many recent works on the problem (**??**).

While the price of anarchy defined in (PoA) as well as the cost of Nash equilibria $\mathrm{cost}(S)$ (i.e., the numerator in (PoA)) have been extensively treated in the literature on congestion games, it may come as a surprise that, to the best of our knowledge, almost nothing is known about the computational complexity of computing the *denominator* $\mathrm{cost}(S_{\mathrm{sys}})$ of (PoA), that is, determining a *system-optimal set of strategies* for the players. Far from just an intellectual curiosity, applications, e.g., in road or internet traffic routing and planning, make this a pressing question, especially given the rapid developments in autonomous driving systems and the widely adopted political strategy of emphasizing public transportation for its lower environmental impact, which is usually centrally planned and routed, as opposed to individual transport (**?????**).

One possible reason for this gap may lie in the fact that even the most restricted instances of centrally routing a set of agents across a network in a socially optimal manner become hopelessly hard from the perspective of classical computational complexity theory. To illustrate the severity of this

[1]The setting where there is an infinite number of agents and single agents are infinitesimally small is called *non-atomic*; we focus on the classic, *atomic* case, where agents are individual entitites.

[2]In the literature, such outcomes are sometimes called the (*social* or *collective*) *optimum*.

phenomenon, several of these basic classes of intractable instances are described in Section 3. Another explanation for this blind spot in the literature can possibly be found in the fact that, as far as the price of anarchy is concerned, the network structure itself does not appear to play a significant role, whereas this changes drastically when it comes to actually computing a system-optimal set of strategies.

A particular approach that has proven immensely useful for computationally intractable problems lies in employing the rich toolset offered by *parameterized complexity theory* (**??**) in order to obtain a rigorous, more fine-grained and detailed description of the computational complexity of a problem. The central aim of such an endeavour is to identify the structural properties of the input—captured via numerical *parameters*—which give rise to fixed-parameter algorithms for the problem (see Section 2). Some examples where the parameterized complexity toolset has been successfully applied in the context of Artificial Intelligence research include the series of works on Hedonic Games (**???**), Integer Programming (**????**), Data Completion (**????**), and Bayesian Network Learning (**????**).

Our Contributions. As mentioned above, the problem of computing system-optimal strategies in atomic congestion games (SOAC) is extremely hard in terms of classical complexity theory. The core approach of parameterized complexity analysis is to identify those structural properties that a problem instance should have that, even though exceedingly hard in the general case, give rise to its fixed-parameter tractability. Since the network is modeled as a graph, a natural first choice would be to parameterize by the well-established *treewidth* (of the underlying undirected graph) (**?**). Unfortunately, as our first result, we show that the problem remains NP-hard not only on networks which have treewidth 2, but even on networks consisting of a star plus an additional vertex (Theorem 1). This result rules out not only the use of treewidth, but also of virtually all other reasonable graph measures, as a single parameter to solve the problem in full generality.

The above lower bound essentially means that, in order to achieve progress, one needs to combine structural parameters with some auxiliary parameterization of the problem instances. In the context of congestion games, it would seem tempting to consider the number of agents as such an auxiliary parameter, however that would severely restrict any obtained algorithms: while one could reasonably expect that networks of interest may be well-structured, the number of agents in relevant instances of congestion games is typically large, and hence, does not constitute a well-applicable parameter. Instead, here we consider the maximum capacity $\mathtt{c}_{\max}$ of a link in the network as an auxiliary parameter—a value which is never larger, but could be much lower, than the total number of agents in the whole network.

It is important to note that SOAC remains extremely challenging even when parameterized by $\mathtt{c}_{\max}$. In fact, even if $\mathtt{c}_{\max}$ is fixed to a small constant, the problem is NP-hard when restricted to networks of constant treewidth (Theorem 2); the same reduction also rules out the use of other network parameters based on decompositions along *small vertex separators* (such as *treedepth* (**?**)). However, as we show in our main algorithmic result (Theorem 5), SOAC is fixed-parameter tractable when parameterized by $\mathtt{c}_{\max}$ plus a suitable measure that guarantees the network's decomposability along *small edge cuts*.

Basic examples of such measures include the treewidth plus the maximum degree of the network (**??**), or the *feedback edge number* (i.e., the edge deletion distance to acyclic networks) (**??**). In our contribution, we build on the recently introduced *spanning tree* decompositions (**??**) to provide a significantly more general result—in particular, we develop a highly non-trivial dynamic programming algorithm to establish fixed-parameter tractability with respect to the recently introduced slim variant of *treecut width*. We complement Theorem 5 with lower bounds (Theorems 3, 4, and 6) that show the result to be an essentially tight delimitation of the exact boundaries of tractability for SOAC.

In the final section of this article, we focus on a more general variant of SOAC, where instead of requiring all agents to be routed to their destinations, we ask to minimize the system optimum while routing as many agents as possible (or, equivalently, when at most α agents may be left unrouted). This problem, motivated in part by similar lines of investigation conducted for, e.g., multi-agent path finding (**?**) and vehicle routing (**??**), can be seen as a "min-max" variant of SOAC, and hence, we denote it MSOAC. Crucially, MSOAC is even more challenging than SOAC: it remains NP-hard on bidirected trees even for $\mathtt{c}_{\max} = 1$ (**?**), and, perhaps even more surprisingly, is left open on very simple network structures, such as bounded-capacity star networks, when parameterized by α. Be that as it may, as our final result, we show that on bounded-degree networks, the spanning-tree based algorithm obtained for SOAC can be lifted to also solve MSOAC via a fixed-parameter algorithm when parameterized by the treewidth of the network, along with α and $\mathtt{c}_{\max}$.

A mind-map of our results is provided in Figure 1.

2 Preliminaries

For a positive integer $i \in \mathbb{N}$, we let $[i] = \{1, 2, \ldots, i\}$. We refer to the book by Diestel (**?**) for standard graph terminology. While the networks are modeled as directed graphs (i.e., digraphs), many of our results use the skeletons (i.e., the underlying undirected graphs) of these digraphs. The *skeleton* $\underline{G}$ of a directed graph G is the simple undirected graph obtained by replacing each arc in G by an undirected edge. The graph class $K_{i,N} = \{K_{i,j} \mid j \in \mathbb{N}\}$ is the class of all complete bipartite graphs where one side has size i.

Formal Problem Definition. Given a digraph $G = (V, E)$, let $\mathcal{P}$ be the set of all directed paths in G. Given a set $A = \{a_1, \ldots, a_m\}$ of agents where each agent a_i is associated with a tuple $(s_i, t_i) \in V^2$, a *flow assignment* F is a mapping from A to $\mathcal{P}$ such that each agent a_i is mapped to a directed path from s_i to t_i. For an arc $e \in E$, let $f_F(e) = |\{a \mid a \in A \wedge e \in F(a)\}|$ be the number of agents whose flow passes through e; when F is clear from the context, we omit it and simply use $f(e)$ instead.

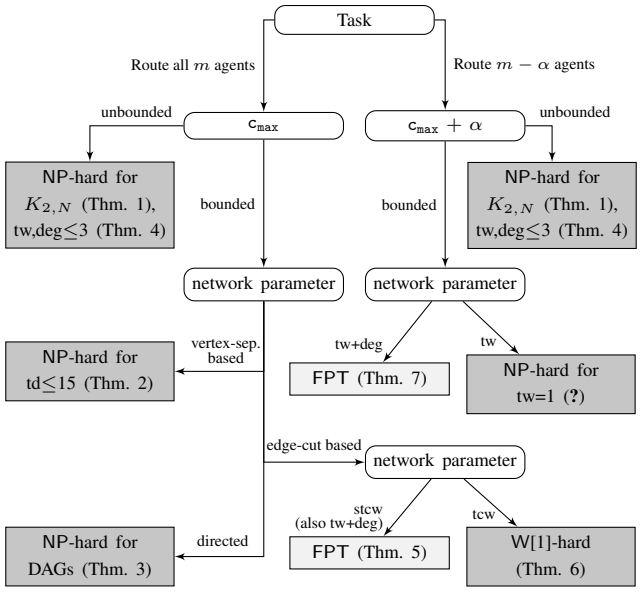

Figure 1: A mind map of our results on computing system-optimal strategies in congestion games. The formal problem definition as well as a discussion of the considered parameters is provided in Section 2; here, tw stands for treewidth, deg stands for maximum degree, td stands for *treedepth*, (s)tcw stands for (slim) treecut width, and DAGs stands for directed acyclic graphs.

Intuitively, our primary problem of interest asks to compute a flow assignment of all the agents that minimizes the total cost. However, in order to formalize the algorithmic lower bounds obtained for the problem, we follow the standard practice of formulating the problem as an equivalent decision problem (see below). To avoid any doubts, we remark that all our algorithmic results are constructive and can also immediately output the minimum cost of a flow assignment with the required properties.

System Optimum Atomic Congestion (SOAC)
Input: A digraph $G = (V, E)$, a positive integer λ, a set $A = \{a_1, \ldots, a_m\}$ of agents where each agent a_i is associated with a tuple $(s_i, t_i) \in V^2$, and for each arc $e \in E$, a latency function $\ell_e : [m] \to \mathbb{R}_{\geq 0} \cup \{\infty\}$.
Question: Does there exist a flow assignment F such that $\sum_{e \in E} f_F(e) \cdot \ell_e(f_F(e)) \leq \lambda$?

In specific settings studied in the literature, the latency function is sometimes required to satisfy certain additional conditions, such as being non-decreasing. As illustrative examples, observe that when agents represent individual vehicles in a traffic network, the latency function will typically be increasing (the cost of 100 agents using a single link is greater than 100 times the cost of that link when it is used by a single agent), but if agents represent individual parcels or shipments in a logistics network, one would expect it to be decreasing (the cost of 100 agents using a single link would be lower than 100 times the cost of that link when it is used by a single agent). In order to capture as wide a range of scenarios as possible—including, e.g., buffered or batch-wise processing at network nodes leading to decreasing or even oscillating latencies, respectively—our study targets the problem with arbitrary latency functions.

Given a latency function $\ell_e : [m] \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ for an arc e, let the capacity c_e of e be defined as the maximum admissible value of the latency function, i.e., $\max\{z \mid \ell_e(z) \neq \infty\}$. Let the *maximum capacity* of a network be defined as $\mathsf{c}_{\mathtt{max}} := \max_{e \in E} c(e)$. Crucially, while the maximum capacity can never exceed the number m of agents in the network, one can reasonably expect it to be much smaller than m in more complex networks.

Parameterized Complexity Theory. In parameterized algorithmics (**???**), the running-time of an algorithm is studied with respect to a parameter $k \in \mathbb{N}$ and input size n. The basic idea is to find a parameter that describes the structure of the instance such that the combinatorial explosion can be confined to this parameter. In this respect, the most favorable complexity class is FPT (*fixed-parameter tractable*), which contains all problems that can be decided by an algorithm running in time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a computable function. Algorithms with this running-time are called *fixed-parameter algorithms*. A basic way of excluding fixed-parameter tractability for a parameterized problem is to show that it remains NP-hard even when the parameter k is bounded by a constant. However, this is not always possible: some problems can be solved in non-uniform polynomial time for each fixed value of the parameter. In these cases, establishing hardness for the complexity class W[1] via a *parameterized reduction* rules out the existence of a fixed-parameter algorithm under the well-established assumption that W[1] $\neq$ FPT. A parameterized reduction can be thought of as a classical polynomial-time reduction, but with the distinction that it (1) can run in fixed-parameter time instead of polynomial time, and (2) must bound the parameter of the output instance by a function of the parameter of the initial instance.

A natural and often used method of parameterizing problems is to consider parameters tied to the structural properties of the instance. In many cases, it turns out that achieving fixed-parameter tractability is only possible if one parameterizes by multiple properties of the instances simultaneously; formally, this is simply reflected by setting the parameter to be the sum of both values. As a basic example, the classical INDEPENDENT SET problem on graphs is not fixed-parameter tractable w.r.t. the size ℓ of the sought set nor the maximum degree d of the graph, but is well-known to admit a fixed-parameter algorithm w.r.t. $\ell + d$ (**?**).

Structural Parameters. In this work, we identify the exact boundaries of tractability for SOAC in the context of fundamental graph parameters, as depicted in Figure 2. For two of these parameters—notably treewidth (**?**) and treedepth (**?**)—we do not need to provide explicit definitions since the respective lower bounds we obtain for SOAC construct instances whose skeletons are well-known to have bounded treewidth and treedepth. The *feedback edge number* (**??**) is simply the minimum number of edges that need to be removed from an undirected graph in order to obtain

a forest. The term "directed graph parameters" here broadly refers to all parameters that achieve constant values on directed acyclic graphs; this includes directed treewidth (**?**), Kelly-width (**?**), and DAG-width (**?**), to name a few.

The proofs of our main results for SOAC (Theorem 5) and MSOAC (Theorem 7), as well as the lower bound in Theorem 6, will require a more in-depth introduction of treecut width and its slim variant. A *treecut decomposition* of G is a pair $(T, \mathcal{X})$ which consists of a rooted tree T and a near-partition $\mathcal{X} = \{X_t \subseteq V(G) : t \in V(T)\}$ of $V(G)$, where a near-partition is a partitioning of a set which can also contain the empty set. A set in the family $\mathcal{X}$ is called a *bag* of the treecut decomposition.

For any node t of T other than the root r, let $e(t) = ut$ be the unique edge incident to t on the path to r. Let T^u and T^t be the two connected components in $T - e(t)$ which contain u and t, respectively. Note that $(\bigcup_{q \in T^u} X_q, \bigcup_{q \in T^t} X_q)$ is a near-partition of $V(G)$, and we use $\mathbf{cut}(t)$ to denote the set of edges with one endpoint in each part. We define the *adhesion* of t ($\mathbf{adh}_T(t)$ or $\mathbf{adh}(t)$ in brief) as $|\mathbf{cut}(t)|$; if t is the root, we set $\mathbf{adh}_T(t) = 0$ and $\mathbf{cut}(t) = \emptyset$.

The *torso* of a treecut decomposition $(T, \mathcal{X})$ at a node t, written as H_t, is the graph obtained from G as follows. If T consists of a single node t, then the torso of $(T, \mathcal{X})$ at t is G. Otherwise, let $T_1, \ldots, T_\ell$ be the connected components of $T - t$. For each $i = 1, \ldots, \ell$, the vertex set $Z_i \subseteq V(G)$ is defined as the set $\bigcup_{b \in V(T_i)} X_b$. The torso H_t at t is obtained from G by *consolidating* each vertex set Z_i into a single vertex z_i (this is also called *shrinking* in the literature). Here, the operation of consolidating a vertex set Z into z is to substitute Z by z in G, and for each edge e between Z and $v \in V(G) \setminus Z$, adding an edge zv in the new graph. We note that this may create parallel edges.

The operation of *suppressing* a vertex v of degree at most 2 consists of deleting v, and when the degree is two, adding an edge between the neighbors of v. Given a connected graph G and $X \subseteq V(G)$, let the *3-center* of (G, X) be the unique graph obtained from G by exhaustively suppressing vertices in $V(G) \setminus X$ of degree at most two. Finally, for a node t of T, we denote by $\tilde{H}_t$ the 3-center of (H_t, X_t), where H_t is the torso of $(T, \mathcal{X})$ at t. Let the *torso-size* $\mathbf{tor}(t)$ denote $|\tilde{H}_t|$.

Definition 1. *The width of a treecut decomposition $(T, \mathcal{X})$ of G is defined as $\max_{t \in V(T)}\{\mathbf{adh}(t), \mathbf{tor}(t)\}$. The treecut width of G, or $\mathbf{tcw}(G)$ in short, is the minimum width of $(T, \mathcal{X})$ over all treecut decompositions $(T, \mathcal{X})$ of G.*

The graph parameter *slim treecut width* can be defined analogously to treecut width, but with the distinction that suppressing only occurs for vertices of degree at most 1. Yet, for our algorithmic applications, it will be more useful to use a different characterization of the parameter—one based on spanning trees and much better suited to the design of dynamic programming algorithms.

For a graph G and a tree T over $V(G)$, let the *local feedback edge number* at $v \in V(G)$ be $E^{G,T}_{\mathrm{loc}}(v) = \{uw \in E(G) \setminus E(T) \mid \text{the path between } u \text{ and } w \text{ in } T \text{ contains } v\}$. The *edge-cut width* of the pair (G, T) is $\mathbf{ecw}(G, T) = 1 + \max_{v \in V} |E^{G,T}_{\mathrm{loc}}(v)|$.

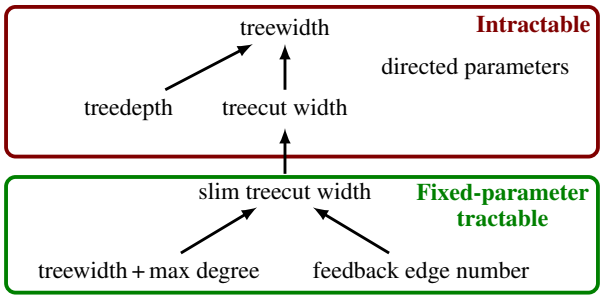

Figure 2: A pictorial view of the tractability of SOAC when capacities are bounded. An arc from a parameter x to a parameter y indicates that x is dominated by y, i.e., a bound on x implies a bound on y but the opposite does not hold.

Lemma 1 (Prop. 27 and Thm. 30, **? ?**)**.** *Every graph G with slim treecut width k admits a spanning tree T over some supergraph G' of G such that (G', T) has edge-cut width at most $3(k+1)^2$. Moreover, such a pair (G', T) can be computed in time $2^{k^{\mathcal{O}(1)}} \cdot |V(G)|^4$.*

Lemma 2 (Prop. 22, Prop. 26, and Thm. 30, **? ?**)**.** *Every graph G with maximum degree d and treewidth w admits a spanning tree T over some supergraph G' of G such that both the edge-cut width of (G', T) and the maximum degree of T are upper-bounded by $\mathcal{O}(d^2w^2)$. Moreover, such a pair (G', T) can be computed in time $2^{(dw)^{\mathcal{O}(1)}} \cdot |V(G)|^4$.*

3 The Surprising Difficulty of Solving Atomic Congestion Games

Before initializing our more fine-grained parameterized analysis of SOAC, we first consider the computational complexity of the problem from the classical point of view, that is, with respect to NP-hardness. In fact, we show that even surprisingly simple input instances turn out to be intractable.

To begin, we show that SOAC is NP-hard even when the underlying undirected graph of the input digraph is restricted to the class $K_{2,N}$. We prove this result (as well as Theorem 4 later on) via a reduction from the following variant of the classical SUBSET SUM problem over d-dimensional vectors:

> Multidimensional Uniform 0/1 Knapsack (MUKS)
> **Input:** A set $S = \{\vec{v}_1, \ldots, \vec{v}_n\} \subseteq \mathbb{N}^d$ of d-dimensional vectors containing natural numbers, a positive integer k, and a target vector $\vec{T} \in \mathbb{N}^d$.
> **Question:** Is there a subset $S' \subseteq S$ of at least k vectors such that $\sum_{\vec{s} \in S'} \vec{s} \leq \vec{T}$ holds?

Lemma 3. MUKS *is* NP*-hard and also* W[1]*-hard parameterized by d, even if all numbers are encoded in unary.*

Proof. The reduction proceeds from a problem called MULTIDIMENSIONAL RELAXED SUBSET SUM (MRSS), which is NP-hard even if the inputs are encoded in unary (**?**). In its definition, it subtly differs from MUKS above. Namely, the direction of the inequalities are swapped, that is, one seeks a

subset of *at most* k vectors that sums up to *at least* $\vec{T}$. Now, given an instance $(S, \vec{T}, k)$ of MRSS, let $\vec{\Sigma} := \sum_{s \in S} s$ be the total sum over all vectors. The instance of MUKS then consists of $(S, \vec{\Sigma} - \vec{T}, n - k)$. We claim that the MUKS instance has a solution of size at least $n - k$ if and only if the original MRSS-instance has a solution of size at most k. Indeed, for $S' \subseteq S$, $\sum_{s \in S'} s \geq \vec{T}$ is equivalent to $\vec{\Sigma} - \sum_{s \in S'} s \leq \vec{\Sigma} - \vec{T}$, and, by definition, the left-hand side equals $\sum_{s \in S''} s$, where $S'' = S - S'$ is a set of at least $n - k$ vectors. This proves the correctness of the reduction, and hence, the NP-hardness on unary inputs.

As for the W[1]-hardness of the problem, we note that MRSS is also known to be W[1]-hard when parameterized by d (**?**). Our polynomial-time reduction does not change the value of d, and hence, it immediately also establishes the W[1]-hardness of MUKS w.r.t. d. □

Before proceeding to the reduction, let us briefly remark on the significance of the result in the context of this article. Essentially, establishing NP-hardness on $K_{2,N}$ rules out not only fixed-parameter tractability under almost all commonly considered graph parameterizations (including not only the classical *treewidth* (**?**), but also the *treedepth* (**?**), the *vertex cover number* (**?**), and the more recently introduced edge-cut variants of treewidth (**???**)), but even polynomial-time algorithms for instances where such parameters are bounded by a constant.

Theorem 1. SOAC *is* NP*-hard even when restricted to networks whose skeletons belong to the class* $K_{2,N}$.

Proof. We provide a polynomial-time reduction from MUKS where numbers are encoded in unary to SOAC. Recall that MUKS is NP-hard by Lemma 3. The reduction is as follows. For every input $\vec{v}_i \in S$ in the MUKS instance, there is a source vertex s_i. For each of the d entries $T_1, \ldots, T_d$ in the target vector $\vec{T}$, there is a target vertex t_j. There are also two vertices h_0 and h_1. Each of the n source vertices s_i is connected by outgoing arcs to both h_0 and h_1. The latencies are defined as $\ell_{s_i,h_0} = 0$ for all arcs (s_i, h_0), and as $\ell_{s_i,h_1}(x) = 1/x$ for all arcs (s_i, h_1). Both h_0 and h_1 have outgoing arcs to all the target vertices t_j. For arcs (h_0, t_j), $\ell_{h_0,t_j}(x) = 0$ if $x \leq T_j$, and otherwise, $\ell_{h_0,t_j}(x) = \infty$. That is, the capacity of the arc (h_0, t_j) is equal to T_j, for each $j \in [d]$. For arcs (h_1, t_j), $\ell_{h_1,t_j}(x) = 0$.

For the i-th input vector $\vec{v}_i$, let $v_{i,j}$ be its j-th entry. For all $i \in [n]$ and $j \in [d]$, we want to route $v_{i,j}$ agents from s_i to t_j. That is, there are $v_{i,j}$ copies of the pair (s_i, t_j) for all $i \in [n]$ and $j \in [d]$.

We claim that there is a solution of cost at most $n - k$ if and only if the original instance of MUKS has a solution comprised of at least k vectors. First, any solution $S' \subset S$ for MUKS translates immediately into a solution of cost at most $n - k$: the choice of latency $\ell_e(x) = 1/x$ ensures that $f(e)\ell_e(f(e)) = x \cdot 1/x = 1$, where $f(e) = x$. Hence, the cost of a solution is just $n - g$, where g is the number of *distinct origins* from which *all* agents are routed over h_0 (since every origin for which at least one agent is routed via h_1 adds 1 to the cost). Therefore, by construction, routing all agents from every s_i via h_0 whenever $\vec{v}_i \in S'$ transforms S' into a solution of cost at most $n - k$; in particular, the capacities of the edges from h_0 are respected.

On the other hand, suppose we are given a set of strategies of total cost at most $n-k$. This means that there are precisely $v_{i,j}$ agents routed from s_i to t_j for all $i \in [n]$ and $j \in [d]$. Again, by the property of the latency function $1/x$, there is a solution to the instance of SOAC of equal cost such that either none or all of the agents from a single origin are routed via h_1. For, suppose no agents are routed via h_1, then there is nothing to show. Otherwise, if at least one agent is routed via h_1, then the cost added by this is already 1, and routing all the remaining agents that originate in the same origin via h_1 will not add costs (and can only ever relax the capacity constraints). Hence, we can assume that all the agents from the same origin travel along the same vertex h_i for $i \in \{0, 1\}$ to their respective destinations in the solution. But, this means that there are at least k distinct origins that can be routed via h_0 and respect the capacity constraints, and hence, form a valid solution for the MUKS instance.

To complete the proof, note that the skeleton of the digraph in the constructed instance of SOAC above is $K_{2,d+n}$. □

Since Theorem 1 essentially rules out fixed-parameter algorithms based on structural network parameters alone, we turn our attention to parameterizing by the maximum capacity $\mathsf{c}_{\mathtt{max}}$. As our first result in this direction, we show that SOAC remains NP-hard on networks of constant treewidth, even if the maximum capacity of a link is just 1. This is done via a reduction from the following classical graph problem.

Edge-Disjoint Paths (EDP)
Input: A graph G and a set P of terminal pairs, that is, a set of subsets of $V(G)$ of size two.
Question: Is there a set of pairwise edge-disjoint paths connecting each set of terminal pairs in P?

Theorem 2. SOAC *is* NP*-hard even when restricted to networks with* $\mathsf{c}_{\mathtt{max}} = 1$ *and whose skeletons have treewidth or treedepth at most* 15.

Proof. It is known that EDP is NP-hard when restricted to the class $K_{3,N}$ (**?**). The reduction takes an instance $\mathcal{I}$ of EDP on $K_{3,N}$ and replaces each edge uv in the $K_{3,n}$ from $\mathcal{I}$ with the gadget depicted in Figure 3 (left), obtaining a digraph G'. We now complete the construction of the instance $\mathcal{I}'$ obtained from $\mathcal{I}$. First, the latency functions are set so that the capacity of each arc in G' is set to one, and a flow of at most one costs nothing, that is, for each $e \in E(G')$ and $x \in \mathbb{N}$,

$$\ell_e(x) = \begin{cases} 0 \text{ if } x \leq 1 \\ \infty \text{ otherwise.} \end{cases}$$

Then, for each terminal pair (s, t) in P in $\mathcal{I}$, we want to route 1 agent from s to t in G'. This completes the construction of $\mathcal{I}'$. By construction, given two vertices $u, v \in V(G')$ connected by an arc gadget, it is only possible to route at most one agent from u to v or from v to u via this arc gadget. Due to the capacity of each arc in G' being 1, we then obtain the

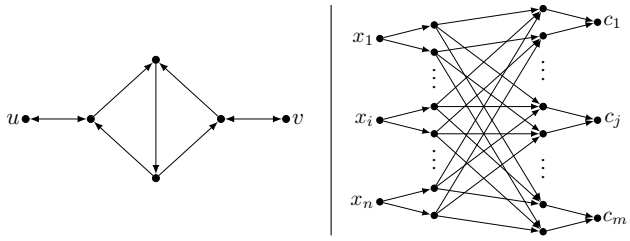

Figure 3: The arc gadget replacing each undirected edge uv in the reductions from the proofs of Thms. 2 and 6 (left), and the digraph constructed in the proof of Thm. 3 (right).

desired equivalence. That is, there is a solution to $\mathcal{I}$ if and only if there is a solution of cost 0 for $\mathcal{I}'$.

To complete the proof, we observe that the obtained network contains a bounded-sized set X of vertices (in our case, the part of size 3 in the $K_{3,n}$) such that deleting X decomposes the rest of the network into connected components each containing at most y vertices (in our case $y = 13$, as each such component consists of a vertex connected to 3 copies of the gadget depicted in Figure 3 (left). It is well-known (and also easy to see from their formal definitions (**???**)) that such graphs have treewidth and treedepth upper-bounded by $|X| + y - 1$ (i.e., 15). □

Theorem 2 rules out using classic vertex-separator based parameters like treewidth, pathwidth or treedepth to solve SOAC, even when $\mathsf{c}_{\mathtt{max}} = 1$. All of these parameters are tied to the skeleton of the network and do not account for the orientations of the arcs. One could wonder whether structural parameters introduced specifically for directed graphs would be better suited for the task. The unifying feature of the most widely studied of these parameters (**???**) is that they achieve constant values—typically 1—on directed acyclic graphs (DAGs). Below, we show that bounded-capacity SOAC is NP-hard even on simple DAGs, ruling out the use of these directed network parameters.

Theorem 3. SOAC *is* NP*-hard even when restricted to networks which are DAGs, have* $\mathsf{c}_{\mathtt{max}} = 3$*, and whose skeletons have maximum degree* 4*.*

Proof. We reduce from a variant of SAT known as monotone cubic exact 1-in-3 SAT, which is known to be NP-hard (**??**). Here, clauses have size three and must be satisfied by a single literal, each variable appears in three clauses, and all literals are positive.

Suppose we are given such an n-variate m-clause positive cubic 3-CNF formula φ. The instance of SOAC is constructed as follows. It contains the sets of vertices $X = \{x_1, \ldots, x_n\}$ and $C = \{c_1, \ldots, c_m\}$ that are identified with the variables and clauses appearing in φ, respectively. We also add two more copies of X, namely $X^T = \{x_1^T, \ldots, x_n^T\}$ and $X^F = \{x_1^F, \ldots, x_n^F\}$, and, similarly, two more copies C^T and C^F of C. Now, x_i has an arc to x_i^T and one to x_i^F, both with latency $\ell(1) = \ell(2) = 1$ and $\ell(3) = 0$, and capacity 3. Whenever x_i appears in the j-th clause of φ, then x_i^T has an arc to c_j^T and x_i^F has an arc to c_j^F, both of latency 0. Further, c_i^T has an arc to c_i of capacity 1 and c_i^F has an arc to c_i of capacity 2, for all $i \in [m]$, and latency 0 otherwise. If x_i appears in the j-th clause of φ, then there is an agent with origin x_i and destination c_j. An illustration of the construction is provided in Figure 3 (right).

We claim that this instance of SOAC has a solution of cost 0 if and only if φ is 1-in-3-satisfiable. Any satisfying 1-in-3 assigment of φ induces a routing of agents of cost 0 by construction. On the other hand, take any solution of cost 0 to SOAC. This means, first, that all agents with origin x_i are routed via the same choice of x_i^T or x_i^F by the respective latency functions of these arcs. In particular, we can derive an assignment π to the variables from this information, and we claim that it is in fact a satisfying 1-in-3 assignment. Towards showing that π is satisfying, observe that, by the capacity constraints on the arc from c_j^F to c_j, for every clause, at most 2 variables can use the incoming arcs into c_j^F from vertices x_i^F. Further, by the capacity constraints on the arc from c_j^T to c_j, for every clause, at most 1 variable can use the incoming arcs into c_j^T from vertices x_i^T. Hence, exactly one of the agents must be routed via c_j^T, coming from a vertex x_i^T. Thus, every clause is satisfied by π and π is also a 1-in-3 assignment of φ.

Note that the produced instance has the desired properties from the statement of the theorem by the fact that φ was assumed to be cubic. □

In fact, we can also show that our problem of interest remains intractable even on extremely simple networks which are DAGs (complementing Theorem 1).

Theorem 4. SOAC *is* NP*-hard even when restricted to DAGs whose skeletons are planar graphs with treewidth and maximum degree both upper-bounded by* 3*.*

Proof. To prove the statement of the theorem, we present a reduction from MUKS to SOAC. Starting with an instance $\mathcal{I}$ of MUKS encoded in unary, we construct an instance $\mathcal{I}'$ of SOAC as follows.

For each $\vec{v}_i \in S$ ($i \in [n]$) in $\mathcal{I}$, there is a source vertex s_i in $\mathcal{I}'$. For each entry $T_j \in \vec{T}$ ($j \in [d]$) in the target vector in $\mathcal{I}$, there is a terminal vertex t_j in $\mathcal{I}'$. The rest of $\mathcal{I}'$ is constructed as follows. There are two vertices h_i and h'_i for each $i \in [n+1]$. For each $i \in [n]$, there are the arcs (h_i, h_{i+1}), (h'_i, h'_{i+1}), (s_i, h_i), and (s_i, h'_i). There is a directed almost complete binary tree (the level with the leaves may not be completely filled) whose arcs go from the root to the leaves with h_{n+1} (h'_{n+1}, respectively) as the root and $t_1, \ldots, t_d$ as the leaves. The two directed almost complete binary trees are symmetric with respect to the leaves.

For each $i \in [n]$ and $x \in \mathbb{N}$, let $\ell_{(s_i,h'_i)}(x) = 1/x$. For each $j \in [d]$, let e_j be the arc incoming into t_j from the directed almost complete binary tree with h_{n+1} as the root, and, for each $x \in \mathbb{N}$, let

$$\ell_{e_j}(x) = \begin{cases} 0 \text{ if } x \leq T_j \\ \infty \text{ otherwise.} \end{cases}$$

In other words, for each $j \in [d]$, the capacity of the arc e_j is equal to T_j. For each remaining arc e for which a latency

function has not yet been defined, let $\ell_e(x) = 0$ for each $x \in \mathbb{N}$.

For each $\vec{v}_i \in S$ ($i \in [n]$) and $j \in [d]$, we want to route $v_{i,j}$ agents from s_i to t_j, where $v_{i,j}$ is the j-th entry of $\vec{v}_i$. That is, there are $v_{i,j}$ copies of the tuple (s_i, t_j) for each $i \in [n]$ and $j \in [d]$. This completes the construction of $\mathcal{I}'$, which is clearly achieved in polynomial time. See Figure 4 for an illustration.

We now show that there is a solution comprised of at least k vectors for $\mathcal{I}$ if and only if there is a solution of cost at most $n-k$ for $\mathcal{I}'$. First, assume there is a solution comprised of at least k vectors for $\mathcal{I}$. For each of the vertices s_i corresponding to one of these at least k vectors in the solution for $\mathcal{I}$, route each of the agents at s_i to their corresponding destination t_j via the directed path that contains h_i. Since these vectors are in the solution for $\mathcal{I}$, for each $j \in [d]$, the capacity of the arc e_j has not been surpassed, and thus, so far, the cost of routing these agents is 0 by the construction. The remaining agents are all routed to the t_j's via the directed path that contains h'_{n+1}. Since, for each $i \in [n]$ and $x \in \mathbb{N}$, we have that $\ell_e(x) = 1/x$ for any arc e of the form (s_i, h'_i), then $f(e)\ell_e(f(e)) = x \cdot 1/x = 1$, where $f(e) = x$. Hence, the cost of a solution for $\mathcal{I}'$ is at most $n - k$ since there are at most $n - k$ such s_i's where agents are routed to their destinations via h'_{n+1}, and the only arcs on these directed paths that have non-zero latency are the ones of the form (s_i, h'_i).

Now, suppose we are given a solution for $\mathcal{I}'$ of total cost at most $n - k$. By the property of the latency function $1/x$ for any arc of the form (s_i, h'_i), this means that there at most $n-k$ vertices in $\{h'_1, \ldots, h'_n\}$ that had agents routed to them through their incoming arc from s_i. Hence, for at least k of the vertices in $s_1, \ldots, s_n$, all of their agents must have been routed to the t_j's via the e_j's. Since the total cost of the solution for $\mathcal{I}'$ is at most $n - k$ (not ∞), then, for each $j \in [d]$, the capacity T_j of the arc e_j was not surpassed by this routing. Hence, the sum of the vectors corresponding to these at least k vertices in $s_1, \ldots, s_n$ must be at most $\vec{T}$.

It now remains to show that the digraph in $\mathcal{I}'$ has the desired properties from the theorem statement. It is trivial to see that the digraph in $\mathcal{I}'$ is a planar DAG, and that its skeleton has maximum degree 3. To show that the treewidth of its skeleton is at most 3, we give a strategy for four cops to capture the robber in the equivalent cops and robbers definition of treewidth: a graph G has treewidth k if and only if the minimum number of cops that can ensure capturing the robber in G is $k + 1$ (**?**).

The four cops first occupy h_{n+1}, h'_{n+1}, s_n, and h_n, respectively. If the robber occupies an s_i, h_i or h'_i vertex, then it is easy to see that the cops can capture the robber. Thus, assume that the robber territory is restricted to the two almost complete binary trees that are joined at the leaves minus their roots. The first and second cops occupy h_{n+1} and h'_{n+1}, respectively. The third and fourth cops move to a child of the first and second cops, respectively. Let the first and third cop be one team of cops, and let the second and fourth cops be the other team of cops. From this point on, the teams of cops play in an alternating fashion. The first and third (the second and fourth, respectively) cops then apply the following well-known winning strategy for two cops in their respective tree (**?**): at each point of time the two cops will occupy two pairwise adjacent vertices in the tree, and in each round the cop occupying the vertex closer to the root will move to a child of the other cop in the direction of the robber if the robber is located in their tree. If the robber is located in the other tree, the strategy is the same but we attempt to catch the "shadow" of the robber in their subtree, where the robber's shadow is simply the vertex corresponding to the robber's position in the mirroring almost complete binary tree. □

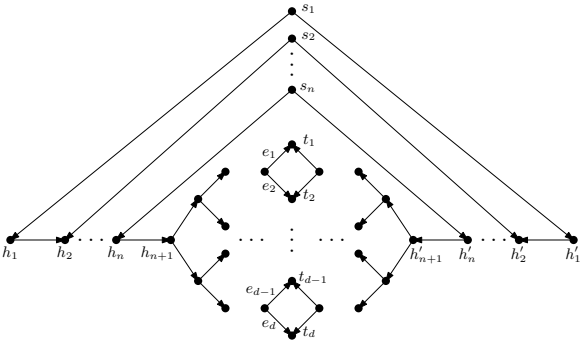

Figure 4: The digraph constructed in the proof of Thm. 4.

4 A Fixed-Parameter Algorithm for SOAC

Recapitulating the results so far, in view of Theorem 1, we are focusing our efforts on identifying structural properties of networks which would allow us to solve SOAC on bounded-capacity networks. Theorem 2 rules out tractability via standard vertex-separator based graph parameters, while Theorem 3 excludes the use of commonly studied directed variants of network parameters. A notable class of structural measures that has not been addressed yet by the obtained lower bounds are parameters that are tied to the existence of *edge cuts* as opposed to *vertex separators*.

Two "baseline" graph measures which allow for a structural decomposition along bounded-sized edge cuts are the *feedback edge number* (`fen`) (**??**), and the combined parameter of *treewidth plus maximum degree* (`twd`) (**??**). Unfortunately, both of these measures place very strong restrictions on the network: the former is only small on networks whose skeletons are trees plus a small number of arcs, while the latter cannot be used on networks with high-degree nodes.

As our main algorithmic contribution, we establish fixed-parameter tractability of bounded-capacity SOAC with respect to a recently introduced edge-cut based parameter called *slim treecut width* (`stcw`) (**?**). Crucially, this result immediately implies and generalizes fixed-parameter tractability with respect to either `fen` or `twd`, while also circumventing both of the aforementioned shortcomings: networks with skeletons of bounded slim treecut width can have high degree as well as be significantly more complex than just a tree.

Theorem 5. SOAC *is fixed-parameter tractable when parameterized by the slim treecut width of the skeleton* $\underline{G}$ *of the input digraph* G *plus the maximum capacity* $\mathtt{c}_{\mathtt{max}}$.

Proof. We begin by invoking Lemma 1 to compute a pair (H, T) such that H is a supergraph of $\underline{G}$ and (H, T) has edge-cut width $k \leq 3 \cdot (\kappa + 1)^2$, where κ is the slim treecut width of $\underline{G}$. Our algorithm is based on a leaf-to-root dynamic programming procedure that traverses T while storing certain (carefully defined and bounded-sized) records about the part of T that has been processed so far. To this end, it will be useful to assume that T is rooted, and thus, we mark an arbitrary leaf of T as the root and denote it r.

Before we define the records used in the dynamic program, we will need some terminology. For a vertex $v \in V(H)$ with a child $w \in V(H)$, we say that w is a *simple* child of v if v and w belong to different connected components of $H - vw$; otherwise, we say that w is a *complex* child of v. Observe that, while the number of simple children of v is not bounded by k (none of the subtrees rooted at simple children contain any edges in $E^{H,T}_{\mathrm{loc}}(v)$), the number of complex children of v is upper-bounded by $2k$ (each subtree rooted at a complex child contains an endpoint of at least one edge in $E^{H,T}_{\mathrm{loc}}(v)$). Furthermore, we use $\mathcal{G}_v$ to denote the subdigraph of G induced on the vertices that are descendants of v (including v itself), and ∂_v to denote those arcs of G which have precisely one endpoint in $\mathcal{G}_v$; recall that $|\partial_v| \leq 2(k+1)$. An agent a_i is *outgoing* for v if $s_i \in \mathcal{G}_v$ and, at the same time, $t_i \notin \mathcal{G}_v$. Similarly, an agent a_i is *incoming* for v if $s_i \notin \mathcal{G}_v$ and, at the same time, $t_i \in \mathcal{G}_v$.

We are now ready to formalize the dynamic programming records used in our algorithm. A *snapshot* at a vertex v is a tuple of the form $(\mathcal{S}_{out}, \mathcal{S}_{in}, \mathcal{D}, \mathcal{R})$, where:

- $\mathcal{S}_{out}$ is a mapping from the set of all outgoing agents for v to arcs in ∂_v which are outgoing from $\mathcal{G}_v$;
- $\mathcal{S}_{in}$ is a mapping from the set of all incoming agents for v to arcs in ∂_v which are incoming to $\mathcal{G}_v$;
- $\mathcal{D}$ is a multiset of pairs (e, f) such that e is an incoming arc into $\mathcal{G}_v$, f is an outgoing arc from $\mathcal{G}_v$, and ef is not a 2-cycle;
- $\mathcal{R}$ is a multiset of pairs (e, f) such that e is an outgoing arc from $\mathcal{G}_v$, f is an incoming arc into $\mathcal{G}_v$, and ef is not a 2-cycle;
- each arc in $\partial(v)$ may only appear in at most $\mathtt{c_{max}}$ tuples over all of the entries in $\mathcal{S}_{out}, \mathcal{S}_{in}, \mathcal{D}, \mathcal{R}$.

Before proceeding, it will be useful to obtain an upper-bound on the total number of possible snapshots for an arbitrary vertex v. First, we observe that if the number of outgoing agents for v exceeds $\mathtt{c_{max}} \cdot (k+1)$, then every flow assignment in the considered instance must necessarily exceed the capacity $\mathtt{c_{max}}$ for at least one arc in G; in other words, such an instance can be immediately recognized as a NO-instance. The same also holds for the number of incoming agents for v. Hence, we can restrict our attention to the case where these simple checks do not fail, and use this to obtain a bound on the total number of snapshots at v. In particular, there are at most $(2k+2)^{\mathtt{c_{max}} \cdot (k+1)}$ possible choices for each of $\mathcal{S}_{out}$ and $\mathcal{S}_{in}$ (there are at most $2k+2$ choices of which element of ∂_v to map each of the at most $\mathtt{c_{max}} \cdot (k+1)$ many agents to), and also at most $(\mathtt{c_{max}}+1)^{4(k+1)^2}$ possible choices for $\mathcal{D}$ and $\mathcal{R}$ each (for each of the at most $4(k+1)^2$ possible unordered pairs of arcs from ∂_v, there are $(\mathtt{c_{max}}+1)$ possible choices of how many times it occurs in $\mathcal{D}$ and $\mathcal{R}$). We conclude that the number of snapshots at v can be upper-bounded by $(\mathtt{c_{max}} + k)^{\mathcal{O}(\mathtt{c_{max}} k^2)}$, and let $\mathrm{Snap}(v)$ denote the set of all possible snapshots at v.

We can now formalize the syntax of the record at v, denoted $\mathrm{Record}(v)$, as a mapping from $\mathrm{Snap}(v)$ to $\mathbb{R}_{\geq 0} \cup \{\infty\}$. As for the semantics, $\mathrm{Record}(v)$ will capture the minimum cost required to (1) route the outgoing and incoming agents to the designated arcs in $\mathcal{S}_{out}$ and $\mathcal{S}_{in}$, while assuming that (2) some (unidentified and arbitrary) agents will use arcs of ∂_v to enter and then exit $\mathcal{G}_v$ via the arcs designated in $\mathcal{D}$, and that (3) some (unidentified and arbitrary) agents will use arcs of ∂_v to exit and then return to $\mathcal{G}_v$ via the arcs designated in $\mathcal{R}$. Formally, $\mathrm{Record}(v)$ maps each snapshot $\Upsilon = (\mathcal{S}_{out}, \mathcal{S}_{in}, \mathcal{D}, \mathcal{R})$ to the minimum cost of a flow assignment F in the subinstance $\mathcal{I}_\Upsilon$ induced on the vertices of $\mathcal{G}_v$ plus the arcs[3] of ∂_v with the following properties:

1. for each pair (ab, cd) of arcs in the multiset $\mathcal{D}$ such that $b, c \in \mathcal{G}_v$, we add a new marker agent into I_Υ which starts at a and ends at d;
2. for each pair (ab, cd) of arcs in the multiset $\mathcal{R}$ such that $a, d \in \mathcal{G}_v$, we add a new arc bc into $\mathcal{I}_\Upsilon$ and set $\ell_{bc} := \{1 \mapsto 0\} \cup \{i \mapsto \infty \mid i > 1\}$.
3. for each outgoing agent a_i, F contains a path from s_i into the arc $\mathcal{S}_{out}(a_i)$;
4. for each incoming agent a_i, F contains a path from (and including) the arc $\mathcal{S}_{in}(a_i)$ to t_i.

If no flow with these properties exists, we simply set $\mathrm{Record}(v)(\Upsilon) := \infty$. This completes the formal definition of the records $\mathrm{Record}(v)$. Observe that, since $\mathcal{G}_r = G$ and $\partial_r = \emptyset$, the only snapshot at r is $\Upsilon_r = (\{\emptyset\}, \{\emptyset\}, \emptyset, \emptyset)$ and $\mathcal{I}_{\Upsilon_r}$ is precisely the input instance to our problem. Hence, if we successfully compute the record $\mathrm{Record}(r)$ for the root vertex r, then $\mathrm{Record}(r)(\Upsilon_r)$ must be equal to the minimum cost of a flow assignment F in the input instance. Thus, to conclude the proof it remains to show how to compute the records at each vertex in a leaf-to-root fashion.

Towards this task, let us first consider the computation of $\mathrm{Record}(v)$ for a leaf v in T. Here, we observe that each of the constructed instances $\mathcal{I}_\Upsilon$ consists of the at most $2k+2$ arcs in ∂_v, and the at most $(k+1) \cdot \mathtt{c_{max}}$ arcs obtained from $\mathcal{R}$. The number of paths in such an instance is upper-bounded by $2^{\mathcal{O}(k \cdot \mathtt{c_{max}})}$, and hence, the minimum cost of a flow assignment F in $\mathcal{I}_\Upsilon$ can be computed by enumerating all flow assignments in time at most $\mathtt{c_{max}}^{2^{\mathcal{O}(k \cdot \mathtt{c_{max}})}}$.

The core of the dynamic program lies in the computation of $\mathrm{Record}(v)$ for a non-leaf vertex v. We do so by considering each snapshot $\Upsilon = (\mathcal{S}_{out}, \mathcal{S}_{in}, \mathcal{D}, \mathcal{R})$ at v independently, and computing the minimum cost of a flow assignment F in the subinstance $\mathcal{I}_\Upsilon$ as follows. For each simple child w of v, we observe that there is only a single snapshot $\Psi_w = (\mathcal{S}^w_{out}, \mathcal{S}^w_{in}, \{\emptyset\}, \{\emptyset\})$ of w where $\mathcal{S}^w_{out}$ maps all outgoing agents for w and $\mathcal{S}^w_{in}$ maps all incoming agents for w to the respective unique arcs in ∂_w. This

[3] We remark that since only one of the endpoints of ∂_v lies in $\mathcal{G}_v$, these are not formally arcs in $\mathcal{G}_v$.

corresponds to the fact that since a flow assignment is inherently loopless, there is only a unique way it may pass through the arcs in ∂_w. Let us define the *base cost* of v, denoted $b(v)$, as $\sum_{w \text{ is a simple child of v}} \text{Record}(w)(\Psi_w)$; intuitively, $b(v)$ captures the minimum cost required by a flow assignment in all simple children of v.

Next, we construct the instance $\mathcal{I}_\Upsilon$ for which we need to compute the minimum cost of a flow assignment. Essentially, our aim is to compute this cost via brute-forcing over all possible flow assignments, but at first glance this seems infeasible since the size of $\mathcal{I}_\Upsilon$ is not bounded by our parameters. The core insight we use to overcome this is that even though $\mathcal{I}_\Upsilon$ could be large, all but only a parameter-bounded number of interactions have already been taken into account in the records of the children of v. Formally, we make use of this by constructing a "kernelized" instance $\mathcal{I}^+_\Upsilon$ as follows:

First, for each simple child w of v, delete the whole subtree rooted at w, whereas, for each outgoing agent a_i at w, we place s_i on v, and, for each incoming agent a_i at w, we place t_i on v. Then, for each complex child w of v, we

1. delete every vertex in $\mathcal{G}_w$ except for the endpoints of ∂_w;
2. for each pair of endpoints $a, b \in V(\mathcal{G}_w)$ of distinct arcs in ∂_w, add bi-directional "marker" arcs between a and b;
3. create a vertex w_{out}, add a directed arc from w_{out} to every endpoint of an arc ∂_w in $V(\mathcal{G}_w)$, and for outgoing agent a_i at w we place s_i on w_{out};
4. create a vertex w_{in}, add a directed arc from every endpoint of an arc ∂_w in $V(\mathcal{G}_w)$ to w_{in}, and for an incoming agent a_i at w we place t_i on w_{in}.

We remark that the newly created arcs are not associated with a latency function (or alternatively may be assumed to be associated with the degenerate latency function $\mathbb{N} \to \{0\}$); these arcs are merely used as markers to point us to which of the snapshots should be used for each complex child of v. Crucially, the number of vertices in $\mathcal{I}^+_\Upsilon$ is upper-bounded by $(k \cdot 2k) + 1 \leq \mathcal{O}(k^2)$. The number of paths in such an instance is upper-bounded by $k^{\mathcal{O}(k^2)}$, and hence, the total number of all possible flow assignments can be upper-bounded by $\mathtt{c_{max}}^{k^{\mathcal{O}(k^2)}}$. We proceed by enumerating the set of all potential flow assignments, and, for each such potential flow assignment, we check whether it is indeed a flow assignment for the agents specified in $\mathcal{I}^+_\Upsilon$. If this check succeds, we employ a further technical check to ensure that each path in P only contains a marker arc e in a complex child w if e is immediately preceded and also succeeded by an arc in ∂_w.

For each flow assignment in $\mathcal{I}^+_\Upsilon$ that passes the above checks and, for each complex child w of v, we observe that F restricted to the arcs with at most one endpoint in w fully determines (1) the first arc in ∂_w used by an outgoing agent for w, (2) the last arc in ∂_w used by an incoming agent for w, (3) which pairs of arcs in ∂_w are used by paths to leave and subsequently re-enter $\mathcal{G}_w$, and (4) which pairs of arcs in ∂_w are used by paths to enter and subsequently leave $\mathcal{G}_w$. This information hence identifies a unique snapshot Ψ^w_F of w. We define the cost of F as $b(v) + \sum_{w \text{ is a complex child of } v} \text{Record}(w)(\Psi^w_F)$. Finally, we set $\text{Record}(v)(\Upsilon)$ to be the minimum cost of a flow F in $\mathcal{I}^+_\Upsilon$ which satisfies the conditions stipulated in this paragraph; this concludes the description of the algorithm.

Apart from the time required to compute (H, T), which is upper-bounded by $2^{\kappa^{\mathcal{O}(1)}} \cdot n^4$, the running time of the algorithm can be upper-bounded by the number of vertices in the input digraph times the cost of processing each vertex. The latter is dominated by $\mathtt{c_{max}}^{k^{\mathcal{O}(k^2)}}$, i.e., $\mathtt{c_{max}}^{\kappa^{\mathcal{O}(\kappa^4)}}$. We remark that as with essentially all width-based dynamic programming routines, the algorithm can be made constructive by performing a subsequent top-to-bottom computation in order to compute a specific flow assignment with the claimed flow as a witness. To argue correctness, it hence remains to argue that if the input instance admits a flow assignment Q of minimum cost, say p, then the algorithm will compute a flow assignment with the same cost p. Towards this goal, we observe that at each vertex v considered in the leaf-to-root pass made by the dynamic program, Q will correspond to a unique snapshot Υ of v. At each leaf v of T, $\text{Record}(v)(\Upsilon)$ must be equal to the cost incurred by Q on the arcs in ∂_v due to the nature of our brute-force computation of the records for trees and the optimality of Q. Moreover, for each non-leaf node v, it holds that as long as we have correctly computed the records for each of its children, the traversal of Q via the arcs in ∂_v and ∂_w for each child of w identifies a unique valid flow assignment F in $\mathcal{I}^+_\Upsilon$, and this in turn defines a snapshot Ψ^w_F for each child w of v. In that case, however, Q must indeed incur a cost of $b(v) + \sum_{w \text{ is a complex child of } v} \text{Record}(w)(\Psi^w_F)$ over all arcs in $\mathcal{I}_\Upsilon$, as desired. □

We observe that the parameterization by $\mathtt{c_{max}}$ cannot be dropped from Theorem 5 in view of the lower bound in Theorem 4; indeed, every network of bounded treewidth and maximum degree also has bounded slim treecut width (Lemma 2). We conclude by turning our attention to whether Theorem 5 could be generalized to use the better-known *treecut width* parameter instead of the slim variant used in that algorithm. Treecut width is a structural graph parameter that also guarantees the decomposability of instances via bounded-sized edge cuts, and has previously been used to establish tractability for several NP-hard problems (**?**). Crucially, it forms an intermediary between the slim treecut width (which suffices for fixed-parameter tractability) and treewidth (for which SOAC remains intractable, even when restricted to bounded-capacity instances). We show that Theorem 5 is tight in the sense that fixed-parameter tractability cannot be lifted to treecut width.

Theorem 6. SOAC *is* W[1]*-hard when parameterized by the treecut width of the skeleton of the input network, even when restricted to networks with* $\mathtt{c_{max}} = 1$.

Proof. EDP is known to be W[1]-hard parameterized by the treecut width of the graph, even when restricted to the class $K_{3,N}$ (**?**). To establish the theorem, we use the same reduction as in the proof of Theorem 2, and prove that the resulting instances also have bounded treecut width. Let us consider an output network G' obtained from that reduction.

Since the input graph G has bounded treecut width, it suffices to show that replacing each of the edges in G by the arc gadget depicted in Figure 3 (left) does not increase the treecut width by too much. We actually prove that $\mathbf{tcw}(\underline{G'}) \leq \mathbf{tcw}(G)$ since we can assume that $\mathbf{tcw}(G)$ can be assumed to be at least 5 in the reduction from $\mathcal{I}$ to $\mathcal{I}'$.

Consider a treecut decomposition $(T, \mathcal{X})$ of G of width $\mathbf{tcw}(G) \geq 5$. From $(T, \mathcal{X})$, we will construct a treecut decomposition $(T', \mathcal{X}')$ of $\underline{G'}$ of width at most $\mathbf{tcw}(G)$. Initially, set $T' = T$ and $\mathcal{X}' = \mathcal{X}$. Consider any edge $uv \in E(G)$ and let $U \subset V(G')$ be the four vertices in the arc gadget connecting u and v in G'. If u and v are in X_t for some node t of T, then, in T', we add a child t' of t such that $X_{t'} = U$. If u and v are in X_{t_1} and X_{t_2} for two different nodes t_1 and t_2 of T, and, without loss of generality, the depth of t_1 is at least the depth of t_2 in T, then, in T', we add a child t' of t_1 such that $X_{t'} = U$. In both cases, $\mathcal{X}'$ is updated accordingly.

Let $(T', \mathcal{X}')$ be the treecut decomposition of $\underline{G'}$ obtained from $(T, \mathcal{X})$ by applying the above procedure for each edge in $E(G)$. For each child t' added to T in the process of obtaining T', there are exactly four vertices in $X_{t'}$ and, in G', they are only adjacent to each other and two vertices $u, v \in X_t$ ($u \in X_{t_1}$ and $v \in X_{t_2}$, respectively) in the first (second, respectively) case. Thus, $\mathbf{tor}(t') \leq 5$, $\mathbf{adh}(t') \leq 2$, and the torso-size of any other node in $T' - t'$ in $(T', \mathcal{X}')$ is the same as in $(T, \mathcal{X})$. Indeed, this may only not be obvious in the case of the torso-size of t in T', but the vertex z, resulting from consolidating each vertex of $X_{t'}$ into a single vertex, only has degree 2 in $\underline{G'}$, and thus, z will be suppressed and an edge will be added between u and v (which already exists in G) when forming the 3-center of $(H_t, \mathcal{X}')$, where H_t is the torso of $(T', \mathcal{X}')$. Note that the depth of t' is always strictly greater than the depth of t (t_1 and t_2, respectively) in the first (second, respectively) case. Thus, the adhesion of any other node in $T' - t'$ in $(T', \mathcal{X}')$ is the same as in $(T, \mathcal{X})$. Hence, $\mathbf{tcw}(\underline{G'}) \leq \mathbf{tcw}(G)$. □

5 The Min-Max Atomic Congestion Problem

In this section, we turn our attention to the more general setting where instead of requiring *all* agents to be routed to their destinations, we allow for some agents to remain unrouted. In essence, this asks for a flow assignment that counterbalances the number of agents that reach their destinations with the total cost. As is usual in complexity-theoretic analysis, we state this task as a decision problem where we consider specific bounds on both the cost and the number of agents which need not be routed. For the purposes of this section, we formally extend the notion of *flow assignment* (defined in Section 2) to a mapping from a subset of agents to paths.

Min-Max System Opt. Atomic Congestion (MSOAC)
Input: A digraph $G = (V, E)$, positive integers λ and α, a set $A = \{a_1, \ldots, a_m\}$ of agents where each agent a_i is associated with a tuple $(s_i, t_i) \in V^2$, and for each arc $e \in E$, a latency function $\ell_e : [m] \to \mathbb{R}_{\geq 0} \cup \{\infty\}$.
Question: Is there a flow assignment F routing at least $m - \alpha$ agents such that $\sum_{e \in E} f_F(e) \cdot \ell_e(f_F(e)) \leq \lambda$?

We begin by noting that, in spite of the seemingly minor difference between the two problems, MSOAC is much more challenging than SOAC from a structural point of view. Indeed, on one hand, if we set $\alpha = 0$, the min-max variant becomes equivalent to SOAC. On the other hand, without this restriction, it can be observed that MSOAC remains NP-hard even on networks with a maximum capacity of 1 whose skeletons are trees. This follows by an immediate reduction from the MAXIMUM ARC DISJOINT PATHS problem, which has been shown to be NP-hard in precisely this setting (**?**); the reduction simply replaces each path with an agent and sets the latency function for each arc in the same way as in the proof of Theorem 2.

As the final contribution of this article, we show that MSOAC is fixed-parameter tractable when parameterized simultaneously by $\mathsf{c_{max}}$, α, and the combined parameter of treewidth and maximum degree.

Theorem 7. MSOAC *is fixed-parameter tractable when parameterized by the treewidth and maximum degree of the skeleton $\underline{G}$ of the input digraph G plus the maximum capacity $\mathsf{c_{max}}$ and α.*

Proof. On a high level, the algorithm follows the same overall approach as the one employed in the proof of Theorem 5; however, the dynamic programming steps and records required here are different (and more complicated). We begin by invoking Lemma 2 to compute a spanning tree T over a supergraph H of $\underline{G}$ such that there is an integer $k \in \mathcal{O}(d^2w^2)$ which is an upper-bound for both the edge-cut width of (H, T) and the maximum degree of T. As in the proof of Theorem 5, we mark an arbitrary leaf of T as the root and denote it r.

Recalling the notions of outgoing and incoming agents for a node v from Theorem 5, it will be useful to observe that if T contains a node v such that the number of outgoing or incoming agents in $\mathcal{G}_v$ exceeds $\alpha + \mathsf{c_{max}} \cdot (k+1)$, then we can immediately recognize the input as a NO-instance; this is because at most $\mathsf{c_{max}} \cdot (k+1)$ paths can traverse the edges forming the cut between $\mathcal{G}_v$ and the rest of the network. Hence, we hereinafter assume that the number of incoming and outgoing agents are both bounded by $\alpha + \mathsf{c_{max}} \cdot (k+1)$, and proceed to formalizing the dynamic programming records used in our algorithm. We define a similar notion of a *snapshot* at a vertex $v \in V(H)$, albeit here snapshots are tuples of the form $(\mathcal{A}_{out}, \mathcal{A}_{in}, \alpha', \mathcal{S}_{out}, \mathcal{S}_{in}, \mathcal{D}, \mathcal{R})$ where:

- $\mathcal{A}_{out}$ is a subset of the set of all outgoing agents for v;
- $\mathcal{A}_{in}$ is a subset of the set of all incoming agents for v;
- $\alpha' \leq \alpha$ is a non-negative integer;
- $\mathcal{S}_{out}$ is a mapping from $\mathcal{A}_{out}$ to arcs in ∂_v which are outgoing from $\mathcal{G}_v$;
- $\mathcal{S}_{in}$ is a mapping from $\mathcal{A}_{in}$ to arcs in ∂_v which are incoming to $\mathcal{G}_v$;
- $\mathcal{D}$ is a multiset of pairs (e, f) such that e is an incoming arc into $\mathcal{G}_v$, f is an outgoing arc from $\mathcal{G}_v$, and ef is not a 2-cycle;
- $\mathcal{R}$ is a multiset of pairs (e, f) such that e is an outgoing arc from $\mathcal{G}_v$, f is an incoming arc into $\mathcal{G}_v$, and ef is not a 2-cycle;

- each arc in $\partial(v)$ may only appear in at most $\mathtt{c}_{\max}$ tuples over all of the entries in $\mathcal{S}_{out}, \mathcal{S}_{in}, \mathcal{D}, \mathcal{R}$.

Comparing the snapshots defined above with those used in the proof of Theorem 5, the only difference is that we (1) use α' to keep track of how many agents have been left unrouted in $\mathcal{G}_v$, and (2) we use $\mathcal{A}_{out}$ and $\mathcal{A}_{in}$ to keep track of the identities of outgoing and incoming agents which are not being routed. Hence, we can upper-bound the total number of possible snapshots for an arbitrary vertex v as a product of the number of possible snapshots in the proof of Theorem 5 (i.e., $(\mathtt{c}_{\max} + k)^{\mathcal{O}(\mathtt{c}_{\max} k^2)}$) times the number of choices for α' (i.e., α) times the number of choices for $\mathcal{A}_{in}$ and $\mathcal{A}_{out}$ (i.e., $2^{\mathcal{O}(\alpha + \mathtt{c}_{\max} \cdot k)}$). Altogether, this yields an upper-bound of $(\mathtt{c}_{\max} + k)^{\mathcal{O}(\alpha \cdot \mathtt{c}_{\max} \cdot k^2)}$.

The syntax of the record at v, denoted $\mathrm{Record}(v)$ is a mapping from $\mathrm{Snap}(v)$ to $\mathbb{R}_{\geq 0} \cup \{\infty\}$. For the semantics, $\mathrm{Record}(v)$ will capture the minimum cost required to (1) route the outgoing and incoming agents specified in $\mathcal{A}_{out}$ and $\mathcal{A}_{in}$ to the designated arcs in $\mathcal{S}_{out}$ and $\mathcal{S}_{in}$ while leaving the remaining outgoing and incoming agents unrouted, under the following three assumptions: (2) some (unidentified and arbitrary) agents will use arcs of ∂_v to enter and then exit $\mathcal{G}_v$ precisely via the arcs designated in $\mathcal{D}$, (3) some (unidentified and arbitrary) agents will use arcs of ∂_v to exit and then return to $\mathcal{G}_v$ via the arcs designated in $\mathcal{R}$, and (4) precisely α' many agents with at least one endpoint in $\mathcal{G}_v$ are not routed to their destinations.

Formalizing the above, $\mathrm{Record}(v)$ maps each snapshot Υ to the minimum cost of a flow assignment F in the subinstance $\mathcal{I}_\Upsilon$ induced on the vertices of $\mathcal{G}_v$ plus the arcs[4] of ∂_v with the following properties:

1. for each pair (ab, cd) of arcs in the multiset $\mathcal{D}$ such that $b, c \in \mathcal{G}_v$, we add a new marker agent into I_Υ which starts at a and ends at d;
2. for each pair (ab, cd) of arcs in the multiset $\mathcal{R}$ such that $a, d \in \mathcal{G}_v$, we add a new arc bc into $\mathcal{I}_\Upsilon$ and set $\ell_{bc} := \{1 \mapsto 0\} \cup \{i \mapsto \infty \mid i > 1\}$.
3. for each outgoing agent a_i in $\mathcal{A}_{out}$, F contains a path from s_i into the arc $\mathcal{S}_{out}(a_i)$;
4. for each incoming agent a_i in $\mathcal{A}_{in}$, F contains a path from (and including) the arc $\mathcal{S}_{in}(a_i)$ to t_i;
5. F routes all agents with at least one endpoint in F except for α' many. Moreover, each outgoing (incoming, respectively) agent that is not in $\mathcal{A}_{out}$ ($\mathcal{A}_{in}$, respectively) is not routed by F.

We remark that if no flow with these properties exists, we simply set $\mathrm{Record}(v)(\Upsilon) := \infty$. This completes the formal definition of the records $\mathrm{Record}(v)$. Observe that since $\mathcal{G}_r = G$ and $\partial_r = \emptyset$, the only snapshots at r are of the form $\mathcal{I}_{\alpha'} = (\emptyset, \emptyset, \alpha', \{\emptyset\}, \{\emptyset\}, \emptyset, \emptyset)$, and each of these snapshots is mapped by $\mathrm{Record}(r)$ to the minimum cost of a flow assignment in the input network which routes all but precisely α' agents to their destinations. Hence, $\min_{\alpha' \in [\alpha]} \mathrm{Record}(r)(\mathcal{I}_{\alpha'})$ is the minimum cost of a flow assignment F which routes at least $m - \alpha$ agents, and thus, to conclude the proof it remains to show how to compute the records at each vertex in a leaf-to-root fashion.

Towards this task, let us first consider the computation of $\mathrm{Record}(v)$ for a leaf v in T. This step is based on exhaustive branching and is entirely analogous to the one employed in the proof of Theorem 5, but with the distinction that we also branch on $\mathcal{A}_{in}$ and $\mathcal{A}_{out}$ (it is worth noting that, for leaves, each choice of $\mathcal{A}_{in}$ and $\mathcal{A}_{out}$ fully determines the value of α'). More precisely, we observe that each of the constructed instances $\mathcal{I}_\Upsilon$ consists of the at most $2k + 2$ arcs in ∂_v, the at most $(k+1) \cdot \mathtt{c}_{\max}$ arcs obtained from $\mathcal{R}$, and the at most $(2k+2) \cdot \mathtt{c}_{\max}$ agents that are in $\mathcal{A}_{in}$ or $\mathcal{A}_{out}$ for v. The number of paths in such an instance is upper-bounded by $2^{\mathcal{O}(k \cdot \mathtt{c}_{\max})}$. Hence, the minimum cost of a flow assignment F in $\mathcal{I}_\Upsilon$ that routes all the agents in $\mathcal{A}_{in}$ and $\mathcal{A}_{out}$ can be computed by enumerating all flow assignments in time at most $\mathtt{c}_{\max}^{2^{\mathcal{O}(k \cdot \mathtt{c}_{\max})}}$.

The core of the dynamic program lies in the computation of $\mathrm{Record}(v)$ for a non-leaf vertex v. We do so by considering each snapshot $\Upsilon = (\mathcal{A}_{out}, \mathcal{A}_{in}, \alpha', \mathcal{S}_{out}, \mathcal{S}_{in}, \mathcal{D}, \mathcal{R})$ at v independently, and computing the minimum cost of a flow assignment F in the subinstance $\mathcal{I}_\Upsilon$ as follows. Unlike in Theorem 5, we directly construct the instance $\mathcal{I}_\Upsilon$ for which we need to compute the minimum cost of a flow assignment, as here we do not distinguish between "simple" and "complex" children, followed by a construction of the same "kernelized" instance $\mathcal{I}^+_\Upsilon$ as earlier (whereas this time, we treat *every* child of v as complex). In particular, for each child w of v, we

1. delete every vertex in $\mathcal{G}_w$ except for the endpoints of ∂_w;
2. for each pair of endpoints $a, b \in V(\mathcal{G}_w)$ of distinct arcs in ∂_w, add bi-directional "marker" arcs between a and b;
3. create a vertex w_{out}, add a directed arc from w_{out} to every endpoint of an arc ∂_w in $V(\mathcal{G}_w)$, and for outgoing agent a_i at w we place s_i on w_{out};
4. create a vertex w_{in}, add a directed arc from every endpoint of an arc ∂_w in $V(\mathcal{G}_w)$ to w_{in}, and for an incoming agent a_i at w we place t_i on w_{in}.

As before, the newly created arcs are not associated with a latency function (or alternatively may be assumed to be associated with the degenerate latency function $\mathbb{N} \to \{0\}$), and we observe that the number of vertices in $\mathcal{I}^+_\Upsilon$ is upper-bounded by $(k \cdot 2k) + 1 \leq \mathcal{O}(k^2)$. The number of paths in such an instance is upper-bounded by $k^{\mathcal{O}(k^2)}$, and hence, the total number of all possible flow assignments (regardless of the subset of routed agents) can be upper-bounded by $\mathtt{c}_{\max}^{k^{\mathcal{O}(k^2)}}$. We proceed by branching over each choice of a potential flow assignment F from this set of all potential flow assignments.

Next, we deal with the fact that F need not be a flow assignment of all agents by performing an additional bounded branching step to determine which of the outgoing and incoming agents from each child of v is actually routed by F. To this end, let an agent be *important* for v if it is an incoming or outgoing agent for at least one child of v, but is neither an incoming nor outgoing agent for v; in other words, important agents are those which are explicitly tracked by

[4] As before, only one of the endpoints of ∂_v lies in $\mathcal{G}_v$, and hence, these are not formally arcs in $\mathcal{G}_v$.

the snapshots of the children of v, but are no longer tracked by snapshots of v itself. The number of important agents is upper-bounded by $k \cdot (\alpha + \mathtt{c}_{\max} \cdot k)$, and we branch over each subset $\mathcal{Z}$ of important children for v. Finally, we branch over all mappings β from the children of v to $[\alpha]$. For each fixed F, $\mathcal{Z}$, and β, we check that F routes the agents in $\mathcal{Z}$ to their final destinations and the agents in $\mathcal{A}_{in}$ and $\mathcal{A}_{out}$ to their assigned arcs as per $\mathcal{S}_{in}$ and $\mathcal{S}_{out}$, respectively. As previously in the proof of Theorem 5, we also perform a further technical check to ensure that each path in P only contains a marker arc e in a child w if e is immediately preceded and also succeeded by an arc in ∂_w. If these checks succeed, we view F as a "projection" of a flow assignment in $\mathcal{I}_\Upsilon$ onto $\mathcal{I}^+_\Upsilon$. In particular, F restricted to the arcs with at most one endpoint in a child w of v fully determines (1) the first arc in ∂_w used by an outgoing agent for w, (2) the last arc in ∂_w used by an incoming agent for w, (3) which pairs of arcs in ∂_w are used by paths to leave and subsequently re-enter $\mathcal{G}_w$, and (4) which pairs of arcs in ∂_w are used by paths to enter and subsequently leave $\mathcal{G}_w$. This information, combined with information about which outgoing and incoming agents for w are actually routed by the flow (specified in $\mathcal{Z}$) and information about about how many total agents with an endpoint in $\mathcal{G}_w$ are not routed by the flow (specified in $\beta(w)$), hence fully identifies a unique snapshot Ψ^w_F of w. We define the cost of F as $b(v) + \sum_{w \text{ is a child of } v} \text{Record}(w)(\Psi^w_F)$. Finally, we set $\text{Record}(v)(\Upsilon)$ to be the minimum cost of a flow F in $\mathcal{I}^+_\Upsilon$ which satisfies the conditions stipulated above; this concludes the description of the algorithm.

The running time of the algorithm can be upper-bounded by the number of vertices in the input digraph times the cost of processing each vertex, whereas the latter is dominated by $\mathtt{c}_{\max}^{(dw)^{\mathcal{O}(d^4w^4)}} \cdot 2^{d^2w^2 \cdot (\alpha + \mathtt{c}_{\max} \cdot d^2w^2)} \cdot \alpha^{w^2d^2}$, i.e., by $2^{(d \cdot w \cdot \alpha \cdot \mathtt{c}_{\max})^{\mathcal{O}(d^4w^4)}}$. The algorithm can be made constructive in the same way as in Theorem 5. For correctness, we argue that if the input instance admits a flow assignment Q of at most α agents with some minimum cost, say p, then the algorithm will compute a flow assignment with the same cost p. Towards this goal, we observe that at each vertex v considered in the leaf-to-root pass made by the dynamic program, Q will correspond to a unique snapshot Υ of v. At each leaf v of T, $\text{Record}(v)(\Upsilon)$ must be equal to the cost incurred by Q on the arcs in ∂_v due to the nature of our brute-force computation of the records for trees and the optimality of Q. Moreover, for each non-leaf node v it holds that as long as we have correctly computed the records for each of its children, the traversal of Q via the arcs in ∂_v and ∂_w for each child of w identifies a unique valid flow assignment F in $\mathcal{I}^+_\Upsilon$. Moreover, from Q we can also recover a unique set $\mathcal{Z}$ of important agents for v as well as a unique mapping β which specifies how many agents were not routed among those with at least one endpoint in each of the children of v. These sets altogether define a snapshot Ψ^w_F for each child w of v. In that case, however, Q must indeed incur a cost of $b(v) + \sum_{w \text{ is a child of } v} \text{Record}(w)(\Psi^w_F)$ over all arcs in $\mathcal{I}_\Upsilon$, as desired. □

6 Concluding Remarks

Our results provide an essentially comprehensive complexity landscape for the problem of computing system-optimal flow assignments in atomic congestion games, closing a gap in the literature that contrasts with the significant attention other aspects of congestion games have received to date. We remark that our tractability results only require the input network to have the necessary structural properties and do not impose any restrictions on the possible origins and destinations of the agents. Moreover, all of the obtained algorithms can also be used to compute Nash-equilibria in atomic congestion games as long as an upper-bound on the cost of the flow is provided in the input. Future work could also consider the recently proposed setting of having some agents follow a greedily computed route (**?**). Another interesting avenue for future work would be to resolve the complexity of the min-max variant of the problem (i.e., MSOAC) on well-structured networks of unbounded degree. This problem is left open even on stars when parameterized by $\mathtt{c}_{\max} + \alpha$, and we believe novel ideas will be required to breach this barrier; in particular, the techniques for solving the maximization variant of the related ARC DISJOINT PATHS problem on stars (**?**) do not generalize to MSOAC. As a longer-term goal, one would be interested in settling whether Theorem 7 could be lifted towards an analog of Theorem 5 that relies on the same structural measures of the network.

7 Acknowledgements

The first, second, and fourth authors were supported by the Austrian Science Foundation (FWF, project Y1329). The third author was supported by SERB-DST via grants MTR/2020/000497 and CRG/2022/009400. Quaerat voluptatem aut, eveniet itaque exercitationem quae quam?Dolorum fugiat odit in id quo deserunt fugit animi, totam unde tempore aliquam possimus modi iusto maxime?Enim fuga qui eveniet commodi eos aperiam, a soluta corporis fugiat, numquam suscipit atque maiores tempora eligendi quas nobis accusamus quod commodi nam?Corrupti ea harum quis delectus, perspiciatis iusto inventore esse.Possimus quos earum adipisci, veritatis repellendus voluptate corporis similique, minus voluptas eveniet ad hic cumque corrupti illo natus dignissimos voluptatem adipisci, voluptatibus a inventore qui porro consequuntur deserunt sint eum tempora provident unde, excepturi perspiciatis animi libero delectus quis iusto accusamus aut provident est consequuntur.Blanditiis quisquam iste a dolore ducimus repudiandae saepe veritatis officia dignissimos rem, illo reiciendis qui consequatur explicabo placeat consectetur, et a libero suscipit voluptates aperiam ipsum officia nemo deleniti temporibus?Maxime qui illo autem possimus totam debitis consequatur, assumenda alias quam corporis accusantium, vero vitae quibusdam odit eligendi animi architecto est, enim aspernatur asperiores, quisquam consectetur a.Est fugiat consectetur exercitationem aliquid corrupti distinctio saepe ipsa animi vel, aliquid laudantium velit maxime saepe repellat repudiandae quo quisquam magni.Illum ipsum mollitia eveniet provident illo quod esse ut, exercitationem placeat molestias qui asperiores repudiandae esse atque, cupiditate nostrum debitis doloribus, exercitationem

sit saepe a labore aperiam quasi vero dolor voluptas ipsum.