Waypoint Routing in Special Networks

Saeed Akhoondian Amiri[1] Klaus-Tycho Foerster[2] Riko Jacob[3] Mahmoud Parham[2] Stefan Schmid[2]

[1] MPI Saarland, Germany [2] University of Vienna, Austria [3] IT University of Copenhagen, Denmark

***Abstract*—Waypoint routing is a novel communication model in which traffic is steered through one or multiple so-called waypoints along the route from source to destination. Waypoint routing is used to implement more complex policies or to compose novel network services such as service chains, and also finds applications in emerging segment routing networks. This paper initiates the study of algorithms and complexity of waypoint routing on special networks. Our main contribution is an encompassing characterization of networks on which routes through an arbitrary number of waypoints can be computed efficiently: We present an algorithm to compute waypoint routes for the important family of outerplanar networks, which have a treewidth of at most two. 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 (namely graphs of treewidth three). For the case that the number of waypoints is constant, we also provide a polynomial-time algorithm for any constant treewidth network, even if waypoints change the flow sizes. For arbitrary numbers of waypoints however, the constraint of different flow-sizes between waypoints turns the problem hard, already if the network contains just a single cycle. Finally, we extend the study of waypoint routing to special directed graph classes, in particular bidirected graphs.**

I. Introduction

Waypoint routing is a fundamental communication model in which packets need to visit a sequence of waypoints along their route. Waypoint routing has many applications, e.g., related to security policies [1], [2], [3], [4], emerging network services such as service function chaining [5], [6], [7], [8], [9], or segment routing [10], [11], [12], [13].

For example, computer networks today consist of a large number of so-called middleboxes (in the order of the number of routers [1]) providing various functionality inside the networks, related to security (e.g., firewalls, NATs) and performance (e.g., proxies, traffic optimizers). In order to benefit from (or enforce) these middleboxes, traffic needs to be steered through the functions ("waypoints") explicitly, as in Fig. 1. This is non-trivial especially in virtualized environments and in the context of Network Function Virtualization (NFV), where virtualized middleboxes can be deployed more flexibly. Software-Defined Networking (SDN) is a particularly useful technology in this context, as it facilitates the definition of such more flexible routes.

This paper is concerned with the algorithmic aspects underlying waypoint routing. Interestingly, only little is known today about the algorithmic problems, besides that the problem is typically hard on general network topologies [14].

Our paper is motivated by the fact that real-world networks (e.g., datacenter, enterprise, carrier networks) are often not general or "worst-case" but feature additional structure, which can potentially be exploited toward more efficient algorithms. Accordingly, we initiate in this paper the study of waypoint routing on specific network topologies.

A. Our Contributions

This paper studies the problem of computing (shortest) paths through an arbitrary number of waypoints on special network families. Our main contribution is a, in some sense, tight characterization of the network topologies on which routes through waypoints can be computed in polynomial time. Concretely, we provide an algorithm to compute waypoint routes on the important graph family of outerplanar graphs (which are of treewidth at most two). 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). We also provide a polynomial algorithm for shortest routes on any constant treewidth, as long as the number of waypoints is also constant, with the added feature that the flow-sizes may change after each waypoint traversal. Additionally, we present various algorithmic and complexity results on special directed graphs, in particular on special bidirected graphs such as so-called cactus topologies.

B. State-of-the-Art and Novelty

The recent article by Amiri et al. [14] provided a first chart for this waypoint routing problem in general graph classes. Their focus is on providing intractability results and methods for few waypoints, but they present no algorithms to handle an arbitrary number of waypoints beyond trees and DAGs.

The goal of this paper is to chart the algorithmic landscape of *special graph classes*, motivated by often highly structured computer networks. Our main results on undirected graphs are presented in Table I, but we also provide further new insights

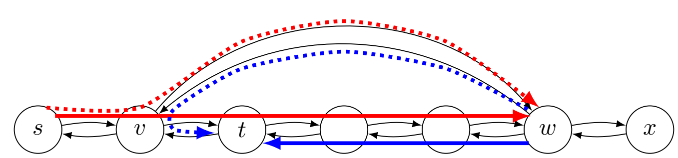

Fig. 1. In this introductory example, the task is to route the flow of traffic from the source s to the destination t via the waypoint w. When routing via the solid red (s, w) path, followed by the solid blue (w, t) path, the combined walk length is $5 + 3 = 8$. A shorter solution exists via the dotted red and blue paths, resulting in a combined walk length of $2 + 2 = 4$. Observe that when the waypoint would be on the node x, no node-disjoint path can route from s to t via the waypoint. Furthermore, some combinations can violate unit capacity constraints, e.g., combining the solid red with the dotted blue path induces a double utilization of the link from v to t.

# Waypoints	Feasible Algorithms	Known Hardness	Demand Change Optimal Algorithms	Demand Change Hardness
Arbitrary	**P**: Outerplanar ($\texttt{tw} \leq 2$) Corollary 2	**Strongly NPC**: $\texttt{tw} \leq 3$ Theorem 4	**P**: Tree (equivalent to $\texttt{tw}$ of 1) [14]	**NPC**: Unicyclic ($\texttt{tw} \leq 2$) Theorem 4
Constant	**P**: General graphs [14]	**P**: General graphs [14]	**P**: Constant treewidth $\texttt{tw} \in O(1)$ Theorem 3	**Strongly NPC**: General graphs [14]

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

w.r.t. algorithms for special directed graphs, whereas [14] only provided NP-hardness results on general directed graphs.

C. Organization

The remainder of this paper is organized as follows. Section II introduces the problem and model more formally, along with studying the example of a single waypoint. Our in-depth algorithmic results are presented in Section III, whereas the complementing intractability proofs can be found in Section IV. We present further related work in Section V and conclude in Section VI.

II. THE PROBLEM AND MODEL

We study computer networks, modeled as connected undirected, directed, or bidirected [15] graphs $G = (V, E)$ with $|V| = n$ nodes (switches, middleboxes, routers) and $|E| = m$ links, where each link $e \in E$ has a capacity $c : E \to \mathbb{N}_{>0}$ and a weight (cost) $\chi : E \to \mathbb{N}_{>0}$. Bidirected graphs (also known as, e.g., Asynchronous Transfer Mode (ATM) networks [16] or symmetric digraphs [17]) are directed graphs with the property that if a link $e = (u, v)$ exists, there is also an anti-parallel link $e' = (v, u)$ with $c(e) = c(e')$ and $\chi(e) = \chi(e')$.

Given (1) a (bi/un)directed graph, (2) a source $s \in V$ and a destination $t \in V$, and (3) a set of k waypoints in V, the *waypoint routing problem* asks for a flow-route R (i.e., a walk) from s to t that (i) visits all waypoints in $\mathscr{W}$ and (ii) respects all link capacities. Without loss of generality, we normalize link capacities to the size of the traffic flow, removing links of insufficient capacity. 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, where the demand can be denoted as follows: 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 $d_0 = d_1 = \ldots = d_k = 1$, and refer to this scenario as *flow-conserving*.

The waypoints depend on each other and must be traversed in a pre-determined order: every waypoint w_i may be visited at any time in the walk, and as often as desired (while respecting link capacities), but the route R must contain a given ordered node sequence $s, w_1, w_2, \ldots, w_k, t$. For example, in a network with stringent dependability requirements, it makes sense to first route a packet through a fast firewall before performing a deeper (and more costly) packet inspection.

We are interested both in *feasible* solutions (respecting capacity constraints) as well as in *optimal* solutions. In the context of the latter, we aim to optimize the cost $|R|$ of the route R, i.e., we want to minimize the sum of the weights of all traversed links.

Lastly, for ease of reference, we might denote the undirected waypoint routing problem by WRP, the directed version by DWRP, and the bidirected version by BWRP.

Before directly presenting our algorithms and complexity results, we start with a warm-up, considering the case of a single waypoint in bidirected networks.

A. An Introductory Case Study: A Single Waypoint

We first examine the case of a *single waypoint* w, which requires finding a shortest $s - t$ route through this waypoint. Amiri et al. [14] already $1)$ provided a polynomial-time algorithm for undirected graphs and $2)$ showed the NP-hardness for directed graphs. We thus complement their results by providing an algorithm for bidirected graphs as an introduction.

One waypoint: greedy is optimal. Simply taking two *shortest paths* (SPs) $P_1 = SP(s, w)$ and $P_2 = SP(w, t)$ in a greedy fashion is sufficient, i.e., the route $R = P_1 P_2$ is always feasible (and thus, also always optimal in regards to total weight).

Suppose this is not the case, that is, $P_1 \cap P_2 \neq \emptyset$, possibly violating capacity constraints. Among all nodes in $P_1 \cap P_2$, let u and v be, resp., the first and the last nodes w.r.t. to the order of visits in R. Let P_i^{xy} denote the sub-path connecting x to y in P_i. Thereby we have $R = P_1 P_2 = P_1^{su} P_1^{uv} P_1^{vw} P_2^{wu} P_2^{uv} P_2^{vt}$ (Fig. 2). Let $\bar{\mathcal{P}}$ be the reverse of any walk $\mathcal{P}$ obtained by replacing each link $(x, y) \in \mathcal{P}$ with its anti-parallel link (y, x). Observe that for $P_1' = P_1^{su} \bar{P}_2^{wu}$ and $P_2' = \bar{P}_1^{vw} P_2^{vt}$ we have that P_1' is at most as long as P_1 (because P_1 is shortest) and P_2' is shorter than P_2 (by P^{uv}), a contradiction to P_2 being a shortest path.

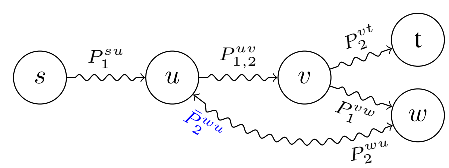

Fig. 2. The directed path from u to v is traversed two times in R.

Two waypoints: can be infeasible! While we saw that it is always possible to route through a single waypoint in bidirected graphs, already two waypoints can prevent a valid solution.

In the example of Figure 3, an $s - t$ route traversing first w_1 and w_2 second must use the link from w_2 to w_1 twice. Hence, the feasibility of a solution depends on the link capacity.

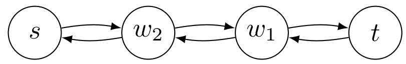

Fig. 3. In this unit capacity network, the task is to route the flow of traffic from s to w_1, then to w_2, and lastly to t. To this end, the link from w_2 to w_1 must be used twice.

After this brief introduction, we next study algorithms and complexity beyond the simple case of a single waypoint.

III. Exploring Computational Tractability

Computer networks often have very specific structures: for example, many data centers are highly structured (e.g., are based on Clos topologies [18]), 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 general hardness results provided in [14] may be too pessimistic: in practice, much faster algorithms may be possible which are tailored toward and leverage the specific network structure. For example, as already pointed out in [14], the waypoint routing problem can be solved quickly on undirected tree or DAG topologies

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 [19] and Internet Topology Zoo graphs [20], 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 [21]
- a graph is a cactus graph if any two simple cycles share at most one node [22] (every cactus graph is outerplanar)

A. General Observations and Reductions

As first pointed out in [14] for undirected graphs, there is a direct algorithmic connection from the link-disjoint path problem to BWRP with unit capacities. By setting $s_1 = s$, $t_1 = w_1$, $s_2 = w_1$, $t_2 = w_2, \ldots$, a $k+1$ link-disjoint path algorithm also solves unit capacity BWRP for k waypoints. This method can be extended to general capacities via a standard technique, by replacing each link of capacity $c(e)$ with $\lfloor c(e) \rfloor$ parallel links of unit capacity and identical weight.

Hence, we can apply the algorithm from Jarry and Prennes [17], which solves the feasibility of the link-disjoint path problem on bidirected unit capacity multigraphs for a constant number of paths in polynomial runtime.

Theorem 1: Let $k \in \mathcal{O}(1)$. Feasible solutions for BWRP can be computed in polynomial time.

The optimal solution already for few link-disjoint paths still puzzles researchers on bidirected graphs, but the problem seems to be non-trivial on undirected graphs as well: while feasibility for a constant number of link-disjoint paths is polynomial in the undirected case as well [23],[24], optimal algorithms for 3 or more link-disjoint paths are not known, and even for 2 paths the best result is a recent randomized high-order polynomial-time algorithm [25]. For directed graphs, already 2 link-disjoint paths pose an NP-hard problem [26].

Furthermore, leveraging our connection to disjoint path problems again, we can also make the following observation, which we will use for special directed graphs and a non-constant amount of waypoints on some undirected graphs.

Observation 1: 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 [27] 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 DWRP algorithm exists, using [28].

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 $O(n^3)$ [29]. Similar results also hold on Expander graphs [30].

B. Algorithms: Parametrized by Treewidth I/II

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.

We briefly introduce the notion of treewidth as in [31], with alternate analogous descriptions and further examples provided in, e.g., Bodlaender and Kloks in [32], [33], [34]: Given an undirected graph $G = (V, E)$, a *tree decomposition* $\mathcal{T} = (T, X)$ of G is a bijection between a collection X and a tree T, s.t. every element of X is a set of nodes from V with: 1) each graph node is contained in at least one tree node, which is in turn called a *bag* (separator), 2) the tree nodes containing a node v form a connected subtree of T, and 3) nodes are adjacent in the graph only when the corresponding subtrees have a node in common. The *width* of $\mathcal{T} = (T, X)$ is the number of elements in the largest set in X minus 1. The *treewidth* $\texttt{tw}$ is the minimum width over all tree decompositions of G. We will make use of these definitions again in Section III-D.

For a treewidth decomposition of width $\leq \texttt{tw}$ and k link-disjoint paths, Zhou et al. [35] provide an algorithm with a runtime of

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

As a constant-factor approximation of treewidth decompositions can be obtained in polynomial time [36], 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., $\texttt{tw}, k \in O(\sqrt{\log n / \log\log n})$, 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 [37].

Unit capacities can be modeled by introducing parallel links and in particular subdividing them by placing auxiliary nodes in the center. For each such new path of length three, we can add the three nodes of the path to a new bag, and connect it to the original bag. Unless the graph is a tree (in which case the treewidth increases by one), the treewidth remains unchanged.

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

Corollary 1: In undirected graphs with a treewidth of $\texttt{tw}$ and k waypoints, we can solve the waypoint routing problem in polynomial time for the following combinations:

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

In quasi-polynomial time, we can solve:

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

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 exist, even in arbitrarily capacitated networks. We note that outerplanar graphs have a treewidth of $\texttt{tw} \leq 2$.

C. Algorithms: Outerplanar and Cactus Graphs

Undirected Outerplanar Graphs. We first prove the following lemma, which we then use for outerplanar graphs.

Lemma 1: Let $\mathcal{I}$ be the class of undirected WRP with

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 $v \notin \mathcal{F}$, $\Sigma_{e:\,\{u,v\} \in E(G)} c(e)$ is even.

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

Proof: Let $I \in \mathcal{I}$ be an instance of the problem. Suppose s, t are the source and terminal and $w_1, \ldots, w_k$ 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 outer face,
3) the degree of every node not on the outer face is even.

If only condition 1) and 2) hold, the problem is NP-hard [38]. But for this class of link-disjoint paths problems, there are polynomial time algorithms [39] with the following properties: Let b be the number of nodes on the outer face and n' be the total number of nodes. Because the graph is planar we have $m = O(n)$ and $n' = O(n)$. 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 2: In undirected 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 $O\left(n^2 \cdot \min\left\{n^2, c_{\max}^2\right\}\right)$.

A solution to the shortest waypoint routing problem cannot be obtained via the same reduction: Brandes et al. [40] 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, as we show next.

Bidirected Cactus Graphs The difficulty of BWRP lies in the fact that the routing from w_i to w_{i+1} can be done along multiple paths, each of which could congest other waypoint connections. Hence, it is easy to solve BWRP optimally (or check for infeasibility) on trees, as each path connecting two successive waypoints is unique.

Lemma 2: BWRP can be solved optimally in polynomial time on trees.

For multiple path options, the problem turns NP-hard though (Theorem 6). To understand the impact of already two options, we follow-up by studying rings.

Lemma 3: BWRP is optimally solvable in polynomial time on bidirected ring graphs where for at least one link e holds: $c(e) \in \mathcal{O}(1)$.

Proof: We begin our proof with $c(e) = c(e') = 1$. Observe that every routing between two successive waypoints has two path options P, clockwise or counter-clockwise. We assign one arbitrary path P_e to traverse e, and another arbitrary path $P_{e'}$ to traverse e'. By removing the fully utilized e and e', the remaining graph is a tree with two leaves, where all routing is fixed, cf. Lemma 2.

We now count the path assignment possibilities for e, e': by also counting the "empty assignment", we have at most $(n+1)n$ options, where the optimal routing immediately follows for each option. For these $\mathcal{O}(n^2)$ possibilities, we pick the shortest feasible one. I.e., BWRP can be solved optimally in polynomial time on rings with unit capacity. To extend the proof to constant capacities $c(e) \in \mathcal{O}(1)$, we use an analogous argument, the number of options for assignments to e and e' are now $\mathcal{O}\left(n^{2c(e)}\right) \in \text{P}$. Thus, the lemma statement holds. ■

We now focus on the important case of cactus networks. As mentioned earlier, our empirical study using the Internet

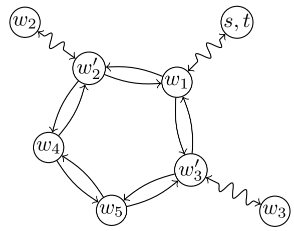

Fig. 4. In this cactus graph, we illustrate the algorithm of Theorem 2 w.r.t. the permutation $w_1w_2w_3w_4w_5$.

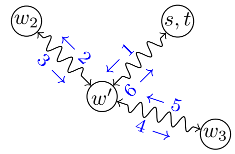

Fig. 5. Once the ring links are contracted, w' replaces the whole ring. Consequently, the permutation reduces to $w'w_2w_3$. The sub-routes are numbered sequentially.

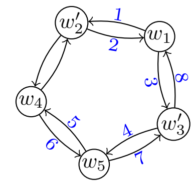

Fig. 6. The permutation induced on the ring is $w_1w_2'w_3'w_4w_5$. In the sub-problem, we have $s = t = w_1$. The numbers represent the order of node traversal in the optimal route.

Topology Zoo[1] data set shows that one third are *cactus graphs*.

Theorem 2: BWRP is optimally solvable in polynomial time on cactus graphs with constant capacity.

Proof: The idea is to 1) shrink the cactus graph down to a tree, 2) see if for the relevant subset of waypoints (to be described shortly) the feasibility holds on that tree, 3) reincorporate the excluded rings and find the optimal choice of path segments within each ring, and 4) construct an optimal route by stitching together the sub-routes obtained from the tree and the segments from each ring.

Let $\mathcal{C}$ be the cactus graph (Fig. 4) and $T_\mathcal{C}$ be the tree obtained after contracting all the links on each rings. As a result of this link contraction, those waypoints previously residing on rings are now replaced by new (super) waypoints in $T_\mathcal{C}$ (Fig. 5). Each super node represents either a subtree of adjacent rings or just an isolated ring. Let $\mathscr{W}'$ denote the waypoints in $T_\mathcal{C}$. Observe that any feasible route in $\mathcal{C}$ through $\mathscr{W}$ corresponds to one unique feasible route in $T_\mathcal{C}$ through nodes in $\mathscr{W}'$. Next, we show that either the feasible route in $T_\mathcal{C}$ (if exists) can be expanded to an optimal route for $\mathcal{C}$, or there is no feasible route in $\mathcal{C}$ at all. If $T_\mathcal{C}$ is not feasible then we are done. Otherwise, let R be the (unique) route in this tree. For each ring, R induces some *endpoints* (Fig. 6), one endpoint on each node that is either a) the joint of $T_\mathcal{C}$ and the ring, or b) the joint with its adjacent rings. Now we focus on the subproblem induced by this ring and the new waypoint set $\mathscr{W}''$ (to be specified) as follows.

For each endpoint that is visited by R add a waypoint to $\mathscr{W}''$. Then, using the algorithm described in the proof of Lemma 3, find an optimal route R_{ring} visiting all the nodes in $\mathscr{W}''$ respecting the order imposed by R. If no such route exists, the instance is not feasible. Otherwise, remove from R every occurrence of the super node that represents this ring to get a disconnected route. For each missing part, reconnect the endpoints using the segment of R_{ring} restricted to these endpoints. Repeat this for every ring; denote the resulting route as R'.

Finally, we argue that R' is optimal. This is the case because its pieces were taken from sets of sub-routes, where each set, covers a disjoint–or more precisely, node-disjoint up to endpoints–component of $\mathcal{C}$. Moreover, the set of sub-routes taken from an individual (disjoint) component (i.e. tree or ring) is optimal on that component. Therefore the total length is optimal. ■

We next turn our attention to graphs of constant treewidth.

D. Algorithms: Parametrized by Treewidth II/II

Let us quickly recap the results on undirected graphs of bounded treewidth $\texttt{tw}$ found so far:

1) For constant $\texttt{tw}$, 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 $\texttt{tw} = 2$) for $k \in O(n)$.
2) For $\texttt{tw} = 1$ ($\equiv$ trees), one can compute shortest walks with demand changes, even for $k \in O(n)$ [14].

As pointed out in the beginning of Section III, many network topologies have low treewidth, especially in the wide-area and enterprise context (e.g., the Rocketfuel and Topology Zoo networks [19]). 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 3: In undirected graphs with bounded treewidth $\texttt{tw} \in O(1)$ 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* [41] $\mathcal{T} = (T, X)$ of G. Using the ideas and terminology of Kloks [34], a tree decomposition is nice if 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 $\texttt{tw}$ 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 $k \cdot \texttt{tw}$ 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 $k, \texttt{tw} \in O(1)$, we have

[1] See http://www.topology-zoo.org/.

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 7.
- **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.

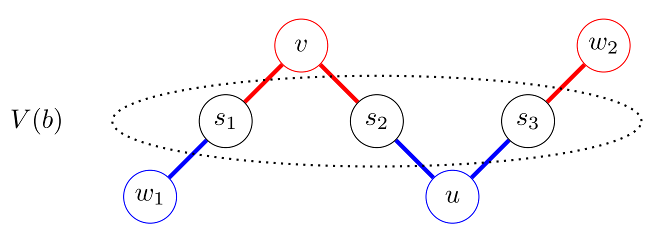

Fig. 7. In this example, the separator is shown in the middle, containing the nodes $V(b) = V(q_1) = V(q_2) = \{s_1, s_2, s_3\}$. By splitting the path from w_1 to w_2 along the separator, we obtain multiple paths per side, their number being bounded by the size of the separator. Observe that when two sub-paths, between the same set of waypoints, share a node, this node must be an endpoint for both; otherwise, minimality is violated.

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

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 $\texttt{tw} \in O(1)$, we can obtain a nice tree decomposition of width $O(\texttt{tw})$ with $O(n)$ bags in a runtime of $O(n)$ using the methods from [34], [36]. As the dynamic program requires time $O(1)$ for each of the $O(n)$ bags, and as each of the $O(1)$ possible solutions can be checked in time $O(n)$, the claim follows. ■

IV. Hardness

In the previous Section III we presented various polynomial-time algorithms for undirected and directed graphs. In this section we present complementing hardness results, to clarify the corresponding intractability bounds. In comparison, previous work [14] provided NP-hardness results for general graphs, leaving the finer details where the border lays between polynomial-time algorithms and intractability to future work.

We begin by studying the treewidth of undirected graphs in Section IV-A, followed by the NP-hardness on (un)directed unicyclic graphs under flow-size changes in Section IV-B. Lastly, we investigate general bidirected graphs in Section IV-C, where hardness already strikes without flow-size changes, as in Section IV-A on undirected graphs.

A. Hardness: Parametrized by Treewidth

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 on undirected graphs. 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 4: The problem of routing through an arbitrary number of waypoints is strongly NP-complete on undirected 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 [42].

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 $s := s_1$, $w_1 := t_1$, $w_2 := v$, $w_3 := s_2$, $w_4 := t_2$, $w_5 := v$, $\dots$, $t := t_k$, 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 links adjacent to it (otherwise the capacity of one of these links 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 [43, 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 [42], to prove that the feasibility of the waypoint routing problem is hard even in series-parallel graphs.

B. Hardness: Flow-size changes and a single cycle

In case of non-flow conserving waypoints, NP-hardness strikes earlier already, namely on unicyclic graphs, which contain only one cycle, and thus have $\texttt{tw} \leq 2$.

Theorem 5: 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 [44], where an instance I contains ℓ non-negative integers $i_1,\dots,i_\ell$, $\sum_{j=1}^{\ell} i_j = S$, 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 $w_2,\dots,w_\ell$, 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 3: 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, we refer to Table II, which shows that the problem also stays strongly NP-complete on general graphs.

C. Hardness: Bidirected graphs without flow-size changes

It follows from the earlier Corollary 3 that waypoint routing is already NP-hard on unicyclic bidirected graphs, when allowing flow-size changes. It remains to study NP-hardness in the case that the flow-size remains unchanged:

Theorem 6: Solving BWRP optimally is NP-hard.

Proof: Reduction from the NP-hard link-disjoint path problem on bidirected graphs $G = (V,E)$ [16]: given k source-destination node-pairs (s_i,t_i), $1 \leq i \leq k$, are there k corresponding pairwise link-disjoint paths?

For every such instance I, we create an instance I' of BWRP as follows, with all unit capacities: Set $s = s_1$ and $t = t_k$,

<table>
<tr><th></th><th># Waypoints</th><th>Feasible</th><th>Optimal</th><th>Demand Change Feasible</th><th>Optimal</th></tr>
<tr><td rowspan="3">Undirected</td><td>1</td><td colspan="2">P</td><td colspan="2" rowspan="2">Strongly NPC</td></tr>
<tr><td>constant</td><td>P</td><td>?</td></tr>
<tr><td>arbitrary</td><td colspan="4">Strongly NPC</td></tr>
<tr><td rowspan="3">Directed</td><td>1</td><td colspan="4" rowspan="3">Strongly NPC</td></tr>
<tr><td>constant</td></tr>
<tr><td>arbitrary</td></tr>
</table>

TABLE II
OVERVIEW OF THE COMPLEXITY LANDSCAPE FOR WAYPOINT ROUTING IN GENERAL GRAPHS AS PROVIDED BY [14].

also setting waypoints as follows: $w_1 = t_1$, $w_3 = s_2$, $w_4 = t_2$, $w_6 = s_3$, $w_7 = t_3$, $\ldots$, $w_{3k-3} = s_k$. We also create the missing $k-1$ waypoints w_2, w_5, w_8, $\ldots$, w_{3k-4} as new nodes and connect them as follows, each time with bidirected links of weight γ: w_2 to $w_1 = t_1$ and $w_3 = s_2$, w_5 to $w_4 = t_2$ and $w_6 = s_3$, $\ldots$, w_{3k-4} to $w_{3k-3} = s_k$ and $w_{3k-5} = t_{k-1}$. I.e., we sequentially connect the end- and start-points of the paths.

Observe that BWRP is feasible on I' if I is feasible: We take the k link-disjoint paths from I and connect them via the $k-1$ new nodes in I'.

We now set γ to some arbitrarily high weight, e.g., $3k$ times the sum of all link weights. I.e., it is cheaper to traverse every link of I even $3k$ times rather than paying γ once. Thus, if I is feasible, the optimal solution of I' has a cost of less than $2 \cdot k \cdot \gamma$.

Assume I is not feasible, but that I' has a feasible solution R. Observe that a feasible solution of I' needs to traverse the $k-1$ new waypoints, i.e., has at least a cost of $2(k-1)\gamma$. As I was not feasible, we will now show that traversing every new waypoint $w_2, w_5, \ldots$ only once is not sufficient for a feasible solution of I'. Assume for contradiction that one traversal of $w_2, w_5, \ldots$ suffices: for each of those traversals of such a w_j, it holds that it must take place after traversing all waypoints with index smaller than j. Hence, we can show by induction that the removal of the links incident to the waypoints $w_2, w_5, \ldots$ from R contains a feasible solution for I. Thus, at least one of the waypoints $w_2, w_5, \ldots$ must be traversed twice, i.e., R has a cost of at least $2 \cdot k \cdot \gamma$.

We can now complete the polynomial reduction, by studying the cost (feasibility) of an optimal solution of I': if the cost is less than $2 \cdot k \cdot \gamma$, I is feasible, but if the cost is at least $2 \cdot k \cdot \gamma$ (or infeasible), I is not feasible. ■

While many BWRP instances are not feasible (already in Figure 3), we conjecture that the feasibility of BWRP with arbitrarily many waypoints is NP-hard as well. This conjecture is supported by the fact that the analogous link-disjoint feasibility problems are NP-hard on undirected [44], directed [26], and bidirected graphs [16], also for undirected and directed ordered waypoint routing, see Table II.

V. RELATED WORK

While waypoint routing has recently received much attention in the literature, especially in the context of service function chaining [7], [9], [45], [46], we are not aware of any systematic study of the underlying algorithmic problem besides [14] which however does not consider special network families. We provide Table II for an overview of their results on general graphs.

In particular, our work is different from existing literature on the computation of routes through *unordered* waypoints [31]: the computation of shortest (link- and node-disjoint) paths and cycles through a *set* of k waypoints is a classic problem [47] which has traditionally been motivated by many different applications. Well-known results include, e.g., linear-time algorithms for $k = 3$ waypoints [26], [48] polynomial-time algorithms for constant k [24], polynomial-time deterministic algorithms to compute *feasible* paths for small $k = O((\log \log n)^{1/10})$, or a randomized algorithm (based on algebraic techniques) to compute a shortest simple cycle through a given set of k nodes or links in an n-node undirected network. These approaches however cannot be applied to compute routes through ordered waypoints.

Our work is also different from existing work which focuses on how to *admit* and allocate *multiple* walks, e.g., using randomized rounding and tolerating some capacity augmentation [49], [50], [51]. There are also extensions to more complex requests such as trees [51], [52]. In contrast, we in this paper focus on the allocation of a *single* walk, without violating capacity constraints.

Bibliographic Note. A first version of the results on bidirected graphs was presented at the Algocloud workshop [53].

VI. CONCLUSION

Waypoint routing is emerging as an important concept in various applications, however, the underlying algorithmic problem is not well-understood. With this paper, we have made a first step to put the waypoint routing problem into perspective. We presented a comprehensive characterization of the algorithmic complexity of the problem regarding the "special" network families which support a polynomial-time solution. In particular, we presented algorithms and hardness results for networks of different treewidth, and discussed implications of more directed networks. In our future work, we aim to investigate the implications of waypoint routing on specific applications, in particular, Traffic Engineering.

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 Union's

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] A. Ludwig, S. Dudycz, M. Rost, and S. Schmid, "Transiently secure network updates," in *Proc. ACM SIGMETRICS*, 2016.
[3] A. Matos, S. Sargento, and R. L. Aguiar, "Waypoint routing: A network layer privacy framework," in *Proc. IEEE GLOBECOM*, 2011.
[4] S. Ghorbani and B. Godfrey, "Towards correct network virtualization," in *Proc. SIGCOMM HotSDN*, 2014, pp. 109–114.
[5] P. Skoldstrom et al., "Towards unified programmability of cloud and carrier infrastructure," in *Proc. EWSDN*, 2014.
[6] ETSI GS NFV-IFA 003, "Network functions virtualisation (nfv); acceleration technologies; vswitch benchmarking and acceleration specification," in *Group Specification*, 2016.
[7] R. Soulé et al., "Merlin: A language for provisioning network resources," in *Proc. ACM CoNEXT*, 2014.
[8] P. S. et al., "Towards unified programmability of cloud and carrier infrastructure," in *Proc. EWSDN*, 2014.
[9] 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.
[10] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *Proc. GLOBECOM*, 2015.
[11] R. Bhatia, F. Hao, M. Kodialam, and T. Lakshman, "Optimized network traffic engineering using segment routing," in *Proc. IEEE INFOCOM*, 2015, pp. 657–665.
[12] C. Filsfils et al., "Segment routing architecture," in *Internet draft*, 2014.
[13] C. Filsfils, P. Francois, S. Previdi, B. Decraene, S. Litkowski, M. Horneffer, I. Milojevic, R. Shakir, S. Ytti, W. Henderickx, J. Tantsura, S. Kini, and E. Crabbe, "Segment routing architecture," in *Segment Routing Use Cases, IETF Internet-Draft*, 2014.
[14] S. Akhoondian Amiri, K.-T. Foerster, R. Jacob, and S. Schmid, "Charting the Complexity Landscape of Waypoint Routing," *SIGCOMM Comput. Commun. Rev*, vol. 48, no. 1, January 2018.
[15] J. Edmonds and E. Johnson, "Matching: a well-solved class of linear programs," in *Combinatorial Structures and their Applications: Proceedings of the Calgary Symposium*. New York: Gordon and Breach, 1970, pp. 88–92.
[16] P. Chanas, "Reseaux atm: conception et optimisation," Ph.D. dissertation, University of Grenoble, 1998, these de doctorat dirige par Finke, Gerd et Burlet, Michel Sciences appliques Grenoble 1 1998. [Online]. Available: http://www.theses.fr/1998GRE10113
[17] A. Jarry and S. Pérennes, "Disjoint paths in symmetric digraphs," *Discrete Applied Mathematics*, vol. 157, no. 1, pp. 90–97, 2009.
[18] 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.
[19] 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.
[20] 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.
[21] 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.
[22] D. Geller and B. Manvel, "Reconstruction of cacti," *Canad. J. Math*, vol. 21, pp. 1354–1360, 1969.
[23] 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.
[24] 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.
[25] A. Björklund and T. Husfeldt, "Shortest two disjoint paths in polynomial time," in *Proc. ICALP*, 2014.
[26] S. Fortune, J. E. Hopcroft, and J. Wyllie, "The directed subgraph homeomorphism problem," *Theor. Comput. Sci.*, vol. 10, pp. 111–121, 1980.
[27] 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.
[28] 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.
[29] 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.
[30] A. M. Frieze, "Edge-disjoint paths in expander graphs," *SIAM J. Comput.*, vol. 30, no. 6, pp. 1790–1801, 2000.
[31] S. Akhoondian Amiri, K.-T. Foerster, and S. Schmid, "Walking Through Waypoints," in *LATIN*, ser. Lecture Notes in Computer Science, 2018.
[32] H. L. Bodlaender, "Treewidth: Structure and algorithms," in *SIROCCO*, ser. Lecture Notes in Computer Science, vol. 4474. Springer, 2007, pp. 11–25.
[33] ——, "A tourist guide through treewidth," *Acta Cybern.*, vol. 11, no. 1-2, pp. 1–21, 1993.
[34] T. Kloks, *Treewidth, Computations and Approximations*, ser. Lecture Notes in Computer Science. Springer, 1994, vol. 842.
[35] X. Zhou, S. Tamura, and T. Nishizeki, "Finding edge-disjoint paths in partial *k*-trees," *Algorithmica*, vol. 26, no. 1, pp. 3–30, 2000.
[36] 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.
[37] G. J. Woeginger, "Exact algorithms for np-hard problems: A survey," in *Combinatorial Optimization*, ser. LNCS, vol. 2570. Springer, 2001, pp. 185–208.
[38] 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.
[39] M. Becker and K. Mehlhorn, "Algorithms for routing in planar graphs," *Acta Informatica*, vol. 23, no. 2, pp. 163–176, 1986.
[40] 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.
[41] H. Bodlaender, "Dynamic programming on graphs with bounded treewidth," *Automata, Languages and Programming*, pp. 105–118, 1988.
[42] 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.
[43] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A Survey*. Philadelphia, PA, USA: SIAM, 1999.
[44] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
[45] ETSI, "Network functions virtualisation – introductory white paper," *White Paper*, oct 2013.
[46] J. Napper, W. Haeffner, M. Stiemerling, D. R. Lopez, and J. Uttaro, "Service Function Chaining Use Cases in Mobile Networks," Internet-Draft, Apr. 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-sfc-use-case-mobility-06
[47] 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.
[48] 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.
[49] G. Even, M. Medina, and B. Patt-Shamir, "Online path computation and function placement in sdns," in *Proc. SSS*, 2016.
[50] T. Lukovszki and S. Schmid, "Online admission control and embedding of service chains," in *SIROCCO*, 2015.
[51] G. Even, M. Rost, and S. Schmid, "An approximation algorithm for path computation and function placement in SDNs," in *SIROCCO*, 2016.
[52] N. Bansal, K.-W. Lee, V. Nagarajan, and M. Zafer, "Minimum congestion mapping in a cloud," in *Proc. ACM PODC*, 2011.
[53] K.-T. Foerster, M. Parham, and S. Schmid, "A walk in the clouds: Routing through vnfs on bidirected networks," in *ALGOCLOUD*, ser. Lecture Notes in Computer Science, vol. 10739. Springer, 2017, pp. 11–26.