Walking Through Waypoints

Saeed Akhoondian Amiri[1], Klaus-Tycho Foerster[2], and Stefan Schmid[3]

[1] TU Berlin and Max Planck Institute for Informatics (Saarland), Germany
saeed.amiri@tu-berlin.de
[2] University of Vienna, Austria
klaus-tycho.foerster@univie.ac.at
[3] University of Vienna, Austria
stefan_schmid@univie.ac.at

Abstract. We initiate the study of a fundamental combinatorial problem: Given a capacitated graph $G = (V, E)$, find a shortest walk ("route") from a source $s \in V$ to a destination $t \in V$ that includes all vertices specified by a set $\mathscr{W} \subseteq V$: the *waypoints*. This waypoint routing problem finds immediate applications in the context of modern networked distributed systems. Our main contribution is an exact polynomial-time algorithm for graphs of bounded treewidth. We also show that if the number of waypoints is logarithmically bounded, exact polynomial-time algorithms exist even for general graphs. Our two algorithms provide an almost complete characterization of what can be solved exactly in polynomial-time: we show that more general problems (e.g., on grid graphs of maximum degree 3, with slightly more waypoints) are computationally intractable.

1 Introduction

How fast can we find a shortest route, i.e., *walk*, from a source s to a destination t which visits a given set of waypoints in a graph, but also respects edge capacities, limiting the number of traversals? This fundamental combinatorial problem finds immediate applications, e.g., in modern networked systems connecting distributed network functions However, surprisingly little is known today about the fundamental algorithmic problems underlying *walks through waypoints*.

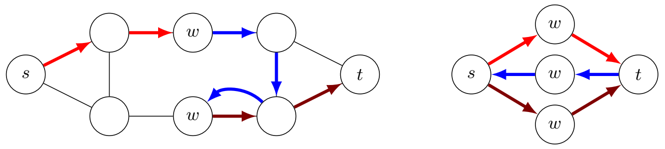

Fig. 1. Two shortest walks and their decompositions into three paths each: In both graphs, we walk through all waypoints from s to t by first taking the *red*, then the *blue*, and lastly the *brown* path. The existence of a solution in the left graph (e.g., a walk of length 7 in this case) relies on one edge incident to a waypoint having a capacity of at least two. In the right graph, it is sufficient that all edges have unit capacity. Note that no $s - t$ path through all waypoints exists, for either graph.

The problem features interesting connections to the disjoint paths problem, however, in contrast to disjoint paths, we (1) consider *walks* (of unit resource *demand* each time an edge is traversed) on *capacitated* graphs rather than *paths* on *uncapaciatated* graphs, and we (2) require that a set of specified vertices are visited. We refer to Figure 1 for two examples.

Model. The inputs to the *Waypoint Routing Problem (WRP)* are: (1) a connected, undirected, capacitated and weighted graph $G = (V, E, c, \omega)$ consisting of $n = |V| > 1$ vertices, where $c\colon E \to \mathbb{N}$ represents edge capacities and $\omega\colon E \to \mathbb{N}$ represents the edge costs. (2) A source-destination vertex pair $s, t \subseteq V(G)$. (3) A set of k waypoints $\mathscr{W} = (w_1, \ldots, w_k) \in V(G)^k$.

We observe that the route (describing a walk) can be decomposed into simple paths between terminals and waypoints, and we ask: Is there a *route* R, which w.l.o.g. can be decomposed into $k+1$ path segments $R = P_1 \oplus \ldots \oplus P_{k+1}$, where:

1. *Capacities are respected:* We assume unit demands and require $|\{i \mid e \in P_i \in R, i \in [1, k+1]\}| \leq c(e)$ for every edge $e \in E$.
2. *Waypoints are visited:* Every element in $\mathscr{W}$ appears as an endpoint of exactly two distinct paths in route R and s is an endpoint of P_1 and t is an endpoint of P_{k+1}. We note that the k waypoints can be visited in any order.
3. *Walks are short:* The length $\ell = |P_1| + \ldots + |P_{k+1}|$ of route R w.r.t. edge traversal cost ω is minimal.

Remark I: Reduction to Edge-Disjoint Problems. Without loss of generality, it suffices to consider capacities $c\colon E \to \{1, 2\}$, as shown in [38, Fig. 1]: a walk R which traverses an edge e more than twice, cannot be a shortest one.

This also gives us a simple reduction of the capacitated problem to an uncapacitated (i.e., unit capacity), *edge-disjoint* problem variant, by using at most two parallel edges per original edge. Depending on the requirements, we will further subdivide these parallel edges into paths (while preserving distances and graph properties such as treewidth, at least approximately).

Remark II: Reduction to Cycles. Without loss of generality and to simplify presentation, we focus on the special case $s = t$. We show that we can modify instances with $s \neq t$ to instances with $s = t$ in a distance-preserving manner and by increasing the treewidth by at most one. Our NP-hardness results hold for $s = t$ as well. The proof is deferred to the full version of this paper.

1.1 Our Contributions

We initiate the study of a fundamental waypoint routing problem. We present polynomial-time algorithms to compute shortest routes (walks) through arbitrary waypoints on graphs of bounded treewidth and to compute shortest routes on general graphs through a bounded (but not necessarily constant) number of waypoints. We show that it is hard to significantly generalize these results both in terms of the family of graphs as well as in terms of the number of waypoints, by deriving NP-hardness results: Our exact algorithms cover a good fraction of the problem space for which polynomial-time solutions exist. More precisely, we present the following results:

1. **Shortest Walks on Arbitrary Waypoints:** While many vertex disjoint problem variants like Hamiltonian path, TSP, vertex disjoint paths, etc. are often polynomial-time solvable in graphs of bounded treewidth, their edge-disjoint counterparts (like the edge-disjoint problem), are sometimes NP-hard already on series-parallel graphs. As the Waypoint Routing Problem is an edge-based problem, one might expect that the problem is NP-hard already on bounded treewidth graphs, similarly to the edge-disjoint paths problem. Yet, and perhaps surprisingly, we prove that a shortest walk through an arbitrary number of waypoints can be computed in polynomial time on graphs of bounded treewidth. By employing a simple trick, we transform the capacitated problem variant to an uncapacitated edge-disjoint problem: the resulting uncapacitated graph has almost the same treewidth. We then employ a well-known dynamic programming technique on a nice tree decomposition of the graph. However, since the walk is allowed to visit a vertex multiple times, we cannot rely on techniques which are known for vertex-disjoint paths. Moreover, we cannot simply use the line graph of the original graph: the resulting graph does not preserve the bounded treewidth property. Accordingly, we develop new methods and tools to deal with these issues.
2. **Shortest Walks on Arbitrary Graphs:** We show that a shortest route through a logarithmic number of waypoints can be computed in randomized time on general graphs, by reduction to the vertex-disjoint cycle problem in [7]. Similarly, we show that a route through a loglog number of waypoints can be computed in deterministic polynomial time on general graphs via [35]. Again, we show that that this is almost tight, in the sense that the problem becomes NP-hard for any polynomial number of waypoints. This reduction shows that the edge-disjoint paths problem is not harder than the vertex-disjoint problem on general graphs, and the hardness result also implies that [7] is nearly asymptotically tight in the number of waypoints.

1.2 A Practical Motivation

The problem of finding routes through waypoints or specified vertices is a natural and fundamental one. We sketch just one motivating application, arising in the context of modern networked systems. Whereas traditional computer networks were designed with an "end-to-end principle" [50] philosophy in mind, modern networks host an increasing number of "middleboxes" or network functions, distributed across the network, in order to improve performance (e.g., traffic optimizers, caches, etc.), security (e.g., firewalls, intrusion detection systems), or scalability (e.g., network address translation). Moreover, middleboxes are increasingly virtualized (a trend known as network function virtualization [23]) and can be deployed flexibly at arbitrary locations in the network (not only at the edge) and at low costs. Accordingly, also more flexible routing schemes have been developed, enabled in particular by the software-defined networking paradigm [27], to route the traffic through these (virtualized) middleboxes to compose more complex network services (also known as service chains [24]). Thus, the resulting traffic routes can be modeled as walks, and the problem of finding shortest routes through such middleboxes (the waypoints) is an instance of WRP.

1.3 Related Work

The Waypoint Routing Problem is closely related to disjoint paths problems arising in many applications [41,44,56]. Indeed, assuming unit edge capacities and a single waypoint w, the problem of finding a shortest walk (s, w, t) can be seen as a problem of finding two shortest (edge-)disjoint paths (s, w) and (w, t) with a common vertex w. More generally, a shortest walk $(s, w_1, \ldots, w_k, t)$ in a unit-capacity graph can be seen as a sequence of $k+1$ disjoint paths. The edge-disjoint and vertex-disjoint paths problem (sometimes called *min-sum* disjoint paths) is a deep and intensively studied combinatorial problem, also in the context of parallel algorithms [36,37]. Today, we have a fairly good understanding of the *feasibility* of k-disjoint paths: for constant k, polynomial-time algorithms for general graphs have been found by Ohtsuki [45], Seymour [54], Shiloah [55], and Thomassen [57] in the 1980s, and for general k it is NP-hard [34], already on series-parallel graphs [43], i.e., graphs of treewidth at most two. However, the *optimization* problem (i.e., finding *shortest* paths) continues to puzzle researchers, even for $k = 2$. Until recently, despite the progress on polynomial-time algoritms for special graph families like variants of planar graphs [4,19,40] or graphs of bounded treewidth [51], no subexponential time algorithm was known even for the 2-disjoint paths problem on general graphs [21,29,40]. A recent breakthrough result shows that optimal solutions can at least be computed in *randomized* polynomial time [8]; however, we still have no *deterministic* polynomial-time algorithm. Both existing feasible and optimal algorithms are often impractical [8,18,52,54], and come with high time complexity. We also note that there are results on the *min-max* version of the disjoint paths problem, which asks to minimize the length of the longest path. The *min-max* problem is believed to be harder than *min-sum* [33,40].

The problem of finding shortest (edge- and vertex-disjoint) paths and cycles through k waypoints has been studied in different contexts already. The cycle problem variant is also known as the *k-Cycle Problem* and has been a central topic of graph theory since the 1960s [47]. A cycle from s through $k = 1$ waypoints back to $t = s$ can be found efficiently by breadth first search, for $k = 2$ the problem corresponds to finding a integer flow of size 2 between two vertices, and for $k = 3$, it can still be solved in linear time [30,32]; a polynomial-time solution for any constant k follows from the work on the disjoint paths problem [48]. The best known deterministic algorithm to compute *feasible* (but not necessarily shortest) paths is by Kawarabayashi [35]: it finds a cycle for up to $k = O((\log\log n)^{1/10})$ waypoints in deterministic polynomial time. Björklund et al. [7] presented a randomized algorithm based on algebraic techniques which finds a shortest simple cycle through a given set of k vertices or edges in an n-vertex undirected graph in time $2^k n^{O(1)}$. In contrast , we assume capacitated networks and do not enforce routes to be edge or vertex disjoint, but rather consider (shortest) walks.

Regarding capacitated graphs, researchers have explored the admission control problem variant: the problem of admitting a maximal number of routing requests such that capacity constraints are met. For example, Chekuri et al. [16] and Ene et al. [22] presented approximation algorithms for maximizing the benefit of admitting disjoint paths in bounded treewidth graphs with both edge and vertex capacities. Even et al. [25,26] and Rost et al. [49] initiated the study of approximation algorithms for admitting a maximal number of routing *walks* through waypoints. In contrast, we focus on the optimal routing of a single walk.

In the context of capacitated graphs and single walks, the applicability of edge-disjoint paths algorithms to the so-called *ordered* Waypoint Routing problem was studied in [2,31], where the task is to find $k+1$ capacity-respecting paths $(s, w_1,), (w_1, w_2), \dots, (w_k, t)$. An extension of their methods to the *unordered* Waypoint Routing problem via testing all possible $k!$ orderings falls short of our results: For general graphs, only $O(1)$ waypoints can be considered, and for graphs of bounded treewidth, only $O(\log n)$ waypoints can be routed in polynomial time [2]; both results concern feasibility only, but not shortest routes. We provide algorithms for $O(\log n)$ waypoints on general graphs and $O(n)$ waypoints in graphs of bounded treewidth, in both cases for shortest routes.

Lastly, for the case that all edges have a capacity of at least two and $s = t$, a direct connection of WRP to the subset traveling salesman problem (TSP) can be made [31]. In the subset TSP, the task is to find a shortest closed walk that visits a given subset of the vertices [38]. As optimal routes for WRP and subset TSP traverse every edge at most twice, optimal solutions for both are identical when $\forall e \in E : c(e) \geq 2$. Hence, we can make use of the subset TSP results of Klein and Marx, with time of $(2^{O\left(\sqrt{k} \log k\right)} + \max_{\forall e \in E} \omega(e)) \cdot n^{O(1)}$ on planar graphs. Klein and Marx also point out applicability of the dynamic programming techniques of Bellman and of Held and Karp, allowing subset TSP to be solved in time of $2^k \cdot n^{O(1)}$. For a PTAS on bounded genus graphs, we refer to [14]. We would like to note at this point that the technique for $s \neq t$ of Remark II does not apply if all edges must have a capacity of at least two. Similarly, it is in general not clear how to directly transfer $s = t$ TSP results to the case of $s \neq t$, cf. [53]. Notwithstanding, as WRP also allows for unit capacity edges (to which subset TSP is oblivious), WRP is a generalization of subset TSP.

Paper Organization. In Section 2 we present our results for bounded treewidth graphs and Section 3 considers general graphs. We derive distinct NP-hardness results in Section 4 and conclude in Section 5. Due to space constraints, some technical contents are deferred to the full version of this paper.[4]

2 Walking Through Waypoints on Bounded Treewidth

The complexity of the Waypoint Routing Problem on bounded treewidth graphs is of particular interest: while vertex-disjoint paths and cycles problems are often polynomial-time solvable on bounded treewidth graphs (e.g., vertex disjoint paths [48], vertex coloring, Hamiltonian cycles [6], Traveling Salesman [13], see also [11,28]) many edge-disjoint problem variants are NP-hard (e.g., edge-disjoint paths [43], edge coloring [42]). Moreover, the usual *line graph* construction approaches to transform vertex-disjoint to edge-disjoint problems are not applicable as bounded treewidth is not preserved under such transformations.

[4] A preliminary full version is provided at [3].

Against this backdrop, we show that indeed shortest routes through arbitrary waypoints can be computed in polynomial-time for bounded treewidth graphs.

Theorem 1. *The Waypoint Routing Problem can be solved in time of $n^{O(\mathtt{tw}^2)}$, where $\mathtt{tw}$ denotes the treewidth of the network.*

In other words, the Waypoint Routing Problem is in the complexity class XP [17,20] w.r.t. treewidth. We obtain:

Corollary 1. *The Waypoint Routing Problem can be solved in polynomial time for graphs of bounded treewidth $\mathtt{tw} \in O(1)$.*

Overview. We describe our algorithm in terms of a *nice tree decomposition* [39, Def. 13.1.4] (§2.1). We first transform the edge-capacitated problem into an edge-disjoint problem (on unit edge capacity graphs §2.2), leveraging a simple observation on the structure of waypoint walks and preserving distances. We show that this transformation changes the treewidth by at most an additive constant. We then define the separator signatures (§2.3) and describe how to inductively generate valid signatures in a bottom up manner on the nice tree decomposition, applying the *forget*, *join* and *introduce* operations [39, Def. 13.1.5] (§2.4).

The correctness of our approach relies on a crucial observation on the underlying Eulerian properties of the Waypoint Routing Problem in Lemma 2, allowing us to bound the number of partial walks we need to consider at the separator, see Figure 2 for an example. Finally in §2.5, we bring together the different bits and pieces, and sketch how to dynamically program [10] the shortest waypoint walk on the rooted separator tree.

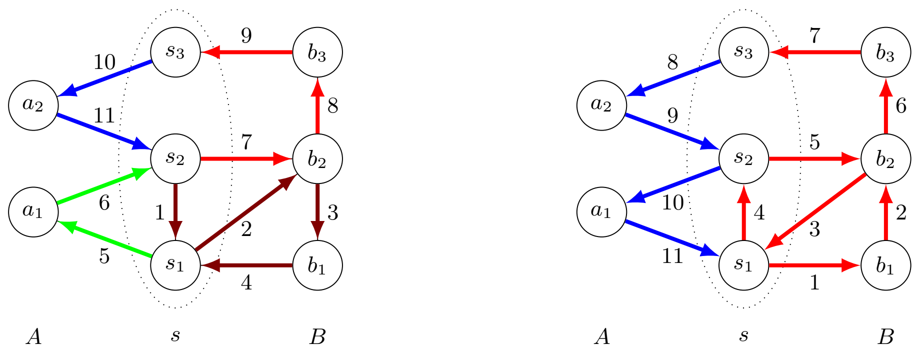

Fig. 2. Two different methods to choose an Eulerian walk, where the numbers from 1 to 11 describe the order of the traversal. In the left walk, the separator s is crossed 4 times, but only 2 times in the right walk. Furthermore, in the left walk, there are 2 walks each in $G[A]$ (green and blue) and $G[B]$ (brown and red), respectively. In the right walk, there is only 1 walk for $G[A]$ (blue) and 1 walk for $G[B]$ (red).

2.1 Treewidth Preliminaries

A *tree decomposition* $\mathcal{T} = (T, X)$ of a graph G consists of a bijection between a tree T and a collection X, where every element of X is a set of vertices of G such that: (1) each graph vertex is contained in at least one tree node (the *bag* or *separator*), (2) the tree nodes containing a vertex v form a connected subtree of T, and (3) vertices are adjacent in the graph only when the corresponding subtrees have a node in common.

The *width* of $\mathcal{T} = (T, X)$ is the size of the largest set in X minus 1, with the *treewidth* of G being the minimum width of all possible tree decompositions of G.

A *nice tree decomposition* is a tree decomposition such that: (1) it is rooted at some vertex r, (2) leaf nodes are mapped to bags of size 1, and (3) inner nodes are of one of three types: *forget* (a vertex leaves the bag in the parent node), *join* (two bags defined over the same vertices are merged) and *introduce* (a vertex is added to the bag in the parent node). The tree can be iteratively constructed by applying simple *forget*, *join* and *introduce* types.

Let $b \in X$ be a bag of the decomposition corresponding to a vertex $b \in V(T)$. We denote by T_b the maximal subtree of T which is rooted at bag b. By $G[b]$ we denote the subgraph of G induced on the vertices in the bag b and by $G[T_b]$ we denote the subgraph of G which is induced on vertices in all bags in $V(T_b)$. We will henceforth assume that a nice tree decomposition $\mathcal{T} = (T, X)$ of a graph G is given, covering its computation in the final steps of the proof of Theorem 1.

2.2 Unified Graphs

We begin by transforming our graphs into graphs of unit edge capacity, preserving distances and approximately preserving treewidth.

Definition 1 (Unification). *Let G be an arbitrary, edge capacitated graph. The* unified *graph G^u of G is obtained from G by the following operations on each edge $e \in E(G)$: We replace e by $c(e)$ parallel edges $e_1, \dots, e_{c(e)}$, subdivide each resulting parallel edge by creating vertices $v_i^e, i \in [c(e)]$), and set the weight of each subdivided edge to $w(e)/2$ (i.e., the total weight is preserved). We set all edge capacities in the unified graph to 1. Similarly, given the original problem instance I of the Waypoint Routing Problem, the* unified instance *I^u is obtained by replacing the graph G in I with the graph G^u in I^u, without changing the waypoints, the source and the destination.*

It follows directly from the construction that I and I^u are equivalent with regards to the contained walks. Moreover, as we will see, the unification process approximately preserves the treewidth. Thus, in the following, we will focus on G^u and I^u only, and implictly assume that G and I are unified. Before we proceed further, however, let us introduce some more definitions. Using Remark I, w.l.o.g., we can focus on graphs where for all $e \in E$, $c(e) \leq 2$. The treewidth of G and G^u are preserved up to an additive constant.

Lemma 1. *Let G be an edge capacitated graph such that each edge has capacity at most 2 and let $\mathtt{tw}$ be the treewidth of G. Then G^u has treewidth at most $\mathtt{tw} + 1$.*

Leveraging Eulerian Properties A key insight is that we can leverage the Eulerian properties implied by a waypoint route. In particular, we show that the traversal of a single Eulerian walk (e.g., along an optimal solution of WRP) can be arranged s.t. it does not traverse a specified separator too often, for which we will later choose the root of the nice tree decomposition.

Lemma 2 (Eulerian Separation). *Let G be an Eulerian graph. Let s be an (A, B) separator of order $|s|$ in G. Then there is a set of $\ell \leq 2|s|$ pairwise edge-disjoint walks $\mathcal{W} = \{W_1, \dots, W_\ell\}$ of G such that*

1. *For every $W \in \mathcal{W}$, W has both of its endpoints in $A \cap B$.*
2. *Every walk $W \in \mathcal{W}$ is entirely either in $G[A]$ (as $\mathcal{W}_A$) or in $G[B]$ (as $\mathcal{W}_B$).*
3. *Let β_A be the size of the set of vertices used by $\mathcal{W}_A$ as an endpoint in s. Then, $\mathcal{W}_A$ contains at most β_A walks. Analogously, for β_B and $\mathcal{W}_B$.*
4. *There is an Eulerian walk W of G such that: $W := W_1 \oplus \ldots \oplus W_\ell$.*

2.3 Signature Generation and Properties

We next introduce the signatures we use to represent previously computed solutions to subproblems implied by the separators in the (nice) tree decomposition. For every possible signature, we will determine whether it represents a proper/valid solution for the subproblem, and if so, store it along with an exemplary sub-solution of optimal weight.

In a nutshell, the signature describes endpoints of (partial) walks on each side of the separator. These partial walks hence need to be iteratively merged, forming signatures of longer walks through the waypoints.

Definition 2 (Signature). *Let $b \in X$. A signature σ of b (σ_b) is a pair, either containing*

1. *1) an unordered tuple of pairs of vertices $s_i, r_i \in b$ and 2) a subset $E_b \subseteq E(G[b])$ with $\sigma_b = (((s_1, r_1), (s_2, r_2), \dots, (s_\ell, r_\ell)), E_b)$ s.t. $\ell \leq |b|$, or*
2. *1) $\emptyset$ and 2) $\emptyset$, with $\sigma_b = (\emptyset, \emptyset)$, also called an* empty *signature $\sigma_{b,\emptyset}$.*

Note that in the above definition we may have $s_i = r_i$ for some i. We can now define a valid signature and a sub-solution, where we consider the vertex $s = t$ to be a waypoint.

Definition 3 (Valid Signature and Sub-Solution). *Let $b \in X$ and let either $\sigma_b = (\{(s_1, r_1), (s_2, r_2), \dots, (s_\ell, r_\ell)\}, E_b)$ or $\sigma_b = \sigma_{b,\emptyset}$ be a signature of b. $\sigma_b \neq \sigma_{b,\emptyset}$ is called a* valid signature *if there is a set of pairwise edge-disjoint walks $\mathcal{W}_{\sigma_b} = \{W_1, \dots, W_\ell\}$ such that:*

1. *If W_i is an open walk then it has both of its endpoints on (s_i, r_i), otherwise, $s_i = r_i$ and $s_i \in V(W_i)$.*
2. *Let β be the size of the set of endpoints used by σ_b. Then, it holds that $\beta \geq \ell$.*
3. *For every waypoint $w \in V(T_b)$ it holds that w is contained in some walk $W_j, 1 \leq j \leq \ell$.*

4. *Every (pairwise edge-disjoint) walk $W_j \in \mathcal{W}_{\sigma_b}$ only uses vertices from $V(T_b)$ and only edges from $E(T_b) \setminus \overline{E}_b$, with $\overline{E}_b = E(b) \setminus E_b$.*
5. *Every edge $e \in E_b$ is used by a walk in $\mathcal{W}_{\sigma_b}$.*
6. *Among all such sets of ℓ walks, $\mathcal{W}_{\sigma_b}$ has minimum total weight.*

Additionally, if for a signature $\sigma_b \neq \sigma_{b,\emptyset}$ there is such a set $\mathcal{W}_{\sigma_b}$ (possibly abbreviated by $\mathcal{W}_b$ if clear in the context), we say $\mathcal{W}$ is a valid sub-solution *in $G[T_b]$. For some waypoint contained in $G[T_b]$, we call a signature $\sigma_{b,\emptyset}$* valid*, if there is one walk W associated with it, s.t. W traverses all waypoints in $G[T_b]$, does not traverse any vertex in $V(b)$, and among all such walks in $G[T_b]$ has minimum weight. If $G[T_b]$ does not contain any waypoint, we call a signature $\sigma_{b,\emptyset}$ valid, if there is no walk associated with it.*

Lemma 3 (Number of different signatures). *There are $2^{O(|b|^2)}$ different signatures for $b \in X$.*

2.4 Programming the Nice Tree Decomposition

The nice tree decomposition directly gives us a constructive way to dynamically program WRP in a bottom-up manner. We first cover leaf nodes in Lemma 4, and then work our way up via forget (Lemma 5), introduce (Lemma 6), and join (Lemma 7) nodes, until eventually the root node is reached. Along the way, we inductively generate all valid signatures at every node.

Lemma 4 (Leaf nodes). *Let b be a leaf node in the nice tree decomposition $\mathcal{T} = (T, X)$. Then, in time $O(1)$ we can find all the valid signatures of b.*

Proof. We simply enumerate all possible valid signatures. As a leaf node only contains one vertex v from the graph, all possible edge sets in the signatures are empty, and we have two options for the pairs: First, none, second, $((v, v))$. The second option is always valid, but the first (empty) one is only valid when v is not a waypoint. □

Due to space constraints, we cannot provide the longer proofs of the other three nodes types, especially for the introduce and join nodes. Nonetheless, we at least sketch the proof idea for the join nodes, as a reduction in their time complexity would be interesting for future work, see also our remarks in Section 5.

Lemma 5 (Forget nodes). *Let b be a forget node in the nice tree decomposition $\mathcal{T} = (T, X)$, with one child $q = child(b)$, where we have all valid signatures for q. Then, in time $2^{O(|b|^2)}$ we can find all the valid signatures of b.*

Lemma 6 (Introduce nodes). *Let b be an introduce node in the nice tree decomposition $\mathcal{T} = (T, X)$, with one child $q = child(b)$, where we have all valid signatures for q. Then, in time $|b|^{O(|b|^2)}$ we can find all the valid signatures of b.*

Lemma 7 (Join nodes). *Let b be a join node in the nice tree decomposition $\mathcal{T} = (T, X)$, with the two children $q_1 = child(b)$ and $q_2 = child(b)$, where we have all valid signatures for q_1 and q_2. Then, in time $n^{O(|b|)} \cdot 2^{O(|b|^2)}$ we can find all the valid signatures of b.*

Our proof for join nodes consists of two parts, making use of the following fact: For a given valid signature of b, two valid sub-solutions with different path traversals have the same total length, if the set of traversed edges is identical. As thus, when trying to re-create a signature of b with a valid sub-solution, we do not need to create this specific sub-solution, but just *any* sub-solution using the *same* set of endpoints and edges. We show:

1. We can partition the edges of a valid sub-solution into two parts along a separator, resulting in a valid signature for each of the two parts, where each sub-solution uses exactly the edges in its part.
2. Given a sub-solution for each of the two parts separated, we can merge their edge sets, and create all possible signatures and sub-solutions using this merged edge set.

2.5 Putting it All Together

We now have all the necessary tools to prove Theorem 1:

Proof (Thm. 1). **Dynamically programming a nice tree decomposition.** Translating an instance of the Waypoint Routing Problem to an equivalent one with $s = t$ and unit edge capacities only increases the treewidth by a constant amount, see Remark II and Lemma 1. Although it is NP-complete to determine the treewidth of a graph and compute an according tree decomposition, there are efficient algorithms for constant treewidth [12,47]. Furthermore, Bodlaender et al. [9] presented a constant-factor approximation in a time of $O(c^{\texttt{tw}}n)$ for some $c \in \mathbb{N}$, also beyond constant treewidth: Using their algorithm $O(\log \texttt{tw})$ times (via binary search over the unknown treewidth size), we obtain a tree decomposition of width $O(\texttt{tw})$. Following [39], we generate a nice tree decomposition of treewidth $O(\texttt{tw})$ with $O(\texttt{tw}n) \in O(n^2)$ nodes in an additional time of $O(\texttt{tw}^2 n) \in O(n^3)$. The total time so far is $O(c^{\texttt{tw}}n \log \texttt{tw}) + O(\texttt{tw}^2 n)$ for some $c \in \mathbb{N}$.

We can now dynamically program the Waypoint Routing Problem on the nice tree decomposition in a bottom-up manner, using Lemma 4 (leaf nodes), Lemma 5 (forget nodes), Lemma 6 (introduce nodes), and Lemma 7 (join nodes). The time for each programming of a node is at most $O(\texttt{tw})^{O(\texttt{tw}^2)}$ or $n^{O(\texttt{tw})} \cdot 2^{O(\texttt{tw}^2)}$, meaning that we obtain all valid signatures with valid sub-solutions at the root node r, in a combined time of $n^{O(\texttt{tw}^2)}$, specifically:

$$\left(O(\texttt{tw})^{O(\texttt{tw}^2)} + n^{O(\texttt{tw})} \cdot 2^{O(\texttt{tw}^2)}\right) \cdot O(\texttt{tw} \cdot n) + O(c^{\texttt{tw}} n \log \texttt{tw}) + O(\texttt{tw}^2 n).$$

Obtaining an optimal solution. If an optimal solution I to the Waypoint Routing Problem exists (on the unified graph with $s = t$), then the traversed edges E^* and vertices V^* in I yield an Eulerian graph $G^* = (V^*, E^*)$. With each bag in the nice tree decomposition having $O(\texttt{tw})$ vertices, we can now apply (the Eulerian separation) Lemma 2: There must be a valid signature of the root r whose sub-solution uses exactly the edges E^*. As thus, from all the valid sub-solutions at r, we pick any solution to WRP with minimum weight, obtaining an optimal solution to the Waypoint Routing Problem. □

3 Walking Through Logarithmically Many Waypoints

While the Waypoint Routing Problem is generally NP-hard (as we will see below), we show that a shortest walk through a bounded (not necessarily constant) number of waypoints can be computed in polynomial time. We make use of reductions to shortest vertex-disjoint [7,35] cycles problems, where the cycle has to pass through specified vertices.

Theorem 2. *For a general graph G with polynomial edge weights, a shortest walk through $k \in O(\log n)$ waypoints can be computed in randomized polynomial time, namely $2^k n^{O(1)}$. Furthermore, a walk through $k \in O\left((\log \log n)^{1/10}\right)$ waypoints in G can be computed in deterministic polynomial time.*

4 NP-Hardness

Given our polynomial-time algorithms to compute shortest walks through arbitrary waypoints on bounded treewidth graphs as well as to compute shortest walks on arbitrary graphs through a bounded number of waypoints, one may wonder whether exact polynomial time solutions also exist for more general settings. In the following, we show that this is not the case: in both dimensions (number of waypoints and more general graph families), we inherently hit computational complexity bounds. Our hardness results follow by reduction from a special subclass of NP-hard Hamiltonian cycle problems [1,15]:

Theorem 3. *WRP is NP-hard for any graph family of degree at most 3, for which the Hamiltonian Cycle problem is NP-hard.*

We have the following implication for grid graphs [5,15,46] of maximum degree 3, and can use similar ideas for the class of 3-regular bipartite planar graphs.

Corollary 2. *For any fixed constant $r \geq 1$ it holds that WRP is NP-hard on* 1) 3*-regular bipartite planar graphs and* 2) *grid graphs of maximum degree* 3*, respectively, already for $k \in O(n^{1/r})$.*

Our proof techniques also apply to the k-Cycle problem studied by Björklund et al. [7], whose solution is polynomial for logarithmic k. All possible edge-disjoint solutions are also vertex-disjoint, due to the restriction of maximum degree 3.

Corollary 3. *For any fixed constant $r \geq 1$ it holds that the k-Cycle problem is NP-hard on* 1) 3*-regular bipartite planar graphs and* 2) *grid graphs of maximum degree* 3*, respectively, already for $k \in O(n^{1/r})$.*

5 Conclusion

Motivated by the more general routing models introduced in modern software-defined and function virtualized distributed systems, we initiated the algorithmic study of computing shortest walks through waypoints on capacitated networks. We have shown, perhaps surprisingly, that polynomial-time algorithms exist for a wide range of problem variants, and in particular for bounded treewidth graphs.

In our dynamic programming approach to the Waypoint Routing Problem, parametrized by treewidth, we provided fixed-parameter tractable (FPT) algorithms for leaf, forget, and introduce nodes, but an XP algorithm for join nodes. In fact, while we do not know whether our problem can be expressed in monadic second-order logic MSO2, we can show that simply concatenating child-walks for join nodes does not result in all valid parent signatures.

We believe that our paper opens an interesting area for future research. In particular, it will be interesting to further chart the complexity landscape of the Waypoint Routing Problem, narrowing the gap between problems for which exact polynomial-time solutions do and do not exist. Moreover, it would be interesting to derive a lower bound on the runtime of (deterministic and randomized) algorithms on bounded treewidth graphs.

Acknowledgments. The authors would like to thank Riko Jacob for helpful discussions and feedback. Klaus-Tycho Foerster's and Stefan Schmid's research was partly supported by the Villum project ReNet and by Aalborg University's PreLytics project. Saeed Amiri's research was partly supported by the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No 648527).

References

1. Takanori Akiyama, Takao Nishizeki, and Nobuji Saito. NP-completeness of the hamiltonian cycle problem for bipartite graphs. *Journal of Information processing*, 3(2):73–76, 1980.
2. Saeed Akhoondian Amiri, Klaus-Tycho Foerster, Riko Jacob, and Stefan Schmid. Charting the Complexity Landscape of Waypoint Routing. *arXiv preprint 1705.00055*, 2017.
3. Saeed Akhoondian Amiri, Klaus-Tycho Foerster, and Stefan Schmid. Walking Through Waypoints. *arXiv preprint 1708.09827*, 2017.
4. Saeed Akhoondian Amiri, Ali Golshani, Stephan Kreutzer, and Sebastian Siebertz. Vertex disjoint paths in upward planar graphs. In *Proc. CSR*, 2014.
5. Esther M. Arkin, Sándor P. Fekete, Kamrul Islam, Henk Meijer, Joseph S. B. Mitchell, Yurai Núñez Rodríguez, Valentin Polishchuk, David Rappaport, and Henry Xiao. Not being (super)thin or solid is hard: A study of grid hamiltonicity. *Comput. Geom.*, 42(6-7):582–605, 2009.
6. Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23(1):11–24, 1989.
7. Andreas Björklund, Thore Husfeld, and Nina Taslaman. Shortest cycle through specified elements. In *Proc. SODA*, 2012.
8. Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. In *Proc. ICALP*, 2014.
9. 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.

10. Hans Bodlaender. Dynamic programming on graphs with bounded treewidth. *Automata, Languages and Programming*, pages 105–118, 1988.
11. Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993.
12. Hans L Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.
13. Hans L. Bodlaender, Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. *Inf. Comput.*, 243:86–111, 2015.
14. Glencora Borradaile, Erik D. Demaine, and Siamak Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, 68(2):287–311, 2014.
15. Michael Buro. Simple amazons endgames and their connection to hamilton circuits in cubic subgrid graphs. In *Proc. Computers and Games*, 2000.
16. Chandra Chekuri, Sanjeev Khanna, and F Bruce Shepherd. A note on multiflows and treewidth. *Algorithmica*, 54(3):400–412, 2009.
17. Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.
18. Marek Cygan, Dániel Marx, Marcin Pilipczuk, and Michal Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In *Proc. FOCS*, 2013.
19. E.C. de Verdière and Alexander Schrijver. Shortest vertex-disjoint two-face paths in planar graphs. *ACM Transactions on Algorithms (TALG)*, 7(2):19, 2011.
20. Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer, 1999.
21. Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete applied mathematics*, 85(2):113–138, 1998.
22. Alina Ene, Matthias Mnich, Marcin Pilipczuk, and Andrej Risteski. On Routing Disjoint Paths in Bounded Treewidth Graphs. In *Proc. SWAT*, 2016.
23. ETSI. Network functions virtualisation. *White Paper*, oct 2013.
24. ETSI. Network functions virtualisation (nfv); use cases. `http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf`, 2014.
25. Guy Even, Moti Medina, and Boaz Patt-Shamir. Online path computation and function placement in SDNs. In *Proc. SSS*, 2016.
26. Guy Even, Matthias Rost, and Stefan Schmid. An approximation algorithm for path computation and function placement in SDNs. In *SIROCCO*, 2016.
27. Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to SDN. *Queue*, 11(12), 2013.
28. Michael Fellows, Fedor V Fomin, Daniel Lokshtanov, Frances Rosamond, Saket Saurabh, Stefan Szeider, and Carsten Thomassen. On the complexity of some colorful problems parameterized by treewidth. In *Proc. COCOA*. Springer, 2007.
29. Trevor Fenner, Oded Lachish, and Alexandru Popa. Min-sum 2-paths problems. *Theor. Comp. Sys.*, 58(1):94–110, January 2016.
30. Herbert Fleischner and Gerhard J. Woeginger. Detecting cycles through three fixed vertices in a graph. *Inf. Process. Lett.*, 42(1):29–33, 1992.
31. Klaus-Tycho Foerster, Mahmoud Parham, and Stefan Schmid. A walk in the clouds: Routing through vnfs on bidirected networks. In *Proc. ALGOCLOUD*, 2017.
32. Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

33. Alon Itai, Yehoshua Perl, and Yossi Shiloach. The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12(3):277–286, 1982.
34. Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.
35. Ken-ichi Kawarabayashi. An improved algorithm for finding cycles through elements. In *Proc. IPCO*, 2008.
36. Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. Processor efficient parallel algorithms for the two disjoint paths problem and for finding a kuratowski homeomorph. *SIAM J. Comput.*, 21(3):486–506, 1992.
37. Samir Khuller and Baruch Schieber. Efficient parallel algorithms for testing k-connectivity and finding disjoint s-t paths in graphs. *SIAM J. Comput.*, 20(2):352–375, 1991.
38. Philip N. Klein and Dániel Marx. A subexponential parameterized algorithm for subset TSP on planar graphs. In *Proc. SODA*, 2014.
39. Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.
40. Yusuke Kobayashi and Christian Sommer. On shortest disjoint paths in planar graphs. In *Proc. ISAAC*, 2009.
41. Bernhard Korte, Laszlo Lovasz, Hans Jürgen Prömel, and Lex Schrijver. *Paths, flows, and VLSI-layout.* Springer-Verlag, 1990.
42. Dániel Marx. List edge multicoloring in graphs with few cycles. *Inf. Process. Lett.*, 89(2):85–90, 2004.
43. Takao Nishizeki, Jens Vygen, and Xiao Zhou. The edge-disjoint paths problem is NP-complete for series–parallel graphs. *Discrete Appl. Math.*, 115, 2001.
44. Richard G Ogier, Vladislav Rutenburg, and Nachum Shacham. Distributed algorithms for computing shortest pairs of disjoint paths. *IEEE transactions on information theory*, 39(2):443–455, 1993.
45. Tatsuo Ohtsuki. The two disjoint path problem and wire routing design. In *Graph Theory and Algorithms*, pages 207–216. Springer, 1981.
46. Christos H. Papadimitriou and Umesh V. Vazirani. On two geometric problems related to the traveling salesman problem. *J. Algorithms*, 5(2):231–246, 1984.
47. Ljubomir Perković and Bruce A. Reed. An improved algorithm for finding tree decompositions of small width. *Int. J. Found. Comput. Sci.*, 11(3):365–371, 2000.
48. Neil Robertson and Paul D. Seymour. Graph Minors .XIII. The Disjoint Paths Problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.
49. Matthias Rost and Stefan Schmid. Service chain and virtual network embeddings: Approximations using randomized rounding. *arXiv preprint 1604.02180*, 2016.
50. Jerome H Saltzer, David P Reed, and David D Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2(4):277–288, 1984.
51. Petra Scheffler. *A practical linear time algorithm for disjoint paths in graphs with bounded tree-width.* Technical Report, TU Berlin, 1994.
52. Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM J. Comput.*, 23(4):780–788, 1994.
53. András Sebö and Anke van Zuylen. The salesman's improved paths: A 3/2+1/34 approximation. In *Proc. FOCS*, 2016.
54. Paul D Seymour. Disjoint paths in graphs. *Discrete Math.*, 29(3):293–309, 1980.
55. Yossi Shiloach. A polynomial solution to the undirected two paths problem. *J. ACM*, 27(3):445–456, July 1980.
56. Anand Srinivas and Eytan Modiano. Finding minimum energy disjoint paths in wireless ad-hoc networks. *Wireless Networks*, 11(4):401–417, 2005.
57. Carsten Thomassen. 2-linked graphs. *European J. Comb.*, 1(4):371–378, 1980.