Charting the Algorithmic Complexity of Waypoint Routing

A Guided Walk

Saeed Akhoondian Amiri[1], Klaus-Tycho Foerster[2], Riko Jacob[3], Stefan Schmid[4]
[1] MPI Saarland, Germany, [2] Aalborg University, Denmark,
[3] IT University of Copenhagen, Denmark, [4] University of Vienna, Austria

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, less is known about the underlying algorithmic traffic routing problems. The goal of this paper is to provide the network community with an overview of algorithmic techniques for waypoint routing and also inform about limitations due to computational hardness. In particular, we put the waypoint routing problem into perspective with respect to classic graph theoretical problems. For example, we find that while computing a shortest path from a source s to a destination t is simple (e.g., using Dijkstra's algorithm), the problem of finding a shortest route from s to t via a single waypoint already features a deep combinatorial structure.

CCS Concepts

•**Networks** → **Routing protocols;**

Keywords

Waypoints, VNFs, Algorithms, Complexity

1. INTRODUCTION

1.1 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 [14]): 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 [6], by *composing* network functions to *service chains* [12, 24]. 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* $(w_1, \ldots, w_k)$? 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 identify algorithmic techniques to solve the different variants of the waypoint routing problem, as well as to explore limitations due to intractability.

1.2 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 $(w_1, \ldots, w_k)$:** which need to be traversed along the way from s to t, forming a route $(s, w_1, \ldots, w_k, t)$. 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 $d_0 = d_1 = \ldots = d_k = 1$.

In general, one is 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 routes: is it possible to route the flow without violating link capacities at all (a **decision problem**)?

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

Table 1: Overview of the Complexity Landscape for Waypoint Routing in General Graphs.

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.

1.3 The Twist: 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 [4, 5, 27] and the k-cycle problem [3]. 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.

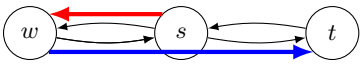

Figure 1: A route (s, w, t) in the depicted network must contain a loop. The only solution is the walk s, w, s, t, resulting from concatenating the red (s, w) and blue (w, t) paths. It can hence not be described as a simple path.

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*.

1.4 Our Contributions

This paper initiates the algorithmic study of the waypoint routing problem underlying many modern networking applications. Waypoint routing is becoming increasingly relevant in the context of modern networking services and applications, such as service chaining [24] (where traffic needs to be steered through network functions), hybrid SDNs [18] (where traffic is steered through OpenFlow switches) or in segment routing [9] (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 reductions *from* resp. *to* classic combinatorial problems, and also derive several new algorithms from scratch which may be of interest beyond our scope.

In summary, we make the following observations. For a single waypoint ($k = 1$) we find that:

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 [4], a route via a single waypoint can in fact be computed very efficiently.
2. **Waypoints which change the flow size are challenging:** We find that routing through a single waypoint is NP-hard in general if the waypoint changes the flow.
3. **Directed links make it hard as well:** While there are fast algorithms for undirected networks, the waypoint routing problem is 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 [25].
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.

An overview of our complexity results shown in this paper can be found in Table 1. 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.

1.5 Paper Organization

We study routing problems via a single waypoint in Sec. 2 and via multiple waypoints in Sec. 3. We cover further related work in Sec. 4, and conclude in Sec. 5.

2. 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.

2.1 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 hard-

ness 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 observe the following.

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

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. 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.

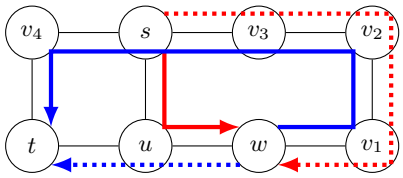

Figure 2: In undirected graphs, path segments need to be jointly optimized: greedily selecting a shortest path from s to w can force a very long path from w to t. Once the solid red (s, w) path has been inserted first as a shortest path, there is only one option for the solid blue (w, s) path, resulting 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 $4+2=6$.

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 [4] 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.

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.

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

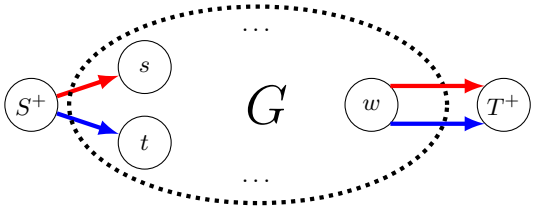

Figure 3: By adding a super-source S^+, we can reduce the waypoint routing problem on undirected graphs to a min cost flow problem: As a $w-t$ flow is also a $t-w$ flow, we can check if there is a flow of size 2 from S^+ to w. An analogous idea can be used to reduce the waypoint routing problem to finding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

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. [17, p. 227].

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 [28]: 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 [29]:

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 [29] in our proof, which solves 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 [23], see Fig. 4, adjusting weights accordingly.

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^+

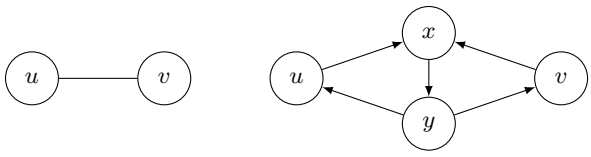

Figure 4: By replacing every undirected link with the construction to the right, we can apply algorithms for directed graphs to undirected graphs. Observe that in both cases, the integer flow possibilities between u and v are identical.

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).

2.2 Flow Size Changes Make it Hard

There are scenarios where waypoints increase or decrease the bandwidth demand, e.g., the addition of an encapsulation header will increase the packet sizes whereas a wide-area network optimizer may compress the packets.

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

PROOF. Reduction from the 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 [16] that determining whether the maximum throughput is 2 or 3 in the 2-splittable flow problem is 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. □

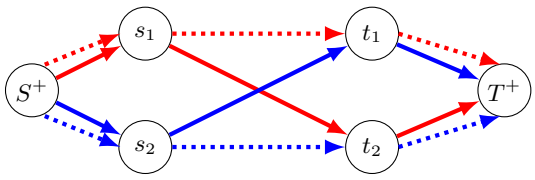

Figure 5: Extending the idea of Figure 3 to two link-disjoint paths can fail, as shown in this figure: Instead of finding a $S^+, s_1, t_1, T^+, t_2, s_2, S^+$ path (depicted in dotted red and blue), the output could be to first visit t_2 and then t_1 second, never visiting t_2 again after (depicted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

2.3 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 NP-complete for a single waypoint.*

PROOF. Our proof is by a reduction from the NP-complete 2-link-disjoint paths problem [11]: Given 2 node pairs (s_1, t_1), (s_2, t_2) in a directed graph $G = (V, E)$, are there two link-disjoint paths $P_1 = s_1, \ldots, t_1$, $P_2 = s_2, \ldots, t_2$?

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?

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 $W = s, \ldots, t_1, w, s_2 \ldots, t$. Then, we can also create two link-disjoint walks $W_1 = s_1, \ldots, t$ and $W_2 = s_2, \ldots, t_2$ 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. □

2.4 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 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

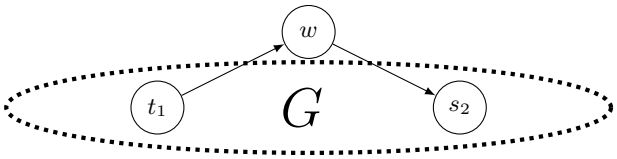

Figure 6: By adding the waypoint w on a directed path between t_1 and s_2, every feasible solution of the waypoint routing problem must be a concatenation of two walks $s_1, \ldots, t_1, w$ and $w, s_2, \ldots, t_2$.

minimized is NP-complete, even with unit link weights. This implies that the waypoint routing problem is 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. For comparison, note that in the absence of waypoints, upper bounds on the route length are easy to maintain, by running a shortest path algorithm on all links with sufficient capacity.

We note that Itai et al. [13] showed the two link-disjoint path problem with distance constaints to be NP-complete on directed acyclic graphs, using exponential link weights (polynomial in binary representation) in their construction. However, as we will see in Sec. 3.3, the distance constrained directed waypoint routing problem is polynomially time solvable on DAGs, even for arbitrarily many waypoints.

3. 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.

Note that the uncapacitated case can be optimally solved by existing shortest path algorithms such as Dijkstra's algorithm or all-pairs-shortest-path algorithms.

3.1 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 [15], building upon the seminal work of Robertson and Seymour [25]: 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 a graph with $O(n + km) \in O(m)$ nodes. □

3.2 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.

THEOREM 6. *The waypoint routing problem is NP-complete on undirected Eulerian graphs.*

PROOF. We briefly introduce some notations of the problem that we will use for the reduction, illustrated in Figure 7. 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., $\{(s_1, t_1), \ldots, (s_k, t_k)\} = E(H)$. The union of both graphs is defined as $(V, E \cup E(H))$.

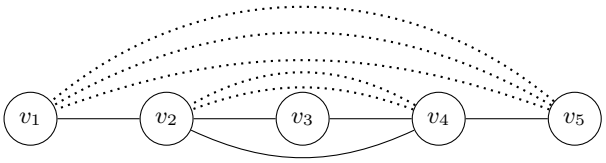

Figure 7: In this example, the supply graph G consists of all nodes V and the links drawn in solid. The demand graph H has the same node set V, but its links $H(E)$ contain two links from v_2 to v_4 and three links from v_1 to v_5, drawn dotted. Note that $(V, E \cup E(H)$ is planar and Eulerian. Still, the link-disjoint path problem given by G, H is not solvable in this instance, e.g., only one path can be routed from v_1 to v_5 via the solid links. 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 NP-complete problem of finding link-disjoint paths where the union of the supply and the demand graph is Eulerian [22]. 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 $w_4, w_8, w_{12}, \ldots, w_{4(k-1)}$; 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', $1 \leq i \leq k$. E.g., in Figure 7, we add six links to v_1 and four links to v_2.

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 $s_2 \ldots$, 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 Eulerian property, the restriction of one waypoint per node, and removing all parallel links. NP-completeness now follows directly via case distinctions. □

3.3 Possible on Trees and DAGs

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 is polynomial 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 DAGs (and, trivially on trees). In comparison, the link-disjoint path problem is polynomialy solvable for a fixed number of link-disjoint paths on DAGs [11], 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.*

4. OTHER RELATED WORK

In this paper, we focus on the allocation of a *single* walk, without violating capacity constraints. Our work hence differs from existing literature on approximation algorithms for (admitting and) allocating multiple walks which allow for (sometimes significant) capacity violation, e.g., based on randomized rounding [7, 8, 21, 26]. Existing literature on approximation algorithms usually also considers the placement of waypoints [20] and sometimes also considers more general requests, which are not limited to paths through waypoints but which may for example also come in the form of trees [2, 8, 26]. Moreover, while we in this work focused on walks through *ordered* waypoints, there is work on routing through *unordered* waypoints [1] and on symmetric digraphs [10]. There is no obvious way to apply algorithms designed for the unordered problem variant to the ordered case.

5. CONCLUSION

We hope that our paper can provide the network community with algorithmic techniques but also inform about complexity bounds. In future research, it would be interesting to study a generalization to practically relevant scenarios in which waypoint placement is also subject to optimization or in which waypoints may be replicated for load balancing purposes. Moreover, more research is required on how to extend existing efficient traffic engineering heuristics to account for waypoint requirements.

Acknowledgments. We thank Thore Husfeldt for inputs. Research supported by the Villum project ReNet and Aalborg University's PreLytics project. Saeed Amiri's research was partly supported by the European Research Council (ERC) grant agreement No 648527.

6. REFERENCES

[1] S. Akhoondian Amiri, K.-T. Foerster, and S. Schmid. Walking Through Waypoints. In *Proc. LATIN*, 2018.

[2] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer. Minimum congestion mapping in a cloud. In *Proc. ACM PODC*, 2011.

[3] A. Björklund, T. Husfeld, and N. Taslaman. Shortest cycle through specified elements. In *Proc. SODA*, 2012.

[4] A. Björklund and T. Husfeldt. Shortest two disjoint paths in polynomial time. In *Proc. ICALP*, 2014.

[5] 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.

[6] ETSI. Network functions virtualisation – introductory white paper. *White Paper*, oct 2013.

[7] G. Even, M. Medina, and B. Patt-Shamir. Online path computation and function placement in sdns. In *Proc. SSS*, 2016.

[8] G. Even, M. Rost, and S. Schmid. An approximation algorithm for path computation and function placement in SDNs. In *SIROCCO*, 2016.

[9] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois. The segment routing architecture. In *Proc. GLOBECOM*, 2015.

[10] K.-T. Foerster, M. Parham, and S. Schmid. A walk in the clouds: Routing through vnfs on bidirected networks. In *Proc. ALGOCLOUD*, 2017.

[11] S. Fortune, J. E. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

[12] 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.

[13] A. Itai, Y. Perl, and Y. Shiloach. The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12(3):277–286, 1982.

[14] J. Sherry et al. Making middleboxes someone else's problem: Network processing as a cloud service. In *Proc. ACM SIGCOMM*, 2012.

[15] K. Kawarabayashi, Y. Kobayashi, and B. A. Reed. The disjoint paths problem in quadratic time. *J. Comb. Theory, Ser. B*, 102(2):424–435, 2012.

[16] R. Koch and I. Spenke. Complexity and approximability of k-splittable flows. *Theoretical Computer Science*, 369(1):338 – 347, 2006.

[17] B. Korte and J. Vygen. *Combinatorial optimization*. Springer, 2012.

[18] 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.

[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*, 26(1):105–115, 1990.

[20] T. Lukovszki, M. Rost, and S. Schmid. It's a match! near-optimal and incremental middlebox deployment. *ACM SIGCOMM Computer Communication Review (CCR)*, 46(1):30–36, 2016.

[21] T. Lukovszki and S. Schmid. Online admission control and embedding of service chains. In *SIROCCO*, 2015.

[22] D. Marx. Eulerian disjoint paths problem in grid graphs is NP-complete. *Discrete Applied Mathematics*, 143(1-3):336–341, 2004.

[23] G. Naves and A. Sebö. Multiflow feasibility: An annotated tableau. In W. J. Cook, L. Lovász, and J. Vygen, editors, *Research Trends in Combinatorial Optimization*, pages 261–283. Springer, 2008.

[24] R. Soulé et al. Merlin: A language for provisioning network resources. In *Proc. ACM CoNEXT*, 2014.

[25] N. Robertson and P. D. Seymour. Graph Minors .XIII. The Disjoint Paths Problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[26] M. Rost and S. Schmid. Service chain and virtual network embeddings: Approximations using randomized rounding. *arXiv preprint*, 2016.

[27] P. D. Seymour. Disjoint paths in graphs. *Discrete Mathematics*, 29(3):293–309, 1980.

[28] J. W. Suurballe. Disjoint paths in a network. *Networks*, 4(2):125–145, 1974.

[29] J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.

[30] J. Vygen. NP-completeness of some edge-disjoint paths problems. *Discrete Applied Mathematics*, 1995.