

Charting the Complexity Landscape of Waypoint Routing

Saeed Akhoondian Amiri¹ Klaus-Tycho Foerster² Riko Jacob³ Stefan Schmid²
¹ TU Berlin, Germany ² Aalborg University, Denmark ³ IT University of Copenhagen, Denmark

Abstract—Modern computer networks support interesting new routing models in which traffic flows from a source s to a destination t can be flexibly steered through a sequence of waypoints, such as (hardware) middleboxes or (virtualized) network functions (VNFs), to create innovative network services like service chains or segment routing. While the benefits and technological challenges of providing such routing models have been articulated and studied intensively over the last years, much less is known about the underlying algorithmic traffic routing problems. This paper shows that the waypoint routing problem features a deep combinatorial structure, and we establish interesting connections to several classic graph theoretical problems. We find that the difficulty of the waypoint routing problem depends on the specific setting, and chart a comprehensive landscape of the computational complexity. In particular, we derive several NP-hardness results, but we also demonstrate that exact polynomial-time algorithms exist for a wide range of practically relevant scenarios.

I. INTRODUCTION

A. The Motivation: Service Chaining and Segment Routing

We currently witness two trends related to the increasing number of middleboxes (e.g., firewalls, proxies, traffic optimizers, etc.) in computer networks (in the order of the number of routers [1]): First, there is a push towards virtualizing middleboxes and network functions, enabling faster and more flexible deployments (not only at the network edge), and reducing costs. Second, over the last years, innovative new network services have been promoted by industry and standardization institutes [2], by *composing* network functions to *service chains* [3], [4], [5]. The benefits and technological challenges of implementing such more complex network services have been studied intensively, especially in the context of Software-Defined Networks (SDNs) and Network Function Virtualization (NFV), introducing unprecedented flexibilities on how traffic can be steered through flexibly allocated (virtualized) network functions (VNFs).

However, much less is known today about the algorithmic challenges underlying the routing through such middleboxes or network functions, henceforth simply called *waypoints*. In a nutshell, the underlying algorithmic problem is the following: How to route a flow (of a certain size) from a given source s to a destination t , via a sequence of k waypoints (w_1, \dots, w_k) ? The allocated flow needs to respect capacity constraints, and ideally, be as short as possible.

The problem can come in many different flavors, depending on whether a shortest or just a feasible route needs to be computed, depending on the number k of waypoints, depending

on the type of the underlying network (e.g., directed vs undirected, Clos vs arbitrary topology), etc. Moreover, as middleboxes provide different functionality (mostly security and performance related), waypoints may or may not be *flow-conserving*: e.g., a tunnel entry point may *increase* the packet size (by adding an encapsulation header) whereas a wide-area network optimizer may *decrease* the packet size (by compressing the packet).

The goal of this paper is to develop algorithmic techniques to solve the different variants of the waypoint routing problem, as well as to explore limitations due to computational intractability.

B. The Problem: Waypoint Routing

More formally, inputs to the *waypoint routing problem* are:

- 1) **A network:** represented as a graph $G = (V, E)$, where V is the set of $n = |V|$ switches/routers/middleboxes (i.e., the nodes) and where the set E of $m = |E|$ links can either be undirected or directed, depending on the scenario. Moreover, each link $e \in E$ may have a bandwidth capacity $c(e)$ and weights $\omega(e)$ (describing costs), both non-negative. If not stated otherwise, we assume that $c(e) = 1$ and $\omega(e) = 1$ for all $e \in E$.
- 2) **A source-destination pair (s, t) and a sequence of waypoints (w_1, \dots, w_k) :** which need to be traversed along the way from s to t , forming a route (s, w_1, \dots, w_k, t) . Unless specified otherwise, we will assume at most one waypoint per node, though it may be that $s = t$. Waypoints may also change the traffic rate: We will denote the demand from s to w_1 by d_0 , from w_1 to w_2 by d_1 , etc. That said, if not stated explicitly otherwise, we will assume that $d_0 = d_1 = \dots = d_k = 1$, and refer to this scenario as *flow-conserving*.

In general, we are interested in shortest routes (an **optimization problem**), i.e., routes of minimal length $|R|$, such that link capacities are respected. However, we also consider the *feasibility* of such routes: is it possible to route such a flow without violating link capacities at all (a **decision problem**)? Sometimes, minimizing the total route length alone may not be enough, but additional, **hard constraints** on the distance (or stretch) between a terminal and a waypoint or between waypoints may be imposed.

C. Novelty: It's a walk!

We will show that the waypoint routing problem is related to some classic and deep combinatorial problems, in particular the disjoint path problem [6], [7], [8] and the k -cycle problem [9].

arXiv:1705.00055v2 [cs.NI] 1 Sep 2017

	# Waypoints	Feasible	Optimal	Demand Change Feasible	Optimal
Undirected	1	P (Thm. 1)		Strongly NPC (Thm. 2)	
	constant	P (Thm. 5)	?		
	arbitrary	Strongly NPC (Thm. 6)			
Directed	1	Strongly NPC (Thm. 3)			
	constant				
	arbitrary				

TABLE I
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN GENERAL GRAPHS.

In contrast to these problems, however, the basic waypoint routing problem considered in this paper comes with a fundamental twist: routes are not restricted to form simple paths, but can rather form arbitrary *walks*, as long as capacity constraints in the underlying network are respected. Indeed, often feasible routes do not exist if restricted to a simple path, see Fig. 1 for an example in which any feasible route must contain a loop.

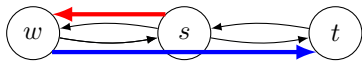


Fig. 1. A route (s, w, t) in the depicted network must contain a loop. The only solution is the walk s, w, s, t , resulting from concatenating the red (s, w) and blue (w, t) paths. It can hence not be described as a simple path.

The problem is non-trivial. For example, consider the seemingly simple problem of routing via a *single* waypoint, i.e., a route of the form (s, w, t) . A naive algorithm could try to first compute a shortest path from s to w , deduct the resources consumed along this path, and finally compute a shortest path (subject to capacity constraints) from w to t on the remaining graph. However, as we will see shortly, such a greedy algorithm is doomed to fail; rather, route segments between endpoints and waypoints must be *jointly optimized*.

D. Our Contributions

This paper initiates the algorithmic study of the waypoint routing problem underlying many modern networking applications, such as service chaining [3] (where traffic needs to be steered through network functions), hybrid SDNs [10] (where traffic is steered through OpenFlow switches) or in segment routing [11] (where MPLS labels are updated at segment endpoints).

We show that whether and how efficiently a *feasible* or *shortest* waypoint route can be found depends on the scenario, and chart a complexity landscape of the waypoint routing problem, presenting a comprehensive set of NP-hardness results and efficient algorithms for different scenarios. In particular, we establish both simple and non-trivial reductions *from* resp. *to* classic combinatorial problems, and also derive several new algorithms from scratch which may be of interest beyond the scope of this paper.

In summary, we make the following contributions. For a single waypoint ($k = 1$), we show the following:

- 1) **Waypoint routes can be computed efficiently on undirected graphs:** We establish a connection to the

classic disjoint paths problem, but show that while the 2-disjoint paths problem is notoriously hard and continues to puzzle researchers [6], a route via a single waypoint can in fact be computed very efficiently.

- 2) **Waypoints which change the flow size are challenging:** We show that routing through a single waypoint is NP-hard in general if the waypoint changes the flow. This can be seen as an interesting new insight into the classic 2-disjoint paths problem as well.
- 3) **Directed links make it hard as well:** While we describe fast algorithms for undirected networks, the waypoint routing problem turns out to be NP-hard already for a single waypoint on directed graphs.
- 4) **Supporting absolute distance and stretch constraints is difficult:** We point out another frontier for the computational tractability of computing routes through a single waypoint: the problem also becomes NP-hard if in addition to minimizing the total length of the route, there are hard distance (or stretch constraints) between the source resp. destination and the waypoint.

For multiple waypoints (arbitrary k), we show:

- 1) **Routes through a fixed number of waypoints can be computed in polynomial time:** This result follows by a reduction to a classic result by Robertson and Seymour [12].
- 2) **Already the decision problem is hard in general:** For general k , even on undirected graphs, the decision problem (whether a feasible route *exists*) is NP-hard.

Motivated by these results and the fact that the topologies of real-world networks (e.g., datacenter, enterprise, carrier networks) are often not arbitrary but feature additional structure, we take a closer look at special networks.

- 1) **In reality, there is hope:** We present several algorithms to compute shortest waypoint routes on specific graph families, and in particular, on networks of bounded treewidth.
- 2) **An accurate characterization of tractability:** We show that it is difficult to go significantly beyond the graph families studied above, by deriving NP-hardness results on slightly more general graph families already (graphs of treewidth three).

An overview of our complexity results derived in this paper can be found in Table I. We further note that in the following figures, we will draw (s, w) paths in solid red and (w, t) paths in solid blue, depicting alternative paths in a dotted style.

E. Paper Organization

We start in Sec. II by considering routing problems via a single waypoint, before studying multiple waypoints in Sec. III. We then discuss our case study in Sec. IV, covering further related work in Sec. V, before concluding in Sec. VI.

II. ROUTING VIA A WAYPOINT

We start by considering the fundamental problem of how to route a flow from s to t via a *single* waypoint w .

A. Undirected Graphs Are Tractable

Many graph theoretical problems revolve around undirected graphs, and we therefore also consider them first. In undirected graphs, flows can consume bandwidth capacity in both directions: e.g., a link of capacity two can accommodate two unit-size flows traversing it both in opposite directions as well as in the same direction.

Before delving into the details of our algorithms and hardness results, we make some general observations. First, we observe that a (shortest) route (s, w, t) can be decomposed into two segments (s, w) and (w, t) . While (s, w, t) can contain loops, the two segments (s, w) and (w, t) are *simple paths*, without loss of generality: any loop on a route segment can simply be shortcut. More generally, we make the following observation.

Observation 1: A shortest route (describing a walk) through k waypoints can be decomposed into $k + 1$ simple paths P_i between terminals and waypoints: $R = (P_1, \dots, P_{k+1})$.

Second, we observe that we can transform the capacitated problem variant to an uncapacitated one, by replacing capacitated links with a (rounded-down) number of parallel, uncapacitated links. Computing a capacity-respecting walk on the capacitated graph is then equivalent to computing a link-disjoint path on the uncapacitated network.¹

These observations provide us with a first idea to compute a shortest route (s, w, t) : we could simply compute the two optimal paths (s, w) and (w, t) independently. That is, we could route the first segment from s to w along the shortest path, subtract the consumed bandwidth along the path, and then compute a shortest feasible path from w to t on the remaining graph. The example depicted in Fig. 2 shows how this strategy fails. Therefore, we conclude that in an undirected setting, we need to *jointly* optimize the two paths.

However, the above observations also allow us to compute an optimal solution: the computation of shortest link-disjoint paths (s_1, t_1) and (s_2, t_2) is a well-known combinatorial problem, to which we can directly reduce the waypoint routing problem by setting $s_1 = s, t_1 = s_2 = w, t_2 = t$. Unfortunately however, while a recent breakthrough result [6] has shown how to compute shortest two disjoint paths in randomized polynomial time, the result is a theoretical one: the order of the runtime polynomial is far from practical.

¹ If parallel links are undesired, each link could additionally be subdivided by placing an additional node in the middle, and splitting the link cost in two halves accordingly. As discussed later in more details, note however that the resulting topology may have different properties than the original one (due to the parallel resp. subdivided links).

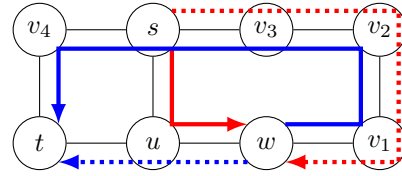


Fig. 2. In undirected graphs, path segments need to be jointly optimized: greedily selecting a shortest path from s to w can force a very long path from w to t . Once the solid red (s, w) path has been inserted first as a shortest path, there is only one option for the solid blue (w, s) path, resulting in a walk length of $2 + 6 = 8$. A joint optimization leads to the dotted red (s, w) path and the dotted blue (w, t) path, with a total length of $4 + 2 = 6$.

Yet, there is hope: our problem is strictly simpler, as the two paths have a common endpoint $t_1 = s_2 = w$. Indeed, the common endpoint w can be leveraged to employ a reduction to an integer flow formulation: introduce a super-source S^+ and a super-destination T^+ , connect S^+ to s and t , and T^+ to w with two links, all of unit capacity, see Fig. 3.

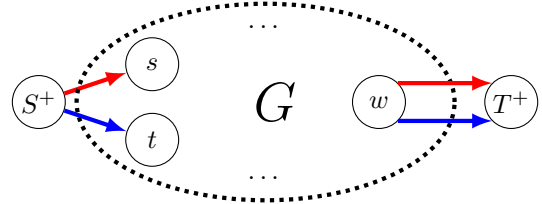


Fig. 3. By adding a super-source S^+ , we can reduce the waypoint routing problem on undirected graphs to a min cost flow problem: As a $w - t$ flow is also a $t - w$ flow, we can check if there is a flow of size 2 from S^+ to T^+ . An analogous idea can be used to reduce the waypoint routing problem to finding two link-disjoint paths from S^+ to T^+ , later reversing the blue path direction in the undirected case.

Next, solve the minimum cost integer flow problem from S^+ to T^+ with a demand of 2. By performing flow decomposition and removing S^+, T^+ , we obtain an $s - w$ and a $w - t$ flow, whose combined length is minimum. Note that in undirected graphs, any $s - t$ flow can also be interpreted as a $t - s$ flow. It is well-known that this flow problem can be solved fairly efficiently: for a single source and a single destination, the minimum cost integer flow can be solved in polynomial time $O((m \log m)(m + n \log n))$, cf. [13, p. 227]².

But there exist even better solutions. We can leverage a reduction to a problem concerned with the computation of two (shortest) disjoint paths *between the same endpoints s and t* . For this problem, there exists a well-known and fast algorithm by Suurballe for node-disjoint paths [14]: it first uses Dijkstra's algorithm to find a first path, modifies the graph links, and then runs Dijkstra's algorithm a second time. It was extended 10 years later to link-disjoint paths by Suurballe and Tarjan [15]:

Theorem 1: On undirected graphs with non-negative link weights, the shortest waypoint routing problem can be solved for a single waypoint in time $O(m \log_{(1+m/n)} n)$.

Proof: We will make use of Suurballe's algorithm extended to the link-disjoint case [15] in our proof, which solves

² Undirected instances can be turned into directed ones, by replacing each link by two antiparallel directed links. After obtaining a directed solution, flows on antiparallel links can be canceled out, obtaining an undirected solution.

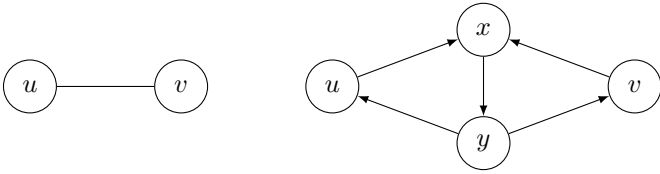


Fig. 4. By replacing every undirected link with the construction to the right, we can apply algorithms for directed graphs to undirected graphs. Observe that in both cases, the integer flow possibilities between u and v are identical. Regarding link weights, we set the weight from x to y to be the one of the undirected case, with all other four weights being zero.

the following problem in time $O(m \log_{(1+m/n)} n)$: Given a directed graph $G = (V, E)$, find two link-disjoint paths from s to t , with $s, t \in V$, where their combined length is minimum. To apply it to the undirected case, we can make use of a standard reduction from undirected graphs to directed graphs for link-disjoint paths, replacing every undirected link with five directed links [16], see Figure 4.

As the flow orientation is not relevant on undirected graphs, we obtain a solution for finding two link-disjoint paths from s to t on undirected graphs.

Note that the above applies to unit link capacities, which we extend to larger link capacities as follows: We can apply a standard reduction technique, creating two parallel undirected links if the capacity suffices. Observe that more than two parallel links do not change the feasibility.

Now add a super-source S^+ and a super-destination T^+ to the transformed directed graph, connecting S^+ to s and t with links of unit capacity, and T^+ to w with two links of unit capacity, see Fig. 3. To remove the parallel link property from the graph, nodes are placed on all links, splitting them into a path of length two, scaling path lengths by a factor of two. In total, the number of nodes and links are still in $O(n)$ and $O(m)$, respectively, allowing us to run Suurballe’s extended algorithm in $O(m \log_{(1+m/n)} n)$. Lastly, by removing S^+, T^+ , translating the graph back to be undirected, and scaling the path lengths back, we obtain an $s - w$ and a $w - t$ path, whose combined length is minimum. If no solution exist, Suurballe’s extended algorithm will notice it during its execution. ■

Thus, we conclude that finding a shortest (s, w, t) walk is significantly simpler than shortest two paths $(s_1, t_1), (s_2, t_2)$. **Remark.** One might wonder whether the above approach can also be employed to efficiently compute 2-disjoint paths $(s_1, t_1), (s_2, t_2)$, e.g., using a construction similar to the one outlined in Figure 5. The problem with this idea is that s_1 may be matched to t_2 and s_2 to t_1 . Indeed, the problem of finding two disjoint paths from $\{s_1, s_2\}$ to $\{t_1, t_2\}$ where the matching is subject to optimization, is significantly simpler (and can be solved, e.g., using a flow algorithm).

B. Flow Size Changes Make it Hard

We have assumed so far that traffic rates are not changed at waypoints. However, there are many scenarios where waypoints increase or decrease the bandwidth demand. For example, the addition of an encapsulation header will increase the packet

sizes whereas a wide-area network optimizer may compress the packets. Unfortunately, it turns out that computing routes through a single waypoint which changes the bandwidth is much harder than routing through waypoints which do not:

Theorem 2: On undirected graphs in which waypoints are not flow-conserving, computing a route through a single waypoint is strongly NP-complete.

An NP-complete problem is strongly NP-complete, if the input can be restricted to numbers in unary representation.

Proof: Reduction from the strongly NP-complete 2-splittable flow problem: Given an undirected graph G with link capacities, are there two paths to route the flow from S^+ to T^+ s.t. the flow is maximized? Koch and Spenke showed in [17] that determining whether the maximum throughput is 2 or 3 in the 2-splittable flow problem is strongly NP-hard on undirected graphs with link capacities of 1 or 2.

Our reduction will be from the corresponding decision problem, i.e., does a flow of size 3 exist? Assume for ease of construction that $s := S^+ =: t$ and $w := T^+$. As all link capacities are either 1 or 2, we only need to check the variants $d_0 \in \{1, 2\}, d_1 \in \{1, 2\}$ of the capacitated waypoint routing problem for feasibility. Therefore, if there was a polynomial algorithm for the capacitated waypoint routing problem on undirected graphs, we would also obtain a polynomial algorithm for the initial problem. Lastly, the capacitated waypoint routing problem is clearly in NP. ■

C. Directions Are Challenging As Well

But not only waypoints changing the flow sizes turn the problem hard quickly: in a *directed* network, already the problem of finding a *feasible* waypoint route is NP-hard, even if waypoints are flow conserving.

Theorem 3: On directed graphs, the waypoint routing problem is strongly NP-complete for a single waypoint.

Proof: Our proof is by a reduction from the strongly NP-complete 2-link-disjoint paths problem [18]: Given two node pairs $(s_1, t_1), (s_2, t_2)$ in a directed graph $G = (V, E)$, are there two link-disjoint paths $P_1 = s_1, \dots, t_1, P_2 = s_2, \dots, t_2$?

We perform a reduction of all problem instances I of the 2-link-disjoint paths problem in graphs G to instances I' in graphs G' as follows: Create a (waypoint) node w , and add the directed links (t_1, w) and (w, s_2) , see Fig. 6.

To finish the construction of the waypoint routing problem in I' , set $s := s_1$ and $t := t_2$: Is there a route from s via w to t , using every link only once?

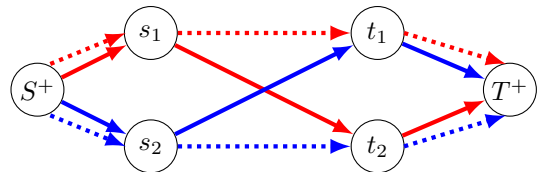


Fig. 5. Extending the idea of Figure 3 to two link-disjoint paths can fail, as shown in this figure: Instead of finding a $S^+, s_1, t_1, T^+, t_2, s_2, S^+$ path (depicted in dotted red and blue), the output could be to first visit t_2 and then t_1 second, never visiting t_2 again after (depicted in solid red and blue).

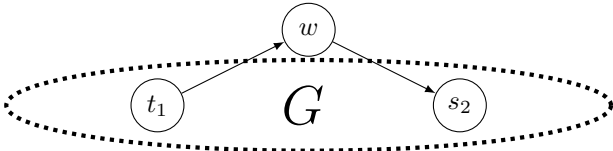


Fig. 6. By adding the waypoint w on a directed path between t_1 and s_2 , every feasible solution of the waypoint routing problem must be a concatenation of two walks s_1, \dots, t_1, w and w, s_2, \dots, t_2 .

If I is a yes-instance, I' is a yes-instance as well, by joining the paths P_1, P_2 via the directed links (t_1, w) and (w, s_2) . Next, we show that if I is a no-instance, I' is a no-instance as well: First, observe that to traverse w in G' starting from s , the only option is via traversing both links (t_1, w) and (w, s_2) , successively in that order. Thus, assume for the sake of contradiction that I' is a yes-instance with a link-disjoint walk $W = s, \dots, t_1, w, s_2, \dots, t$. Then, we can also create two link-disjoint walks $W_1 = s_1, \dots, t$ and $W_2 = s_2, \dots, t_2$ in I by removing both links (t_1, w) and (w, s_2) from W . Removing the loops in W_1 and W_2 results in paths P_1 and P_2 solving I , a contradiction.

To conclude, observe that the reduction can be performed in polynomial time, and as the problem is clearly in NP, the problem is NP-complete. ■

D. Another Complexity: Distance Constraints

Another problem variant arises if we do not only want to find a feasible (or shortest) path from s via w to t , but also have *hard constraints on the distance* or stretch from s to the waypoint, or from the waypoint to the destination.

Theorem 4: Finding a feasible path from s to t via w subject to distance constraints between two consecutive nodes from s, w, t is strongly NP-complete on undirected graphs.

Proof: This follows by reduction due to the hardness of finding 2 link-disjoint paths under a *min max objective*. Li et al. [19] showed that given a graph $G = (V, E)$ and two nodes s' and t' , the problem of finding two disjoint paths from s' to t' such that the length of the longer path is minimized is strongly NP-complete, even with unit link weights. This implies that the waypoint routing problem is strongly NP-complete as well, by setting $s = t = s'$ and $w = t'$. ■

Recall that the directed case of a single waypoint was already hard without distance constraints, see Theorem 3.

We note that Itai et al. [20] showed the two link-disjoint path problem with distance constraints to be NP-complete on directed acyclic graphs, using exponential link weights (polynomial in binary representation) in their construction. However, as we will show later, the distance constrained directed waypoint routing problem is polynomially time solvable on DAGs, even for arbitrarily many waypoints.

III. ROUTING VIA MULTIPLE WAYPOINTS

The advent of more complex network services requires the routing of traffic through *sequences of (multiple) waypoints*. Interestingly, and despite the numerous hardness results derived for a single waypoint in the previous section, we will see that

it is still possible to derive some polynomial-time algorithms even for multiple waypoints.

A. Possible For a Fixed Number of Waypoints...

Interestingly, the k -waypoint routing problem is tractable when the number of waypoints is constant:

Theorem 5: On undirected graphs, one can decide in polynomial time $O(m^2)$ whether a feasible route through a fixed number of waypoints exists.

Proof: The proof follows by application of [21], building upon the seminal work of Robertson and Seymour [12]: the authors show that for any fixed k , the k -link-disjoint path problem can be decided in polynomial-runtime of $O(n^2)$ on undirected graphs. We can apply their result by asking for link-disjoint paths connecting the waypoints in successive order. It only remains to set all link capacities to one: To do so, we divide the links into parallel links, their number bounded by $k \in O(1)$, even if the capacity is higher. Then, we place a node on every link, obtaining $O(n + km) \in O(m)$ nodes. ■

An analogous result holds for bidirected graphs [22].

B. ... Hard Already on Eulerian Graphs

While polynomial-time solutions exist for fixed k on general graphs, we now show that for general k , the problem is computationally intractable already on undirected Eulerian graphs (graphs on which routing problems are often simple), where all nodes have even degree.³

Theorem 6: The waypoint routing problem is strongly NP-complete on undirected Eulerian graphs.

Proof: We briefly introduce some notions of the problem that we will use for the reduction. The link-disjoint path problem can also be formulated via a supply graph $G = (V, E)$, which supplies the links to route the paths, and a demand graph $H = (V, E(H))$, whose links imply between which nodes there is a demand for a path. I.e., $\{(s_1, t_1), \dots, (s_k, t_k)\} = E(H)$. The union of both graphs is defined as $(V, E \cup E(H))$. As this notation is rather uncommon in a networking context, we provide a small introductory example in Figure 7.

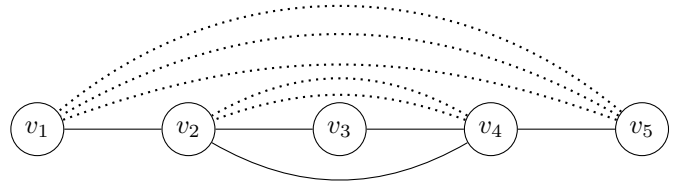


Fig. 7. Here, the supply graph G consists of $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_2, v_4)\}$, drawn in solid. The demand graph H has the same node set V , but its links $H(E)$ contain two links from v_2 to v_4 and three links from v_1 to v_5 , drawn dotted. Note that $(V, E \cup E(H))$ is planar and Eulerian. Still, the link-disjoint path problem given by the supply and demand graph is not solvable in this instance, e.g., only one path can be routed from v_1 to v_5 . If the demand graph did not contain any parallel links, both paths could be routed in a link-disjoint fashion.

We now reduce from the strongly NP-complete problem of finding link-disjoint paths where the union of the supply and

³ Beyond constant k , there is an $O(n^2)$ link-disjoint path algorithm on Eulerian graphs that allows $k \in O((\log \log \log n)^{\frac{1}{2} - \epsilon})$, for any $\epsilon > 0$ [23]. For planar Eulerian graphs, this also extends to $k \in O((\log n)^{\frac{1}{2} - \epsilon})$.

# Waypoints	Feasible Algorithms	Feasible Hardness	Demand Change Optimal Algorithms	Feasible Hardness
Arbitrary	P: Outerplanar ($\tau_w \leq 2$) Corollary 3	Strongly NPC: $\tau_w \leq 3$ Theorem 8	P: Tree (equivalent to τ_w of 1) Observation 2	NPC: Unicyclic ($\tau_w \leq 2$) Theorem 8
Constant	P: General graphs Theorem 5	P: General graphs Theorem 5	P: Constant treewidth $\tau_w \in O(1)$ Theorem 7	Strongly NPC: General graphs Theorem 2

TABLE II
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN SPECIAL UNDIRECTED GRAPHS.

the demand graph is Eulerian [24]. Our polynomial reduction construction of an instance I to an undirected graph $G' = (V', E')$ proceeds as follows: We first initialize $V' = V$ and $E' = E \cup E(H)$. Next, we add a new *center* node v to G' , containing s, t . For simplicity, we will assume that v also contains the $k - 1$ waypoints $w_4, w_8, w_{12}, \dots, w_{4(k-1)}$; those can also be moved to small cycles connected to v . Next, for $1 \leq i \leq k$, we define the remaining waypoints as follows: $w_{4i-3} = s_i, w_{4i-2} = t_i, w_{4i-1} = s_i$. We also add two links between v and each s_i to $E', 1 \leq i \leq k$. I.e., our waypoint problem is now an instance I' : Start in the center node v , go to s_1 , then to t_1 , back to s_1 , then to v ; then proceed similarly for $s_2 \dots$, to s_k , and ending at v . We note that new graph is still Eulerian. Again, in the same spirit as before, we can split the corresponding demand links, possibly twice, moving waypoints there, preserving the restriction of one waypoint per node and removing all parallel links.

We start with the easier case, showing that if I is a yes-instance, I' is as well: We can take the k disjoint paths of the solution of I in G , add the k (t_i, s_i) paths via the links of $H(E)$, and lastly connect the waypoint path-segments in order via the incident links of v .

It remains to show that if I' is a yes-instance, I is as well. First, if the k (t_i, s_i) paths use links outside of $E(H)$, we can alter the solution s.t. only the links of $E(H)$ are used for the k (t_i, s_i) paths: Assume for some i , that the link $(t_i, s_i) \in E(H)$ is not used for the path P from t_i to s_i , but is rather part of another walk W_j from some w_j to w_{j+1} (if the link is not used at all, the swap can be done directly). Then we can swap in W_j the link $(t_i, s_i) \in E(H)$ with P . Second, v has a degree of $2k$. Because s, t and $w_{4i-2} = t_i$ are pairwise non-consecutive waypoints, and no walk between any s_i, t_i or t_i, s_i can use the links incident to v , they must be used for all subwalks starting and ending at v . Thus, the subwalks between the k s_i and t_i (which can be simplified to paths) will now only use links already present in G . Lastly, observing that the problem is in NP finishes the proof. ■

A directed graph is called Eulerian, if for each node u holds: The in- and out-degree of u are identical. Marx also showed in [24] the (implicitly strong) NP-completeness of the directed case. Thus, we can apply analogous proof arguments.

Corollary 1: The ordered waypoint routing problem is strongly NP-complete on directed Eulerian graphs

For the case of bidirected graphs, which are a subset of directed Eulerian graphs, optimally solving the ordered waypoint routing problem is NP-complete [22], but the hardness of the feasibility variant is an open question.

IV. EXPLORING COMPUTATIONAL TRACTABILITY IN SPECIAL NETWORKS

Computer networks often have very specific structures: for example, many data centers are highly structured (e.g., based on Clos topologies [25]), but also enterprise and router-level AS topologies for example, while being less symmetric, often come with specific properties (e.g., are sparse). In this light, the results derived so far may be too conservative: in practice, much faster algorithms may be possible which are tailored toward and leverage the specific network structure. Accordingly, in this section we explore the waypoint routing problem on specific graph families. In particular, we are interested in sparse graphs. We conducted a small empirical study using Rocketfuel topologies [26] and Internet Topology Zoo graphs [27], and found that they often have a low path diversity: almost half of these graphs are *outerplanar*, and one third are *cactus graphs*:

- a graph is outerplanar if it has a planar drawing s.t. all vertices are on the outer face of the drawing [28]
- a graph is a cactus graph if any two simple cycles share at most one node [29] (every cactus graph is outerplanar)

A. Exact Polynomial-Time Algorithms

Tree networks. On tree networks, paths between two given nodes are unique, and finding shortest walks hence trivial: simply compute a shortest path for each path segment (recall: a simple path), one-by-one. If this walk is feasible, it is optimal; if not, no solution exists. Note that this also holds if waypoints change the flow rate, and for directed graphs, when the underlying undirected graph is a tree.

Observation 2: The shortest waypoint routing problem with demand changes can be solved in polynomial time on trees and DAGs.

DAGs. A similar results still holds on Directed Acyclic Graphs (DAGs). When making a choice for the path to the next waypoint, we can use a simple greedy algorithm: Any link that we use will never be used for a later path (due to the acyclic property). Hence, we can also minimize distance constraints for trees and DAGs.

In comparison, the link-disjoint path problem is polynomially solvable for a fixed number of link-disjoint paths on DAGs [18], but NP-complete in general already on planar DAGs [30].

Observation 3: There are graph families for which the waypoint routing problem can be solved efficiently while the disjoint paths problem cannot.

Regarding parametrized complexity: While an $n^{O(k)}$ algorithm exists for DAGs, the link-disjoint path problem on DAGs is W[1] hard [31], i.e., unlikely fixed-parameter tractable in k .

General Observations and Reductions. On the other hand, leveraging our connection to disjoint path problems again, we can also make the following observation:

Observation 4: For any graph family on which the $k + 1$ disjoint paths problem is polynomial-time solvable, we can also find a route through k waypoints in polynomial time on graphs of unit link capacity.

Thus, it immediately follows from [32] that the single waypoint routing problem is polynomial time solvable on semicomplete directed graphs, where a directed graph is called semicomplete, if there is at least one directed link between every pair of nodes.

Another case are directed graphs with constant independence number α , where $\alpha = \alpha(G)$ denotes the maximum size of an independent set in G . Then, for constant $\alpha, k \in O(1)$, a polynomial time algorithm exists [33].

Having a well-connected graph helps as well: On random undirected graphs G , where the set of $2k$ endpoints are chosen by an adversary (e.g., to compute a waypoint routing), it holds with high probability that the k paths exists, if $k \in O(n/\log n)$ and the minimum degree of G is some sufficiently large constant. The paths can be constructed in randomized time of $O(n^3)$ [34]. Similar results also hold on Expander graphs [35].

Bounded Treewidth Graphs I. For a further example, on bounded treewidth graphs, and as long as the number of waypoints k is logarithmically bounded, the problem is polynomial time solvable, because the link-disjoint paths problem is polynomial time solvable: For a treewidth decomposition of width $\leq \tau_w$ and k link-disjoint paths, Zhou et al. [36] provide an algorithm with a runtime of

$$O\left(n\left((k + \tau_w^2)k^{\tau_w(\tau_w+1)/2} + k(\tau_w + 4)^{2(\tau_w+4)k+3}\right)\right). \quad (1)$$

As a constant-factor approximation of treewidth decompositions can be obtained in polynomial time [37], also beyond constant treewidth, it is therefore possible to solve the waypoint routing problem for any values of t and k s.t. Equation (1) stays polynomial. E.g., $\tau_w, k \in O(\sqrt{\log n / \log \log n})$, due to $f(n)^{g(n)} = \exp(\ln(f(n)^{g(n)})) = \exp(g(n) \ln(f(n)))$. This idea can also be extended to polylogarithmic functions $f(n), g(n) \in \text{polylog}(n)$, obtaining quasi-polynomial runtimes of $2^{\text{polylog}(n)} \in \text{QP}$. Quasi-polynomial algorithms fit sort of in between polynomial and exponential algorithms and it is widely believed that NP-complete problems are not in QP [38].

Unit capacities can be modeled by introducing parallel links and in particular subdividing them by placing auxiliary nodes in the center, increasing the τ_w only by a constant factor.

We thus obtain the following corollary, which does not find shortest routes and is not applicable to demand changes:

Corollary 2: In undirected graphs with a treewidth of τ_w and k waypoints, we can solve the waypoint routing problem in polynomial time for the following combinations:

- Constant $\tau_w \in O(1)$, logarithmic $k \in O(\log n)$
- $\tau_w \in O(\sqrt{\log n})$, constant $k \in O(1)$
- $\tau_w, k \in O\left(\sqrt{\log n / \log \log n}\right)$.

In quasi-polynomial time, we can solve:

- $\tau_w, k \in \text{polylog}(n)$.

Nonetheless, note that the non-parallel unit capacity observation is of limited use in general: for a negative example, an outerplanar graph requires nodes to touch the outer face, however, this property will be lost during the graph transformation. Yet, as we will show in the following, solutions for outerplanar graph still exist, even in arbitrarily capacitated networks. We note that outerplanar graphs have a treewidth of $\tau_w \leq 2$.

Outerplanar Graphs. We first prove the following lemma.

Lemma 1: Let \mathcal{I} be a class of WRP s.t.

- 1) the graph G is planar (w.l.o.g. we have a planar drawing),
- 2) the maximum capacity is c_{\max} , w.l.o.g. $n \geq c_{\max} \in \mathbb{N}$,
- 3) s, t and all waypoints touch the outer face \mathcal{F} of G ,
- 4) for every node $v \notin \mathcal{F}$, $\sum_{e: \{u,v\} \in E(G)} c(e)$ is even.

Then the *feasibility* of the ordered waypoint routing problem in the class \mathcal{I} is decidable in time $O(n^2)$, with the explicit construction taking $O(m^2 \cdot c_{\max}^2)$.

Proof: Let $I \in \mathcal{I}$ be an instance of the problem. Suppose s, t are the source and terminal and w_1, \dots, w_k are waypoints. Define $w_0 = s, w_{k+1} = t$. We construct an equivalent instance of the link-disjoint paths problem as follows. Replace each link $e = \{u, v\}$ with capacity c by $c \leq c_{\max}$ links with capacity 1, then subdivide those links once, i.e., the number of nodes is in $O(m \cdot c_{\max})$. In the newly created instance of link-disjoint paths problem:

- 1) The input graph is planar,
- 2) all terminal pairs touch the outerface,
- 3) degree of every node, not in the outerface, is even.

If only condition 1) and 2) hold, the problem is NP-hard [39]. But for this class of link-disjoint paths problem, there are polynomial time algorithms [40] with the following properties: Let b be the number of nodes on the outer face and n' be the total number of nodes. The feasibility of the link-disjoint path problem can be tested in $O(bn')$ and constructing the paths can be done in $O(n'^2)$ which gives us the desired polynomial time solutions for the original problem. ■

This directly implies the following result.

Corollary 3: In outerplanar graphs with a maximum link capacity of c_{\max} , the waypoint routing problem is decidable in time $O(n^2)$, with an explicit construction obtainable in time $O(m^2 \cdot \min\{n^2, c_{\max}^2\})$.

A solution to the shortest waypoint routing problem cannot be obtained via the same reduction: Brandes et al. [41] showed the minimum total length link-disjoint path problem to be NP-hard on graphs satisfying the three conditions mentioned above, already when the maximum degree is at most 4.

For bidirected cactus graphs of constant capacity, the ordered waypoint routing problem can be optimally solved in polynomial time [22].

Bounded Treewidth Graphs II. Let us quickly recap the results on bounded treewidth τ_w found so far:

- 1) For constant τ_w , we can compute walks for $k \in O(\log n)$ waypoints, but those walks will not be optimal (shortest) and the flow has to be of unit size. The same holds for outerplanar graphs (a class with $\tau_w = 2$) for $k \in O(n)$.

2) For $\tau_w = 1$ (\equiv trees), we can compute shortest walks with demand changes, even for $k \in O(n)$.

As pointed out in the beginning of this section, many network topologies have low treewidth, especially in the wide-area and enterprise context (e.g., the Rocketfuel and Topology Zoo networks [26]). We now tackle a problem we thus deem to be realistic: in practice, the number of waypoints visited by a given flow is likely to be a small constant.

Theorem 7: In undirected graphs with bounded treewidth $\tau_w \in O(1)$ and a fixed number $k \in O(1)$ of waypoints, we can solve the shortest waypoint routing problem with demand changes in a runtime of $O(n)$.

Proof: Our proof will be via dynamic programming of a nice tree decomposition [42] $\mathcal{T} = (T, X)$ of G as follows: Using the ideas and terminology of Kloks [43], each bag of \mathcal{T} is either a *leaf bag*, a *forget bag* (one node is removed from the separator), an *introduce bag* (a node is added), or a *join bag* (its two children q_1, q_2 contain the same nodes).

For bags b , we thus define signatures σ_b , representing already computed solutions of b , such that by dynamically programming \mathcal{T} bottom-up, we obtain an optimal walk \mathcal{W} at the root bag of \mathcal{T} , if such a \mathcal{W} exists.

In every optimal solution \mathcal{W} , each path from a w_i to a w_{i+1} will cross each separator b of G at most τ_w times. Due to optimality, these individual paths will traverse every node at most once. Hence, a signature σ_b only needs to represent the at most $k \cdot \tau_w$ crossings (endpoints) of partial paths through the subgraph of b , and the link utilizations these paths use in $E(b)$. We additionally store if a path, for from w_i to w_{i+1} , with only one endpoint in the signature, contains either w_i or w_{i+1} . Note that at most one such path each will exist at any time due to optimality. Due to $k, \tau_w \in O(1)$, we have only $O(1)$ different possible signatures for each bag b , with each signature containing only $O(1)$ elements. As common, we assume that we can perform standard operations (additions, comparisons etc.) of numerical values in constant time, else, an extra logarithmic factor needs to be included in the total runtime. We now present the required algorithms for the induction.

- **Leaf bags b :** In constant time, we can generate all valid signatures, containing at most k paths (each without any links). The only restriction is that if $v \in V(b)$ is a waypoint w_i , its paths to w_{i-1} and w_{i+1} must exist.
- **Forget bags b :** Let v be the node s.t. for the child q of b holds: $V(q) \setminus \{v\} = V(b)$. If v is not a waypoint, then the valid signatures of b are exactly those of q which do not use v as endpoints. If v is a waypoint w_i , then additionally must hold: v must be an endpoint of a path from w_{i-1} and the endpoint of a path to w_{i+1} .
- **Join bags b :** We first 1) describe the program and then 2) prove its correctness. 1): Given two valid signatures of b 's children q_1, q_2 , we perform all possible concatenations, of endpoints of paths for the same w_i to w_{i+1} , at the separator nodes $V(b)$, checking a) that the union of the link utilizations in $E(b)$ respect the link capacities and b) that no loops are created (we know the endpoints of each (sub-)path and the their link utilizations in $E(b)$, if they

share a link outside $E(b)$, a signature of minimum size will not), which results in valid signatures σ_b of b . 2): Assume we missed some valid signature σ_b of b : Given σ_b , we split the paths across the separator, resulting in valid signatures $\sigma_{q_1}, \sigma_{q_2}$ and their subpaths, a contradiction. For an illustration of this procedure, we refer to Figure 8.

- **Introduce bags b :** Again, we first 1) describe the algorithm and then 2) prove its correctness. 1): For each signature σ_q of the child q of b , where $V(q) \cup \{v\} = V(b)$, we first generate all possible combinations of empty paths at v . Then, we distribute the link set of $E(b)$ over the endpoints in all possible variations, checking if each distribution can generate some valid signature by possibly moving the endpoints of the subwalks (and possibly, concatenating some). If the answer is yes, we also generate all possible signatures out of these distributions, again by allowing to move the endpoints and allowing to concatenate paths, always respecting capacity constraints. As we only handle $O(1)$ elements, we only perform $O(1)$ operations (covered below). 2): Again, assume we did not program some valid signature σ_b of b . We then obtain a valid signature of q by removing v , splitting all paths that traverse it into two, or, if they have v as an endpoint, cutting off v , or, if the path only contained v , by removing these paths. As the reverse operation will be performed by the prior algorithm, σ_b would have been obtained.

Each of the above programs be be run in a time of $O(1)$, assuming constant size $b, \tau_w, k \in O(1)$.

Furthermore, we implicitly assumed that for each signature, we also store a representative set of paths s.t. their total length is minimized. I.e., when generating signatures multiple times for introduce and join nodes, we only keep representatives of minimum total length. Hence, after dynamically programming the nice tree decomposition \mathcal{T} bottom-up, we consider all solutions at the root node: If an optimal solution exists, it will be represented by a signature, and thus, we can choose a walk through the waypoints of minimum length.

It remains to prove the desired runtime of $O(n)$: For constant treewidth $\tau_w \in O(1)$, we can obtain a nice tree decomposition of width $O(\tau_w)$ with $O(n)$ bags in a runtime of $O(n)$ using the methods from [37], [43]. As the dynamic program requires time $O(1)$ for each of the $O(n)$ bags, and as each of the $O(1)$

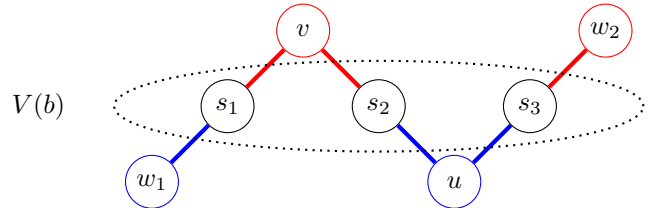


Fig. 8. In this example, the separator is shown in the middle, containing the nodes $V(b) = V(q_1) = V(q_2) = \{s_1, s_2, s_3\}$. By splitting the path from w_1 to w_2 along the separator, we obtain multiple paths per side, their number being bounded by the size of the separator. Observe that when two sub-paths, between the same set of waypoints, share a node, this node must be an endpoint for both; otherwise, minimality is violated.

possible solutions can be checked in time $O(n)$, the claim follows. ■

B. Hardness

Let us return to the general problem where there can be an arbitrary number of waypoints. We have shown that for a large graph family of treewidth at most 2, the outerplanar graphs (which also include cactus graphs for example), the routing paths can be computed efficiently. This raises the question whether the problem can be solved also on graphs of treewidth larger than 2, or at least for *all* graphs of treewidth at most 2. While the latter remains an open question, in the following we show that problems on graphs of treewidth 3 (namely series-parallel graphs with an additional node connected to all other nodes) are already NP-hard in general.

Theorem 8: The problem of routing through an arbitrary number of waypoints is strongly NP-complete on graphs of treewidth at most 3.

Proof: We reduce the ordered waypoint routing problem in graphs of treewidth at most 3 from the link-disjoint paths problem in series-parallel graphs, the latter being strongly NP-complete [44].

Let I be an instance of the link-disjoint paths problem in a series parallel graph G with terminal pairs $T_I = \{(s_1, t_1), \dots, (s_k, t_k)\}$. We construct a new instance \mathcal{I} of the ordered waypoint problem as follows. Create a graph $G' := G$, then add one new node v to G' and links $\{t_i, v\}, \{s_j, v\}$ for $i, j \in [k], j \neq 1, i \neq k$.

For simplicity, set for now $s := s_1, w_1 := t_1, w_2 = v, w_3 := s_2, w_4 := t_2, w_4 := v, \dots, t := t_k$, i.e., the order of waypoints is $s_1, t_1, v, \dots, v, s_i, t_i, v, s_{i+1}, t_{i+1}, v, \dots, t_k$, with $3k - 2$ waypoints in total. I.e., v “hosts” $k - 1$ waypoints, with a degree of $2(k - 1)$. We will show later in the proof how to ensure at most one waypoint per node.

Claim: In any solution for \mathcal{I} , the union of the $k - 1$ link-disjoint walks from s_i via v to t_{i+1} occupy all links incident to v .

Proof: Any walk from s_i via v to t_{i+1} must leave and enter v , using two links. Hence, the union of all these $k - 1$ link-disjoint walks occupy all $2k - 2$ links incident to v . □

We can now prove the theorem: If I is a yes-instance, then \mathcal{I} is a yes-instance as well: We take the k s_i, t_i -paths from I , connect them in index-order with the $k - 1$ paths t_i, v, s_{i+1} , and obtain the desired ordered waypoint routing.

It is left to show that if \mathcal{I} is a yes-instance, then I is a yes-instance as well: Let \mathcal{I} be a yes-instance. Define the path from s_i to t_i as in \mathcal{I} . As these paths do not use v or any of the edges adjacent to it (otherwise the capacity of one of these edges would be exceeded), these paths show that I is a yes-instance.

On the other hand, the treewidth of G' is at most the treewidth of G plus 1 (we can just put v in all bags of an optimal tree decomposition of G). To obtain at most one waypoint on v , we create $k - 1$ cycles of length four, placing a waypoint on each, and merging another node with v . This construction does not increase the treewidth and also retains earlier proof arguments. As series-parallel graphs have a treewidth of at

most 2 [45, Lemma 11.2.1], G' has a treewidth of at most 3. As the problem is clearly in NP, with the reduction being polynomial, the proof is complete. ■

We conjecture that it is possible to directly modify the proof presented in [44], to prove that the feasibility of the waypoint routing problem is hard even in series-parallel graphs.

One cycle is hard. In case of non-flow conserving waypoints, NP-hardness strikes earlier already, namely on unicyclic graphs, which contain only one cycle, and as thus have $\tau_w \leq 2$.

Theorem 9: On undirected unicyclic graphs in which waypoints are not flow-conserving, computing a route through $O(n)$ waypoints is weakly NP-complete, even if all waypoints can just increase (or, just decrease) the flow size by at most a constant factor.

Proof: Reduction from the weakly NP-complete PARTITION problem [46], where an instance I contains ℓ non-negative integers i_1, \dots, i_ℓ , $\sum_{j=1}^{\ell} i_j = S$, with the size of the binary representation of all integers polynomially bounded in ℓ .

We begin with the case that waypoints can change the flow size arbitrarily. W.l.o.g., let ℓ be even and $i_1 \leq i_2 \leq \dots \leq i_\ell$. We create two stars (denoted left and right star) with $1 + \ell/2$ leaf nodes each, where all links have a capacity of S . We connect both star center nodes in a cycle, with the cycle links having a capacity of $S/2$ each, respectively.

Next, we place s , here also identified as w_1 , on a leaf of the left star and t on a leaf in the right star. To distribute the remaining $\ell - 1$ waypoints w_2, \dots, w_ℓ , corresponding to the integers, we place the ones with even indices on leaves in the left star, and those with odd indices in the right star.

Suppose the routing starts with a size of i_1 , is changed to i_2 by w_2 and so on. Then, solving the PARTITION instance I is equivalent to computing a waypoint routing, as the paths going along the cycle have to be partitioned into two sets, each having a combined demand of $S/2$.

So far, we assumed that waypoints can change the flow size arbitrarily – but hardness also holds if each waypoint can just increase (or, just decrease) the flow size by a constant amount. In order to do so, we replace the leaf nodes of the stars with paths of $O(\log S)$ waypoints, which are used to increase the demands to the desired size. ■

The directed graph case is analogous by putting all waypoints to one star, creating the same amount of intermediate dummy waypoints in the other star, which do not change the flow size, and replacing all undirected links with two directed links of opposite directions and identical capacity.

Corollary 4: On directed graphs, with the underlying undirected graph being unicyclic and where waypoints are not flow-conserving, computing a route through $O(n)$ waypoints is NP-complete, even if all waypoints can just increase (or, just decrease) the flow size by at most a constant factor.

For these two proofs, we used flow sizes that can be exponential in the graph size (binary encoded). Nonetheless, recall Theorems 2 and 3, where we showed that the problem also stays strongly NP-complete on general graphs.

V. OTHER RELATED WORK

In this paper, we focus on the allocation of a *single* walk, without violating capacity constraints. However, there also exists literature on how to *admit* and allocate *multiple* walks, e.g., using randomized rounding and tolerating some capacity augmentation [47], [48], [49]; there are also extensions to more complex requests such as trees [49], [50].

Moreover, while we focused on walks through *ordered* waypoints, there is work on routing through *unordered* waypoints [22], [51]. The problem of finding shortest (link- and node-disjoint) paths and cycles through a *set* of k waypoints has been a central topic of graph theory for several decades [52]. A cycle from s through $k = 1$ waypoints back to $t = s$ can be found efficiently by breadth first search, for $k = 2$ the problem corresponds to finding a integer flow of size 2 between two nodes, and for $k = 3$, it can still be solved in linear time [53], [18]; a polynomial-time solution for any constant k follows from the work on the disjoint path problem [12]. The best known deterministic algorithm to compute *feasible* (but not necessarily shortest) paths is by Kawarabayashi [54]: it finds a cycle for up to $k = O((\log \log n)^{1/10})$ waypoints in deterministic polynomial time. Björklund et al. [9] presented a randomized algorithm based on algebraic techniques which finds a shortest simple cycle through a given set of k node or links in an n -node undirected graph in time $2^k n^{O(1)}$. However, there is no obvious way to apply algorithms designed for the unordered problem variant to the ordered problem variant.

VI. CONCLUSION

This paper initiated the study of a fundamental algorithmic problem underlying modern network services: the routing via waypoints using walks. We hope that our paper can provide the network community with algorithmic techniques but also inform about complexity bounds. While we present a comprehensive set of algorithms and hardness results, there are several interesting directions for future research, for example regarding randomized algorithms or algorithms for scenarios where capacity constraints may be violated slightly.

Acknowledgments. We like to thank Thore Husfeldt for inspiring discussions. Research partly supported by the Villum project ReNet as well as by Aalborg University's PreLytics project. Saeed Amiri's research was partly supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No 648527).

REFERENCES

- [1] J. Sherry et al., "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, 2012.
- [2] ETSI, "Network functions virtualisation – introductory white paper," *White Paper*, oct 2013.
- [3] R. Soulé et al., "Merlin: A language for provisioning network resources," in *Proc. ACM CoNEXT*, 2014.
- [4] P. S. et al., "Towards unified programmability of cloud and carrier infrastructure," in *Proc. EWSDN*, 2014.
- [5] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," in *Proc. SIGCOMM*, 2015.
- [6] A. Björklund and T. Husfeldt, "Shortest two disjoint paths in polynomial time," in *Proc. ICALP*, 2014.
- [7] M. Cygan, D. Marx, M. Pilipczuk, and M. Pilipczuk, "The planar directed k -vertex-disjoint paths problem is fixed-parameter tractable," in *Proc. FOCS*, 2013.
- [8] P. D. Seymour, "Disjoint paths in graphs," *Discrete Mathematics*, vol. 29, no. 3, pp. 293–309, 1980.
- [9] A. Björklund, T. Husfeldt, and N. Taslaman, "Shortest cycle through specified elements," in *Proc. SODA*, 2012.
- [10] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *Proc. USENIX ATC*, 2014.
- [11] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *Proc. GLOBECOM*, 2015.
- [12] N. Robertson and P. D. Seymour, "Graph Minors .XIII. The Disjoint Paths Problem," *J. Comb. Theory, Ser. B*, vol. 63, no. 1, pp. 65–110, 1995.
- [13] B. Korte and J. Vygen, *Combinatorial optimization*. Springer, 2012.
- [14] J. W. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, no. 2, pp. 125–145, 1974.
- [15] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.
- [16] G. Naves and A. Sebö, "Multiflow feasibility: An annotated tableau," in *Research Trends in Combinatorial Optimization*, W. J. Cook, L. Lovász, and J. Vygen, Eds. Springer, 2008, pp. 261–283.
- [17] R. Koch and I. Spenke, "Complexity and approximability of k -splittable flows," *Theoretical Computer Science*, vol. 369, no. 1, pp. 338 – 347, 2006.
- [18] S. Fortune, J. E. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Theor. Comput. Sci.*, vol. 10, pp. 111–121, 1980.
- [19] C.-L. Li, S. T. McCormick, and D. Simchi-Levi, "The complexity of finding two disjoint paths with min-max objective function," *Discrete Applied Mathematics*, vol. 26, no. 1, pp. 105–115, 1990.
- [20] A. Itai, Y. Perl, and Y. Shiloach, "The complexity of finding maximum disjoint paths with length constraints," *Networks*, vol. 12, no. 3, pp. 277–286, 1982.
- [21] K. Kawarabayashi, Y. Kobayashi, and B. A. Reed, "The disjoint paths problem in quadratic time," *J. Comb. Theory, Ser. B*, vol. 102, no. 2, pp. 424–435, 2012.
- [22] K.-T. Foerster, M. Parham, and S. Schmid, "A walk in the clouds: Routing through vnfs on bidirected networks," in *Proc. ALGOCLOUD*, 2017.
- [23] K. Kawarabayashi and Y. Kobayashi, "The edge-disjoint paths problem in eulerian graphs and 4-edge-connected graphs," *Combinatorica*, vol. 35, no. 4, pp. 477–495, 2015.
- [24] D. Marx, "Eulerian disjoint paths problem in grid graphs is NP-complete," *Discrete Applied Mathematics*, vol. 143, no. 1-3, pp. 336–341, 2004.
- [25] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 63–74.
- [26] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.
- [27] S. Knight, H. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 9, pp. 1765 –1775, october 2011.
- [28] G. Chartrand and F. Harary, "Planar permutation graphs," *Annales de l'I.H.P. Probabilites et statistiques*, vol. 3, no. 4, pp. 433–438, 1967.
- [29] D. Geller and B. Manvel, "Reconstruction of cacti," *Canad. J. Math*, vol. 21, pp. 1354–1360, 1969.
- [30] J. Vygen, "NP-completeness of some edge-disjoint paths problems," *Discrete Applied Mathematics*, vol. 61, no. 1, pp. 83–90, 1995.
- [31] A. Slivkins, "Parameterized tractability of edge-disjoint paths on directed acyclic graphs," *SIAM J. Discrete Math.*, vol. 24, no. 1, pp. 146–157, 2010.
- [32] J. Bang-Jensen, "Edge-disjoint in- and out-branchings in tournaments and related path problems," *J. Comb. Theory, Ser. B*, vol. 51, no. 1, pp. 1–23, 1991.
- [33] A. O. Fradkin and P. D. Seymour, "Edge-disjoint paths in digraphs with bounded independence number," *J. Comb. Theory, Ser. B*, vol. 110, pp. 19–46, 2015.
- [34] A. M. Frieze and L. Zhao, "Optimal construction of edge-disjoint paths in random regular graphs," *Combinatorics, Probability & Computing*, vol. 9, no. 3, pp. 241–263, 2000.

- [35] A. M. Frieze, “Edge-disjoint paths in expander graphs,” *SIAM J. Comput.*, vol. 30, no. 6, pp. 1790–1801, 2000.
- [36] X. Zhou, S. Tamura, and T. Nishizeki, “Finding edge-disjoint paths in partial k -trees,” *Algorithmica*, vol. 26, no. 1, pp. 3–30, 2000.
- [37] H. L. Bodlaender, P. G. Drange, M. S. Dregi, F. V. Fomin, D. Lokshtanov, and M. Pilipczuk, “An approximation algorithm for treewidth,” in *Proc. FOCS*, 2013.
- [38] G. J. Woeginger, “Exact algorithms for np-hard problems: A survey,” in *Combinatorial Optimization*, ser. LNCS, vol. 2570. Springer, 2001, pp. 185–208.
- [39] W. Schwärzler, “On the complexity of the planar edge-disjoint paths problem with terminals on the outer boundary,” *Combinatorica*, vol. 29, no. 1, pp. 121–126, 2009.
- [40] M. Becker and K. Mehlhorn, “Algorithms for routing in planar graphs,” *Acta Informatica*, vol. 23, no. 2, pp. 163–176, 1986.
- [41] U. Brandes, G. Neyer, and D. Wagner, “Edge-disjoint paths in planar graphs with short total length,” 1996, Konstanzer Schriften in Mathematik und Informatik; 19.
- [42] H. Bodlaender, “Dynamic programming on graphs with bounded treewidth,” *Automata, Languages and Programming*, pp. 105–118, 1988.
- [43] T. Kloks, *Treewidth, Computations and Approximations*, ser. Lecture Notes in Computer Science. Springer, 1994, vol. 842.
- [44] T. Nishizeki, J. Vygen, and X. Zhou, “The edge-disjoint paths problem is NP-complete for series-parallel graphs,” *Discrete Appl. Math.*, vol. 115, 2001.
- [45] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A Survey*. Philadelphia, PA, USA: SIAM, 1999.
- [46] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [47] G. Even, M. Medina, and B. Patt-Shamir, “Online path computation and function placement in sdns,” in *Proc. SSS*, 2016.
- [48] T. Lukovszki and S. Schmid, “Online admission control and embedding of service chains,” in *SIROCCO*, 2015.
- [49] G. Even, M. Rost, and S. Schmid, “An approximation algorithm for path computation and function placement in SDNs,” in *SIROCCO*, 2016.
- [50] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer, “Minimum congestion mapping in a cloud,” in *Proc. ACM PODC*, 2011.
- [51] S. Akhoondian Amiri, K.-T. Foerster, and S. Schmid, “Walking Through Waypoints,” *arXiv preprint arXiv:1708.09827*, 2017.
- [52] L. Perković and B. A. Reed, “An improved algorithm for finding tree decompositions of small width,” *Int. J. Found. Comput. Sci.*, vol. 11, no. 3, pp. 365–371, 2000.
- [53] H. Fleischner and G. J. Woeginger, “Detecting cycles through three fixed vertices in a graph,” *Inf. Process. Lett.*, vol. 42, no. 1, pp. 29–33, 1992.
- [54] K. Kawarabayashi, “An improved algorithm for finding cycles through elements,” in *Proc. IPCO*, 2008.