Faster Multi-Modal Route Planning With Bike Sharing Using ULTRA

Jonas Sauer
Karlsruhe Institute of Technology, Germany
jonas.sauer2@kit.edu

Dorothea Wagner
Karlsruhe Institute of Technology, Germany
dorothea.wagner@kit.edu

Tobias Zündorf
Karlsruhe Institute of Technology, Germany
zuendorf@kit.edu

Abstract

We study multi-modal route planning in a network comprised of schedule-based public transportation, unrestricted walking, and cycling with bikes available from bike sharing stations. So far this problem has only been considered for scenarios with at most one bike sharing operator, for which MCR is the best known algorithm [6]. However, for practical applications, algorithms should be able to distinguish between bike sharing stations of multiple competing bike sharing operators. Furthermore, MCR has recently been outperformed by ULTRA for multi-modal route planning scenarios without bike sharing [3]. In this paper, we present two approaches for modeling multi-modal transportation networks with multiple bike sharing operators: The *operator-dependent* model requires explicit handling of bike sharing stations within the algorithm, which we demonstrate with an adapted version of MCR. In the *operator-expanded* model, all relevant information is encoded within an expanded network. This allows for applying any multi-modal public transit algorithm without modification, which we show for ULTRA. We proceed by describing an additional preprocessing step called *operator pruning*, which can be used to accelerate both approaches. We conclude our work with an extensive experimental evaluation on the networks of London, Switzerland, and Germany. Our experiments show that the new preprocessing technique accelerates both approaches significantly, with the fastest algorithm (ULTRA-RAPTOR with operator pruning) being more than an order of magnitude faster than the basic MCR approach. Moreover, the ULTRA preprocessing step also benefits from operator pruning, as its running time is reduced by a factor of 14 to 20.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases Algorithms, Route Planning, Bike Sharing, Public Transportation

Digital Object Identifier 10.4230/LIPIcs.SEA.2020.16

Supplementary Material Source code is available at `https://github.com/kit-algo/Bike-Sharing`.

Funding This research was funded by the DFG under grant number WA 654123-2.

1 Introduction

Research on efficient public transit route planning has seen remarkable advances in the past decade [2]. However, in practice, public transit is most often used in combination with other transportation modes, such as walking or cycling. This results in a multi-modal route planning scenario, which is still a challenge to solve efficiently [22]. In this work we present novel and efficient approaches for finding all Pareto-optimal journeys (regarding travel time and number of used public transit vehicles) in a multi-modal network featuring public transit, unrestricted walking, and bicycles available at bike sharing stations from multiple operators.

Related Work. Several algorithms have been proposed for scenarios where only one of the aforementioned modes of transportation is available. Walking (or cycling) on its own results in a classical shortest path problem, which can be solved using Dijkstra's algorithm [10]. The fastest known algorithms for this problem utilize an additional preprocessing phase to achieve fast query times. One such approach is Contraction Hierarchies (CH) [11], where the preprocessing computes shortcuts that allow the query to skip unimportant vertices. A comprehensive overview of speedup techniques for the shortest path problem is given in [2]. For the special case of route planning for bicycles, an extended scenario where travel time is optimized while constraining the altitude difference has been considered and accelerated with CH [21].

Algorithms for public transit networks can be divided into two groups: graph-based algorithms and algorithms operating directly on the timetable. Graph-based approaches can be further subdivided into *time-dependent* and *time-expanded* techniques, which both enable the usage of normal shortest path algorithms [20, 19]. These techniques have also been adapted for additional optimization criteria, such as number of used vehicles or price [18]. However, accelerating these graph-based approaches is quite difficult [4]. Many efficient algorithms therefore operate directly on the timetable and try to exploit its structure. Examples for such approaches are Transfer Patterns [1], Trip-Based Routing [23], CSA [8], and RAPTOR [7].

While route planning for walking, cycling, or public transit is quite well studied, the combined multi-modal problem still holds many challenges. As before, one possible approach is to model all aspects of the multi-modal network as a graph [13, 15]. In [16], a time-dependent approach was combined with ALT [14] in order to improve query times. Accelerating a graph-based search with CH yields even faster queries while also allowing for user-specific transfer mode preferences [9]. Another approach is to heuristically reduce the search space in order to obtain fast query times [5]. The only timetable-based technique that has been adapted for multi-modal scenarios is RAPTOR, resulting in the MCR algorithm [6]. Most recently, ULTRA (short for UnLimited Transfers) [3] has been presented, which uses a preprocessing step to compute shortcuts for non-timetable-based modes of transportation. These shortcuts can then be used to enable multi-modal queries for many timetable-based algorithms.

Most of the multi-modal algorithms also consider bike sharing as one of the available transportation modes. However, to the best of our knowledge, no algorithm so far has considered the more realistic scenario of multiple competing bike sharing operators.

Contribution. In this work we present two distinct approaches for solving the journey planning problem in multi-modal transportation networks with several bike sharing operators. For the first approach, which we call the operator-dependent model, we show how the well-known MCR algorithm can be adapted for our scenario. We then show that the query time of every label-propagating algorithm (such as MCR) is proportional to the number of bike sharing operators present in the network. This theoretical analysis is later confirmed by experiments. The second approach encodes the bike sharing information into an enlarged transfer graph and public transit schedule. Using this operator-expanded network eliminates the need for specialized algorithms. In particular, this enables us to use ULTRA, the fastest known multi-modal algorithm, for journey planning with bike sharing. For even faster queries we propose an additional preprocessing technique called operator pruning, which can be used with both models. This technique not only reduces the search space and running time of the query algorithms significantly, but also reduces the preprocessing time of ULTRA by

more than an order of magnitude. We conclude our work with an extensive experimental evaluation of our algorithms. We show that the combination of ULTRA-RAPTOR and operator pruning on the operator-expanded model yields the fastest algorithm on real-world networks.

2 Preliminaries

This section introduces the basic notation used throughout this work, as well as the problem statement and the algorithms on which our work is based.

Public Transit Network. A public transit network is a tuple $N = (\mathcal{S}, \mathcal{T}, \mathcal{R}, G)$ consisting of a set of *stops* $\mathcal{S}$, a set of *trips* $\mathcal{T}$, a set of *routes* $\mathcal{R}$ and a directed, weighted *transfer graph* $G = (\mathcal{V}, \mathcal{E})$. A stop $v \in \mathcal{S}$ is a location in the network where passengers can enter or exit a vehicle (e.g., a train, bus, ferry, etc.). Associated with each stop is a non-negative *departure buffer time* $\tau_{\text{buf}}(v)$, which is the minimum amount it takes for a passenger to board a vehicle after arriving at the stop. A trip $\mathrm{T} = \langle v_0, \ldots, v_k \rangle \in \mathcal{T}$ is a sequence of at least two stops which are served consecutively by the same vehicle. For each stop v in the trip, we denote the *arrival time* of the vehicle at v by $\tau_{\text{arr}}(\mathrm{T}, v)$ and the *departure time* by $\tau_{\text{dep}}(\mathrm{T}, v)$. The trips are partitioned into routes $r \in \mathcal{R}$, such that two trips are part of the same route if their stop sequence is identical and they do not overtake each other. A trip $\mathrm{T}_a \in \mathcal{T}$ overtakes a trip $\mathrm{T}_b \in \mathcal{T}$ if their stop sequence contains two stops $u, v \in \mathcal{S}$ such that T_a arrives at or departs from u before T_b, but arrives at or departs from v after T_b. The transfer graph $G = (\mathcal{V}, \mathcal{E})$ consists of a set of *vertices* $\mathcal{V}$ with $\mathcal{S} \subseteq \mathcal{V}$, and a set of *edges* $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. Associated with each edge $e = (u, v) \in \mathcal{E}$ is a *walking time* $\tau_{\text{walk}}(e)$, which is the time required to walk from u to v.

Bike Sharing. In addition to walking, we consider bicycle sharing as a second transfer mode. For each edge $e = (u, v) \in \mathcal{E}$, we define the *biking time* $\tau_{\text{bike}}(e)$ as the time required to travel from u to v with a bicycle. Note that $\tau_{\text{walk}}(e)$ or $\tau_{\text{bike}}(e)$ may be ∞ to signify that e is not usable in the respective transfer mode. Some trips in the public transit network may allow passengers to carry along a bicycle with them, while others may not. The *bicycle transport function* $\mathrm{bt} \colon \mathcal{T} \to \{\texttt{true}, \texttt{false}\}$ maps to each trip $\mathrm{T} \in \mathcal{T}$ a boolean value $\mathrm{bt}(\mathrm{T})$ that indicates whether T allows bicycle transport or not.

Bikes can be rented from a number of different *bike sharing operators*. We denote the number of bike sharing operators by o and associate each operator with a number from $\{1, \ldots, o\}$. For simplicity, we will use the number 0 to denote that a passenger is currently not renting a bike. Each operator i operates a set $\mathcal{BS}_i \subseteq \mathcal{V}$ of *bike sharing stations* where passengers can pick up or drop off a bike. Note that a vertex may act as a bike sharing station for more than one operator. Each bike sharing station v has an associated *pickup time* $\tau_{\text{pick}}(v)$ that is required to pick up a bike, and a *dropoff time* $\tau_{\text{drop}}(v)$ that is required to drop off a bike. A passenger may only carry one bike at a time.

Journey. A *journey* J defines the movement of a passenger through the public transit network when traveling from a source vertex $s \in \mathcal{V}$ to a target vertex $t \in \mathcal{V}$. It is an alternating sequence of *trip legs* and *transfers*, where a trip leg is a subsequence of a trip that represents the passenger using that portion of the trip, and a transfer is a path in the transfer graph that connects the final stop of one trip leg (or s for the initial transfer) with the first stop of the following trip leg (or t for the final transfer). Note that some or all of the transfers may be empty.

To define which parts of the journey use bike sharing, the journey is augmented with a sequence $((u_1, v_1), \ldots, (u_n, v_n))$ that contains one tuple of bike sharing stations for each bike that is rented during the journey. For the i-th rented bike, u_i is the station where the bike is picked up and v_i is the station where it is dropped off. It is required that there is a bike sharing operator j for which $u_i, v_i \in \mathcal{BS}_j$ holds. This ensures that multiple bikes are not rented at the same time, and that the journey starts and ends with no bike rented.

The bike sharing stations must be visited by the transfers of the journey in the same order in which they appear in the sequence. If u_i and v_i are visited during different transfers, all trip legs that lie between these two transfers must allow bicycle transport. For every transfer edge $e \in \mathcal{E}$ used between u_i and v_i, the travel time for using the edge is $\tau_{\text{bike}}(e)$. For every edge $e \in \mathcal{E}$ that is traversed without a bike, the travel time is $\tau_{\text{walk}}(e)$.

Problem Statement. The criteria we use to evaluate a journey J are its arrival time at the target vertex t and the number of used public transit vehicles, i.e., the number of trip legs in J. A journey J *dominates* another journey J' if J arrives no later than J' and does not use more trips than J'. We call a journey J *Pareto-optimal* if it is not dominated by any other journey. A *Pareto set* is a set containing a minimal number of journeys such that every valid journey is dominated by a journey in the set. Naturally, all journeys in a Pareto set are Pareto-optimal. Given source and target vertices $s, t \in \mathcal{V}$ and an earliest departure time τ_{dep}, our objective is to find a Pareto set among all journeys from s to t that depart no later than τ_{dep}.

Algorithms. The algorithms we present in this work are based on RAPTOR [7], which is an algorithm designed to find journeys that optimize arrival time and number of trips on a public transit network with a transitively closed transfer graph. It operates in rounds, where round i finds journeys that use exactly i trips. To achieve this, each round extends journeys found in the previous round by performing a single scan over all routes that are incident to stops reached in the previous round. Transfers are explored between rounds by iterating across the outgoing edges of each reached stop.

Several extensions of RAPTOR have been proposed: McRAPTOR is able to optimize additional criteria by replacing the single label that RAPTOR stores per stop and round with a *bag* of labels that do not dominate each other. rRAPTOR answers *profile queries*, which ask for optimal journeys across an interval of possible departure times, rather than just a single departure time. This is done by performing one iteration of the basic RAPTOR algorithm for each possible departure time, in descending order. Journeys found in earlier iterations can be used to prune non-optimal journeys in later iterations, a technique called *self-pruning*. The MCR [6] algorithm extends (Mc)RAPTOR to work on unrestricted transfer graphs that are not transitively closed. This is done by replacing the edge scans between rounds with Dijkstra searches on a *core graph* created via a partial CH contraction of the transfer graph.

A faster alternative to MCR is the ULTRA [3] preprocessing technique, which precomputes a small number of *transfer shortcuts* that are provably sufficient for computing all Pareto-optimal journeys. Any public transit algorithm that normally requires a transitively closed transfer graph can then use these shortcuts to explore intermediate transfers of a journey. Initial and final transfers are handled separately via Bucket-CH [17, 11, 12], a fast technique for one-to-many queries on road networks. Overall, ULTRA extends public transit algorithms such as RAPTOR to unlimited transfers without a significant performance loss.

3 Algorithms

In this chapter we introduce new algorithms for solving the multi-modal route planning problem with multiple bike sharing operators. We propose two approaches for modeling bike sharing in a public transit network: First we show how MCR can be adapted to handle renting and returning of bicycles explicitly within the algorithm. We call this approach the *operator-dependent* model. We then introduce the *operator-expanded* model, where all relevant information regarding bike sharing is encoded directly in the network. This allows any existing multi-modal public transit algorithm to handle bike sharing without modifications. We demonstrate this on the example of ULTRA. Finally, we introduce a preprocessing technique called *operator pruning* to speed up queries in both models, and describe how it can be incorporated into MCR and ULTRA.

3.1 Operator-Dependent Model/MCR

Most public transit algorithms, including MCR, work by propagating vertex labels through the network. For the two criteria arrival time and number of trips, a label at a vertex v can be represented as a tuple (τ_{arr}, k), where τ_{arr} is the arrival time at v, and k is the number of trips used so far. Bike sharing can be incorporated by extending the labels to triples $(\tau_{\text{arr}}, k, i)$, where i is the operator of the currently rented bike (or 0 if no bike is rented). Since rented bikes have to be dropped off before the end of the journey, only labels at the target vertex t with operator 0 represent complete journeys. Whenever a label $(\tau_{\text{arr}}, k, 0)$ reaches a bike sharing station v, a new label $(\tau_{\text{arr}} + \tau_{\text{pick}}(v), k, i)$ must be created for each operator i with $v \in \mathcal{BS}_i$, to represent the passenger picking up a bike of operator i. Similarly, when a label $(\tau_{\text{arr}}, k, i)$ with $i \neq 0$ reaches a bike sharing station $v \in \mathcal{BS}_i$, a new label $(\tau_{\text{arr}} + \tau_{\text{drop}}(v), k, 0)$ must be created to represent the passenger dropping off the bike. The time required to traverse an edge $e \in \mathcal{E}$ is $\tau_{\text{bike}}(e)$ for labels with a rented bike and $\tau_{\text{walk}}(e)$ otherwise. When propagating a label with operator $i \neq 0$ through a route, any trip T that does not permit bicycle transport (i.e., bt(T) = `false`) must be ignored.

Without bike sharing, a label (τ_{arr}, k) may dominate another label (τ'_{arr}, k') if $\tau_{\text{arr}} \leq \tau'_{\text{arr}}$ and $k \leq k'$ hold. The same dominance rule still applies to labels with the same bike sharing operator: A label $(\tau_{\text{arr}}, k, i)$ dominates a label $(\tau'_{\text{arr}}, k', i)$ if $\tau_{\text{arr}} \leq \tau'_{\text{arr}}$ and $k \leq k'$ hold. However, as the following lemma shows, it is not possible to establish dominance rules for labels with different operators:

▶ **Lemma 1.** *Let $A = (\tau_{arr}, k, i)$ and $B = (\tau'_{arr}, k', j)$ be two labels at some vertex v with $\tau_{arr} \leq \tau'_{arr}$, $k \leq k'$, and $i \neq j$. Then A may not dominate B.*

Proof. If $i = 0$, label B has access to a bike and A does not. Using the bike may allow the journey represented by B to overtake the journey represented by A and reach the target faster. If $i \neq 0$, then the passenger represented by A is carrying a bike, which must be returned before reaching the target. This may require a detour that may not be required for B, possibly allowing the journey represented by B to overtake and reach the target faster. ◀

Thus, bike sharing can be incorporated into the Pareto optimization by treating the bike sharing operator as a third criterion whose values are all incomparable with each other.

In MCR, the number of trips is not stored directly in the labels but unrolled into the round data structure. For the MR-∞ variant of MCR, which only optimizes arrival time and number of trips, it is sufficient to store a single arrival time per vertex and round. Variants with additional criteria replace the single arrival time with a bag of non-dominated labels. When a new label is added, it must be compared to all other labels in the bag to eliminate

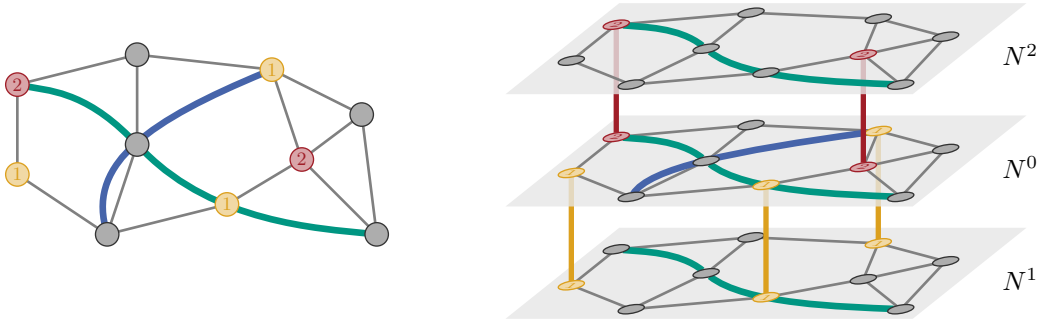

Figure 1 A public transit network (left) and the corresponding operator-expanded network (right). The network features two public transit routes, with the trips of the green route allowing bicycle transport and the trips of the blue route disallowing it. Bikes can be rented and returned at the three bike-sharing stations of operator 1 (yellow) or at the two stations of operator 2 (red).

dominated labels. A naive approach to incorporating bike sharing would be to store bags of labels with two criteria: arrival time and bike sharing operator. However, like the number of transfers, the operator criterion is discrete and only permits a few possible values. Thus, it is more efficient to unroll it into the data structure as well: For each vertex and round, we store an array $(\tau^0_{\text{arr}}, \dots, \tau^o_{\text{arr}})$, where τ^i_{arr} is the best arrival time achieved so far with operator i. As shown by Lemma 1, a new label with operator i only needs to be compared with τ^i_{arr}, since it is incomparable to the other entries. Thus, we can handle all operators independently of each other and do not need bags at all. In each round, we perform $o+1$ independent route scanning phases, where phase i only considers labels with operator i. The resulting algorithm is a variant of MR-∞ whose worst-case running time is proportional to that of the original MR-∞ and $o+1$.

3.2 Operator-Expanded Network/ULTRA

One drawback of the operator-dependent model is that it requires algorithms to be explicitly adapted for bike sharing. An alternative is to unroll the bike sharing information into an enlarged public transit network. Any existing multi-modal public transit algorithm can then handle bike sharing without modification by operating on this *operator-expanded* network.

Given an original public transit network N, the operator-expanded network N^{e} consists of $o+1$ copies $N^0, \dots, N^o$ of N. The idea is that N^i is used by passengers who are currently renting a bike from operator i, with N^0 representing walking. Accordingly, the weight of an edge $e \in \mathcal{E}$ is $\tau_{\text{walk}}(e)$ if it is part of N^0 and $\tau_{\text{bike}}(e)$ otherwise. In all copies N^i with $i \neq 0$, trips that do not allow bicycle transport are removed. The network copies are connected as follows: For a vertex $v \in \mathcal{V}$ in the original network, we denote its copy in network N^i by v^i. For each operator i and each bike sharing station $v \in \mathcal{BS}_i$, the expanded network includes the edges (v^0, v^i) with weight $\tau_{\text{pick}}(v)$ and (v^i, v^0) with weight $\tau_{\text{drop}}(v)$. These edges represent picking up and dropping off a bike, respectively.

Let $\mathcal{OP} = \{0, 1, \dots, o\}$ be the set of bike sharing operators. We then define the operator-expanded network formally as $N^{\text{e}} = (\mathcal{S}^{\text{e}}, \mathcal{T}^{\text{e}}, \mathcal{R}^{\text{e}}, G^{\text{e}} = (\mathcal{V}^{\text{e}}, \mathcal{E}^{\text{e}}))$, with

$$
\begin{aligned}
\mathcal{S}^{\text{e}} &= \{v^i \mid i \in \mathcal{OP} \wedge v \in \mathcal{S}\},\\
\mathcal{T}^{\text{e}} &= \{\mathrm{T}^i = \langle v^i_0, \dots, v^i_k \rangle \mid i \in \mathcal{OP} \wedge \mathrm{T} = \langle v_0, \dots v_k \rangle \in \mathcal{T} \wedge (\mathrm{bt}(\mathrm{T}) \vee i = 0)\},\\
\mathcal{R}^{\text{e}} &= \{\{\mathrm{T}^i \mid \mathrm{T} \in r \wedge (\mathrm{bt}(\mathrm{T}) \vee i = 0)\} \mid i \in \mathcal{OP} \wedge r \in \mathcal{R}\},\\
\mathcal{V}^{\text{e}} &= \{v^i \mid i \in \mathcal{OP} \wedge v \in \mathcal{V}\},\\
\mathcal{E}^{\text{e}} &= \{(v^i, w^i) \mid i \in \mathcal{OP} \wedge (v, w) \in \mathcal{E}\} \cup \{(v^0, v^i), (v^i, v^0) \mid i \in \mathcal{OP} \setminus \{0\} \wedge v \in \mathcal{BS}_i\}.
\end{aligned}
$$

An example of how the operator-expanded network is constructed is shown in Figure 1.

An s-t-query on the original network can be solved with an s^0-t^0-query on the operator-expanded network, using an unmodified multi-modal public transit algorithm (such as MCR) that no longer needs to handle bike sharing explicitly. For ULTRA, both the preprocessing and the query algorithm can be run on the operator-expanded network. The preprocessing, which normally performs profile searches between all vertices, only needs to perform profile searches between the vertices in N^0. As with MCR, ULTRA contracts the transfer graph before the preprocessing is performed. A naive approach would be to create the operator-expanded network first and then contract the expanded transfer graph. However, since the transfer graphs of all networks N^i with $i \neq 0$ are identical, this would lead to redundant work. Instead, it is more efficient to contract two copies of the original transfer graph: one with walking weights and one with biking weights. Bike sharing stations are left uncontracted at this point. These contracted copies can then be inserted into the operator-expanded network in place of the original graph. If desired, an additional contraction can then be performed on the resulting transfer graph of the operator-expanded network.

The ULTRA query consists of two phases: The Bucket-CH search for the initial and final transfers is done on the transfer graph of the operator-expanded network, taking care of bike renting automatically. The main public transit algorithm (e.g., RAPTOR) uses the operator-expanded network with the shortcuts computed by the preprocessing phase. A shortcut (u^i, v^j) corresponds to a shortcut (u, v) in the original network that requires a bike from operator i to access and ends with the passenger having rented a bike from operator j.

3.3 Operator Pruning

A crucial observation that can be used to speed up bike sharing queries is that bike sharing operators typically only serve a limited region (e.g., a single city). Taking a rented bike far outside that region is typically not useful, since the passenger would eventually need to travel back in order to return the bike. Consider a journey that involves taking a train from region A served by operator i to region B served by operator j. If the passenger has rented a bike from operator i, it is usually preferable to return it before taking the train, and then rent a bike from operator j in region B if necessary. However, because labels with different operators may not dominate each other, MCR will also continue exploring the option where the passenger takes the bike from operator i into region B, even though it cannot be returned there. Without additional information, the algorithm will not be aware that the only way to turn this into a valid journey is to travel back to region A, return the bike and then come back to region B, creating an unnecessary detour. To prevent this, we compute an *operator hull* for each operator i, which is a region of the network outside of which it is never useful to travel with a bike from operator i. This allows algorithms to prune journeys once they leave the hull for operator i while carrying a bike from operator i.

Preprocessing. For the hull computation, we define the *cycling network* as $N^{\mathrm{c}} = (\mathcal{S}, \mathcal{T}^{\mathrm{c}}, \mathcal{R}^{\mathrm{c}}, G)$ with $\mathcal{T}^{\mathrm{c}} = \{\mathrm{T} \in \mathcal{T} \mid \mathrm{bt}(\mathrm{T}) = \texttt{true}\}$, $\mathcal{R}^{\mathrm{c}} = \{r \in \mathcal{R} \mid \exists \mathrm{T} \in r\colon \mathrm{bt}(\mathrm{T}) = \texttt{true}\}$, and $\tau_{\mathrm{bike}}(e)$ as the weight for each edge $e \in \mathcal{E}$. This is the network as it appears to a passenger who is using a bike for the entirety of the journey. The *operator hull* $H^i = (\mathcal{V}^i, \mathcal{T}^i)$ for an operator i consists of a set of vertices $\mathcal{V}^i \subseteq \mathcal{V}$ and a set of trips $\mathcal{T}^i \subseteq \mathcal{T}$ such that every journey in N^{c} between bike sharing stations $s, t \in \mathcal{BS}_i$ is dominated by a journey in N^{c} that only uses vertices in $\mathcal{V}^i$ and trips in $\mathcal{T}^i$. It can be computed by running a profile variant of MCR (without bike sharing) on N^{c} from each station $s \in \mathcal{BS}_i$ and unpacking all found journeys ending at another station $t \in \mathcal{BS}_i$. The individual profile searches can be sped up with a simple pruning rule: A profile search from a source station s starts by exploring the initial transfers, computing for each vertex $v \in \mathcal{V}$ the biking time $\tau_{\mathrm{bike}}(s, v)$ from s to v.

From this we can compute $\tau_{\text{bike}}^{\max} := \max_{t \in \mathcal{BS}_i} \tau_{\text{bike}}(s,t)$, which is the maximum biking time to any other bike sharing station of the same operator. Since no optimal journey in N^{c} that ends at a station in $\mathcal{BS}_i$ may be longer than this, the profile search can prune labels whose travel time exceeds $\tau_{\text{bike}}^{\max}$.

Note that the operator hull is an overapproximation of the region outside of which it is never useful to travel with a bike: Journeys that are optimal in N^{c} and therefore contribute to the operator hull may be dominated by journeys that require switching between different bike operators or dropping off a bike to take a trip without bicycle transport. These journeys could be excluded from the hull by using MCR with bike sharing for the hull computation. While this might lead to smaller hulls, it would require significantly higher computation times. Moreover, because the hull computation for one operator would no longer be independent of the other operators, any change in the set of bike sharing stations for one operator would necessitate recomputing all hulls, rather than just the one for the changed operator.

Combination with MCR. Incorporating operator pruning into MCR is straightforward: During the Dijkstra searches, labels with operator i are not propagated to vertices that are not in $\mathcal{V}^i$. Similarly, during the route scanning phase for operator i, routes with no trips in $\mathcal{T}^i$ are ignored and arrival times at stops not in $\mathcal{V}^i$ are not updated. While it would be possible to skip trips that are not in $\mathcal{T}^i$ during the individual route scans, this has no performance benefit since RAPTOR always scans routes until the last stop. Thus, if a trip is skipped, the route scan will simply continue with the next reachable trip.

Combination with Operator-Expanded Network/ULTRA. Given an operator hull $H^i = (\mathcal{V}^i, \mathcal{T}^i)$, we define the *hull network* $N_H^i = (\mathcal{S}^i, \mathcal{T}^i, \mathcal{R}^i, G^i)$ with $\mathcal{S}^i = \mathcal{S} \cap \mathcal{V}^i$, $\mathcal{R}^i \subseteq \mathcal{R}$ the set of routes for which at least one trip is contained in $\mathcal{T}^i$, and G^i the subgraph of G that is induced by $\mathcal{V}^i$. In order to incorporate operator pruning into the operator-expanded network, we first compute operator hulls on the original, non-expanded network. For each operator $i \neq 0$, we then replace the network copy N^i with the hull network N_H^i. The network copy N^0 that represents walking is left unchanged. In the resulting network, leaving the operator hull for the currently rented bike is no longer possible because the corresponding parts of the network have been deleted. Accordingly, any algorithm that runs on this network (including ULTRA) will automatically benefit from operator pruning.

3.4 Extended Scenarios

Free-Floating Bike Sharing. Some bike sharing operators use a *free-floating* (or *dockless*) sharing system without fixed stations, where bikes can be picked up or dropped off at any location within the served region. This can be handled by considering every vertex in the region as a bike sharing station. Unlike with fixed bike sharing stations, this scenario is inherently dynamic: Bikes are not available at every vertex in the region, and it is not known in advance where bikes will be located. Therefore, precomputation techniques such as ULTRA are not applicable. MCR can handle this by checking explicitly whether a bike is available when arriving at a vertex. Operator pruning can also be adapted by running the profile variant of MCR from each boundary vertex of the region, in addition to including all vertices and trips within the region in the hull.

Fixed Pickup Stations with Free-Floating Dropoff. We also consider a hybrid system where pickup is restricted to fixed stations but dropoff is allowed at any location. As with the fully station-based system, we assume that a bike is always available at every station. This makes ULTRA feasible again. Under the reasonable assumption that $\tau_{\text{bike}}(e) \leq \tau_{\text{walk}}(e)$ holds

Table 1 Sizes of the used public transit networks and their bike sharing systems.

Network	Stops	Routes	Trips	Vertices	Edges	Stations	Operators
London	20 595	2 107	125 436	183 025	579 888	823	4
Switzerland	25 426	13 934	369 534	604 167	1 847 140	534	11
Germany	244 055	231 089	2 387 297	6 872 105	21 372 360	2 682	22

for every edge $e \in \mathcal{E}$, it only makes sense to drop off a bike at specific vertices: A bike from operator i may be dropped off at a stop (in order to enter a trip without bicycle transport), a pickup station from a different operator j (in order to switch operators), a boundary vertex of the served region (when leaving the region), or the target vertex. Dropping off the bike at any other vertex would incur unnecessary walking costs. For each such vertex v, the edge (v^i, v^0) is added to the operator-expanded network. Since the target vertex t changes with each query, the edge (t^i, t^0) is only inserted temporarily at query time.

Biking as Additional Trip. In the original MCR publication [6], which considered bike sharing with only one operator, each bike that was used in a journey was counted as an additional trip. Incorporating this into our adapted version of MCR is straightforward. The operator pruning technique can be applied without changes, since its preprocessing only considers journeys that use a single bike for the entire journey. Adapting ULTRA, however, would require fundamental changes because it is based on enumerating all journeys with exactly two trips. If bike usage is counted as an additional trip, two trips are no longer sufficient for finding all relevant transfers.

4 Experiments

All algorithms were implemented in C++17 compiled with GCC version 8.2.1 and optimization flag -O3. All experiments were conducted on a machine with two 8-core Intel Xeon Skylake SP Gold 6144 CPUs clocked at 3.5 GHz, with a turbo frequency of 4.2 GHz, 192 GiB of DDR4-2666 RAM, and 24.75 MiB of L3 cache.

Benchmark Data. We evaluated our algorithms on the transportation network of London, which was also used to evaluate the MCR algorithm [6], as well as the networks of Switzerland and Germany as introduced in [22]. For all three instances, we combined the public transit networks with transfer graphs representing walking and cycling that were extracted from OpenStreetMap[1]. In order to obtain travel times, we assumed an average walking speed of 4.5 km/h and an average cycling speed of 20 km/h. However, the cycling speed was reduced to the speed limit for streets where the speed limit is below 20 km/h. The locations of bike sharing stations were also extracted from OpenStreetMap. We assigned each bike sharing station to an operator based on the information available (e.g., identifying different spellings of the same company). For stations that were not annotated with any information, we chose an operator of other nearby stations at random. Operators with only one bike sharing station were dropped from the dataset, as they are irrelevant for routing. For reproducibility we make the resulting cleaned bike sharing station dataset available[2]. For our experiments we assume a pickup time of 20 seconds and a dropoff time of 10 seconds for all bike sharing stations. An overview of the networks is given in Table 1.

[1] https://download.geofabrik.de/
[2] https://i11www.iti.kit.edu/PublicTransitData/BikeSharing/

Table 2 Overview of all preprocessing steps. We report the sizes of the precomputed data structures as well as the required computation time. Columns labeled with OE contain results for the full operator-expanded network; columns labeled with OE-OP represent the operator-expanded network with operator pruning. The *Total* row corresponds to the preprocessing time of ULTRA-OE(-OP). Entries marked with ? are not reported as their computation would take several weeks. We report single core running times with the exception of two rows marked with (16), where 16 cores were used.

		London		Switzerland		Germany	
	Measured value	OE	OE-OP	OE	OE-OP	OE	OE-OP
Results	Cycling core vertices	–	26 742	–	37 779	–	435 751
	Operator hull vertices	–	13 735	–	19 018	–	351 224
	Expanded stops	102 975	31 216	301 500	36 892	5 613 265	411 980
	Expanded vertices	290 791	197 558	1 019 779	623 228	16 461 221	7 225 923
	Expanded edges	2 081 218	748 938	7 673 596	2 002 615	155 594 242	24 236 935
	ULTRA shortcuts	1 831 779	521 882	3 389 309	435 514	?	7 873 379
Running time [h:m:s]	Walking Core-CH	–	0:06	–	0:28	–	6:36
	Cycling Core-CH	0:06	0:06	0:28	0:28	6:43	6:43
	Operator hulls	–	3:01:21	–	50:20	–	83:38:15
	Operator hulls (16)	–	15:34	–	4:15	–	8:45:22
	Expanded Core-CH	0:08	0:07	0:34	0:29	8:38	7:07
	ULTRA shortcuts (16)	14:14:51	43:31	9:59:43	21:50	?	30:50:13
	Expanded Bucket-CH	0:14	0:09	1:09	0:33	?	17:47
	Total	14:15:19	59:33	10:01:54	28:03	?	40:13:48

4.1 Preprocessing

All algorithms discussed in this work require some form of preprocessing. The most elaborate preprocessing steps are the operator hull computation as well as the ULTRA shortcut computation. In addition to these two steps, several CH computations are required. An overview of the preprocessing steps and their results is given in Table 2.

Regarding the operator hull computation, we observe that the number of vertices contained in the union of all hulls is significantly smaller than the size of the corresponding core graph. This means that some parts of the network will never be visited with a rented bike. The small hulls also have a direct impact on the expanded networks, as their size is also significantly reduced when operator hulls are applied. Using 16 cores, hulls for the small networks can be computed in a few minutes, while Germany requires less than 9 hours. For the networks of London and Switzerland, this corresponds to a speedup factor of 12 compared to a sequential computation. For the Germany network, we only achieve a speedup of 9.5. In order to explain this effect, we measured the average number of instructions executed per CPU cycle. For the sequential hull computation on the Germany network, we recorded a value of 1.1, while it was only 0.9 for the parallel computation. These numbers suggest that the reduced speedup observed for the Germany network is due to an overloaded memory system.

The impact of the operator hulls on the ULTRA shortcuts is also quite strong. On the London and Switzerland networks, operator hulls reduce the preprocessing time by a factor of 14 and 20, and the number of shortcuts by a factor of about 4 and 8, respectively. For the Germany network, ULTRA is only viable with operator hulls. Without operator hulls, only 4.7% of the shortcut computation was finished after one week. We therefore estimate that the complete shortcut computation would take about 21 weeks.

Table 3 Performance overview of all approaches described in this work. All algorithms are evaluated with and without the use of operator pruning (OP). The MR-∞ algorithm is evaluated for both the operator-dependent (OD) and the operator-expanded (OE) model, while ULTRA can only be used with the operator-expanded model. The query results are averages over 10 000 random queries. The Vertices (Routes) column reports the average number of vertices (routes) settled (scanned) by the algorithm.

Network	Algorithm	Preprocessing	Query			
		Time [h:m:s]	Rounds	Vertices	Routes	Time [ms]
London	MR-∞-OD	0:12	9.59	342 361	25 037	112.2
	MR-∞-OD-OP	15:46	8.90	135 765	10 884	51.1
	MR-∞-OE	0:14	9.59	320 286	25 045	119.1
	MR-∞-OE-OP	15:53	8.64	117 188	9 152	34.2
	ULTRA-OE	14:15:19	9.70	78 486	25 922	52.8
	ULTRA-OE-OP	59:33	8.75	23 534	9 532	17.1
Switzerland	MR-∞-OD	0:56	9.55	840 396	171 361	286.8
	MR-∞-OD-OP	5:11	8.49	176 364	54 173	85.0
	MR-∞-OE	1:02	9.55	782 572	171 410	345.0
	MR-∞-OE-OP	5:40	8.35	144 522	43 980	52.8
	ULTRA-OE	10:01:54	9.70	107 627	180 064	117.2
	ULTRA-OE-OP	28:03	8.48	29 394	44 970	21.0
Germany	MR-∞-OD	13:19	11.99	17 421 659	2 888 893	9 830.1
	MR-∞-OD-OP	8:58:41	10.62	2 689 029	706 307	2 183.9
	MR-∞-OE	15:21	11.99	16 120 342	2 889 313	10 599.3
	MR-∞-OE-OP	9:05:48	10.24	2 091 814	679 898	1 322.7
	ULTRA-OE-OP	40:13:48	10.38	301 832	688 525	649.3

4.2 Queries

To evaluate the impact of operator pruning and the differences between the operator-dependent and operator-expanded models, we evaluated all algorithms presented in this work on 10 000 random queries. An overview of the results is given in Table 3.

We use MCR, or more specifically its MR-∞ variant, to compare the operator-dependent and operator-expanded model, as MR-∞ can be used with both models. Without operator pruning, both models perform similarly. This is to be expected, as the operator-dependent algorithm more or less simulates what the standard algorithm does on the operator-expanded model. The number of rounds is exactly the same for both approaches, and the small deviations in the number of settled vertices and scanned rounds can be explained by differences in the respective core graphs and a slightly better target pruning in the dependent model. Still, the operator-dependent model is slightly faster, as it has less memory usage.

Operator pruning improves query times significantly, with the speedup ranging from 2.2 on the operator-dependent London network to 8.0 on the operator-expanded Germany network. Naturally, the speedup is greater on larger networks with more bike sharing operators. Approximately one fewer round is performed on average as a result of reducing the search space. Moreover, the operator-expanded model benefits more strongly from operator pruning than the operator-dependent model, being faster by a factor of 1.5 to 1.7. This is because vertices and trips that are not part of the operator hull are removed entirely from the network instead of being skipped at query time.

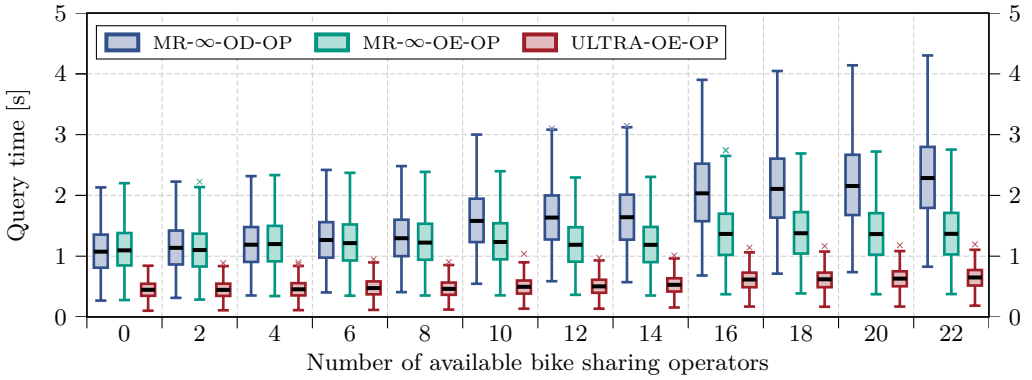

Figure 2 Running time of all query algorithms with operator pruning depending on the number of bike sharing operators available. We used the Germany network without bike sharing and gradually added the bike sharing operators in sets of two (in order to compensate for differences in the number of stations). We evaluated the same 1 000 random queries for each number of bike sharing operators.

Combining ULTRA-RAPTOR with the operator-expanded model reduces query times even further. Compared to the base approach of using MR-∞ on the operator-dependent network, we achieve speedup factors of 6.6, 13.7, and 15.1 for the networks of London, Switzerland, and Germany, respectively. Furthermore, ULTRA-RAPTOR is the first algorithm that enables query times below a second for all networks.

Impact of Additional Bike Sharing Operators. We evaluated how the number of available bike sharing operators influences the query time of the algorithms. For this we used the Germany network, as it is the largest network and has the most bike sharing operators. We started without any bike sharing operators and created partial instances by successively adding operators. To compensate for differences in the number of bike sharing stations, we added operators in pairs of two, pairing large operators with smaller ones. Finally, we evaluated the same set of randomly picked long-range queries for all networks. The results are depicted in Figure 2. We observe that for all algorithms the number of operators is correlated linearly to the query time. The impact on the query time is the strongest for MR-∞-OD-OP, further confirming that the operator-expanded model benefits more from operator pruning than the operator-dependent model.

5 Conclusion

We presented two novel approaches for modeling multi-modal transportation networks with various competing bike sharing operators: the operator-dependent and operator-expanded model. We showed that both models result in similar query performance, with the operator-dependent model being more memory-efficient and the operator-expanded model being compatible with existing query algorithms without modifications. Given its compatibility, we were able to combine the operator-expanded model with ULTRA, a known speedup technique for multi-modal networks, in order to reduce query times. Additionally, we developed a fast preprocessing step called operator pruning, which can be used to accelerate queries in both models. Our experimental evaluation shows that combining operator pruning with ULTRA-RAPTOR enables queries that are more than an order of magnitude faster than the operator-dependent variant of MCR. Beyond that, we showed that using operator pruning also reduces the preprocessing time of ULTRA by more than an order of magnitude.

References

1 Hannah Bast, Erik Carlsson, Arno Eigenwillig, Robert Geisberger, Chris Harrelson, Veselin Raychev, and Fabien Viger. Fast Routing in Very Large Public Transportation Networks using Transfer Patterns. In *European Symposium on Algorithms*, pages 290–301. Springer, 2010.

2 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route Planning in Transportation Networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

3 Moritz Baum, Valentin Buchhold, Jonas Sauer, Dorothea Wagner, and Tobias Zündorf. UnLimited TRAnsfers for Multi-Modal Route Planning: An Efficient Solution. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2019.14.

4 Annabell Berger, Daniel Delling, Andreas Gebhardt, and Matthias Müller-Hannemann. Accelerating Time-Dependent Multi-Criteria Timetable Information is Harder than Expected. In *OASIcs-OpenAccess Series in Informatics*, volume 12. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.

5 Dominik Bucher, David Jonietz, and Martin Raubal. A Heuristic for Multi-Modal Route Planning. In *Progress in Location-Based Services 2016*, pages 211–229. Springer, 2017.

6 Daniel Delling, Julian Dibbelt, Thomas Pajor, Dorothea Wagner, and Renato Werneck. Computing Multimodal Journeys in Practice. In *International Symposium on Experimental Algorithms*, pages 260–271. Springer, 2013.

7 Daniel Delling, Thomas Pajor, and Renato F Werneck. Round-based Public Transit Routing. *Transportation Science*, 49(3):591–604, 2014.

8 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly Simple and Fast Transit Routing. In *International Symposium on Experimental Algorithms*, pages 43–54. Springer, 2013.

9 Julian Dibbelt, Thomas Pajor, and Dorothea Wagner. User-Constrained Multimodal Route Planning. *ACM Journal of Experimental Algorithmics*, 19:3.2:1–3.2:19, April 2015.

10 Edsger W Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1):269–271, 1959.

11 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, June 2008.

12 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact Routing in Large Road Networks Using Contraction Hierarchies. *Transportation Science*, 46(3):388–404, August 2012.

13 Kalliopi Giannakopoulou, Andreas Paraskevopoulos, and Christos Zaroliagis. Multimodal Dynamic Journey-Planning. *Algorithms*, 12(10):213, 2019.

14 Andrew V Goldberg and Chris Harrelson. Computing the Shortest Path: A* Search meets Graph Theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005.

15 Jan Hrnčíř and Michal Jakob. Generalised Time-Dependent Graphs for Fully Multimodal Journey Planning. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 2138–2145. IEEE, 2013.

16 Dominik Kirchler. *Efficient Routing on Multi-Modal Transportation Networks*. PhD thesis, Ecole Polytechnique X, 2013.

17 Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36–45. SIAM, 2007.

18 Matthias Müller-Hannemann and Mathias Schnee. Finding all Attractive Train Connections by Multi-Criteria Pareto Search. In *Algorithmic Methods for Railway Optimization*, pages 246–263. Springer, 2007.

19 Matthias Müller-Hannemann, Frank Schulz, Dorothea Wagner, and Christos Zaroliagis. Timetable Information: Models and Algorithms. In *Algorithmic Methods for Railway Optimization*, pages 67–90. Springer, 2007.

20 Stefano Pallottino and Maria Grazia Scutella. Shortest Path Algorithms in Transportation Models: Classical and Innovative Aspects. In *Equilibrium and Advanced Transportation Modelling*, pages 245–281. Springer, 1998.

21 Sabine Storandt. Route Planning for Bicycles – Exact Constrained Shortest Paths made Practical via Contraction Hierarchy. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

22 Dorothea Wagner and Tobias Zündorf. Public Transit Routing with Unrestricted Walking. In *17th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2017.

23 Sascha Witt. Trip-Based Public Transit Routing. In *Algorithms-ESA 2015*, pages 1025–1036. Springer, 2015.