Partition of Time-Dependent Routes into Fixed-Size Vehicles

Bachelor's Thesis of

Marco Rieger

At the Department of Informatics
Institute of Theoretical Informatics

Reviewer:	T.T.-Prof. Dr. Thomas Bläsius
Second reviewer:	Dr. rer. nat. Torsten Ueckerdt
Advisors:	Adrian Feilhauer
	Michael Zündorf

20.06.2025 – 20.10.2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, 20.10.2025

Marco Rieger

(Marco Rieger)

Abstract

As demand for private traffic increases, roads are increasingly congested by vehicles serving only a single person. However, multiple people traveling along similar routes, could instead use a shared vehicle for their trip, which motivates the Vehicle Sharing Fixed Routes Problem in a road network, for which we introduce a problem formulation. In this problem all vehicles have the same capacity and no transfers between vehicles are allowed. All passengers want to travel along a fixed route, and every vehicle used is owned by a passenger who uses it from their origin to their destination. Along their route, they can fill the vehicle's capacity with other passengers. We show that finding an assignment that minimizes the total vehicle operation time is NP hard, even if we restrict the road network to a simple directed path with unit distance edges. Afterwards, we consider the scenario in which there is only one driver who can transport other passengers. We present an algorithm that solves this scenario in $O(nC \log n)$ for an arbitrary vehicle size $C \geq 2$ by reducing it to Maximal Weight Fixed Interval Scheduling.

Zusammenfassung

Mit steigender Nachfrage nach Individualverkehr, werden Straßen zunehmend durch Fahrzeuge verstopft, die lediglich einer Person dienen. Allerdings könnten mehrere Personen, deren Routen sich ähneln, ein gemeinsames Fahrzeug für ihre Fahrt nutzen. Dies motiviert das Vehicle Sharing Fixed Routes Problem, wobei alle geteilten Fahrzeuge dieselbe Kapazität haben und keine Umstiege zwischen geteilten Fahrzeugen erlaubt sind. In dem Problem ist für jede Person eine feste Route vorgegeben, von der nicht abgewichen werden darf, und jedes geteilte Fahrzeug gehört einer Person, welche das Fahrzeug vom Anfang bis zum Ende ihrer Route verwendet. Während der Fahrt kann die nicht verwendete Fahrzeugkapazität mit anderen Personen besetzt werden, die im geteilten Fahrzeug mitfahren. Wir zeigen, dass es NP-schwer ist, eine Zuweisung von Personen in Fahrzeuge zu finden, welche die Gesamtfahrzeit minimiert, selbst wenn wir das Straßennetz auf einen einfachen gerichteten Pfad, in dem jede Kante dieselben Kosten hat, beschränken. Danach betrachten wir ein Szenario, in dem es nur einen Fahrer gibt, der andere Personen transportieren kann. Für das Szenario finden wir einen Algorithmus, der das Problem in $O(nC \log n)$ für beliebige Fahrzeuggrößen $C \geq 2$ löst.

Contents

1 Introduction

In many urban areas, congested roads are a common problem [SEL19]. Many vehicles serve only a single person [DB22], which is neither very space efficient nor environmentally friendly. A possible approach to tackling this issue would be to group multiple people who would otherwise use their own vehicles into the same shared vehicle. However, one cannot arbitrarily group people with completely different origins and destinations into the same vehicle, or the length of their trips would significantly increase. Therefore, for vehicle sharing to make sense, the involved passengers should share at least some sections of their routes. There already exists an algorithm that attempts to redirect traffic demand onto shared routes to enable vehicle sharing [Blä+25]. However, it raises the question of how to distribute the passengers into shared vehicles. As these routes are intended to be set up in a way that enables vehicle sharing, we assume that we cannot alter the passengers' routes. We aim to find an assignment of the routes into shared vehicles that minimizes vehicle operation time. Because vehicles have a limited capacity, we want to model this and assume that all vehicles we use have a fixed capacity C. We do not allow transferring between vehicles, as this realistically causes additional delays. If we allow it with no imposed penalties, we could simply use separate vehicles for each edge to obtain an optimal solution. We assume that each vehicle used is owned by one of the passengers who uses the vehicle from the origin of their route to their destination. The problem can also be viewed as searching for an optimal configuration of carpools. A precise definition of the Vehicle Sharing Fixed Routes Problem is in Chapter 2. Originally the intention was to possibly make solving the problem more difficult by introducing temporal constraints, which would be present in a realistic environment, on each route. However, in Chapter 3, we show that even if we assume the road network is a simple directed path with unit distance edges and do not use any temporal constraints, finding an optimal solution is NP hard, even though we know which passengers serve as drivers for other passengers. The difficulty appears to lie in the competition among drivers regarding which passengers they get to transport. In Chapter 4, we assume that there is only one driver who can serve the other passengers. That scenario is closely related to the Maximal Weight Fixed Interval Scheduling problem, which has been studied more extensively [AS87 | CL95 | BE96 | KNC07]. We obtain an algorithm that runs in $O(n \log n)$ for vehicle size $C = 2$ and in $O(nC \log n)$ for an arbitrary vehicle size C.

2 Preliminaries

In a directed graph $G = (V, E)$, a walk is a sequence of edges in G, such that the endpoint of the previous edge is the starting point of the next edge. A path is a walk, in which all vertices are distinct. We say G is a directed path if there exists a path that traverses all vertices in V and all edges in E. If a walk/path A contains another walk/path B, we say B is a subwalk/subpath of A.

The goal is to group passengers with shared routes into shared vehicles that have a fixed capacity in a way that minimizes the total vehicle operation time. The route a passenger wants to travel along cannot be modified. Every vehicle used is owned by a passenger who uses it from their origin to their destination. Passengers can use other vehicles as long as the vehicle capacity is not exceeded. However, they are not allowed to transfer between vehicles. We call this the Vehicle Sharing Fixed Routes Problem.
We are working on a directed graph $G = (V, E)$ with a cost function $c : E \to \mathbb{R}$. Let $P = \{p_1, ..., p_k\}$ be the set containing the walks of the routes. Let C be the fixed capacity of each shared vehicle. We are looking to partition P into disjoint sets $S_1, ...S_\ell$ that each represent a shared vehicle. All walks in the set S_i must be a subwalk of the set's representative $r_i \in S_i$ that owns the vehicle. We say that the set's representative transports all other routes in the set. Let L_i refer to the number of edges traversed by r_i. For every vehicle, we want an assignment $u_i : S_i \to [1, L_i]^2$, where $u_i(p) = [a, b]$ with $a < b$ means that the vehicle transports p along the edges of r_i from the a-th edge until the b-th edge. $u_i(p) = [a, b]$ can be seen as an interval containing the edge indices from a to b, where an edge has an edge index of c if it is the c-th edge of r_i. The subwalk of r_i from the a-th to the b-th edge must exactly match the edges of p. The vehicle capacity C may not be exceeded along any edge, which means that every edge index may be inside at most C intervals. Note that $u_i(r_i) = [1, L_i]$, so one slot is always occupied by the passenger who owns the vehicle. The goal is to minimize the total vehicle operation time.

In the next chapter, we show how to reduce SAT to Vehicle Sharing Fixed Routes Problem. An instance of SAT contains a variable set $U = \{u_1, ..., u_m\}$ and a clause set $K = \{k_1, ..., k_n\}$. For every variable $x \in U$, there is a positive literal x and a negative literal $\overline{x}$. Every clause is a logical disjunction of literals. For example, the clause $k_1 = u_1 \vee \overline{u_3} \vee u_7$ is satisfied when u_1 or u_7 are true, or when u_3 is false. SAT asks whether there exists an interpretation of the variables that satisfies all clauses.
For ease of notation, we say $[n]$ for the set containing all natural numbers from 1 to n.

3 NP Hardness for Paths

We now show how to reduce SAT to Vehicle Sharing Fixed Routes Problem for vehicle size $C = 2$, even if we constrain the underlying graph to be a directed path with all edge costs equal to 1.
Given an instance of SAT, we construct a graph with these properties and define routes on that graph such that SAT has a solution if and only if there is a solution to the Vehicle Sharing Fixed Routes Problem in which the total vehicle operation time is below a certain threshold.
For ease of notation, we use integer edge costs in the range from 1 to 4. However, this graph can be transformed into a unit graph by replacing each edge with as many edges as it costs.

Let (U, K) be an instance of SAT, where $U = \{u_1, ..., u_m\}$ is the variable set and $K = \{k_1, ..., k_n\}$ is the clause set.
We now define a graph based on the SAT instance and three kinds of routes on that graph. Literal routes, variable routes, and clause routes. The idea is that, in an optimal solution, variable routes pick their respective negative or positive literal and therefore exhaust that literal. This leaves only the opposite literal, the value that would be true in a possible SAT solution, for the clause routes. Literal routes form the foundation and are set up in a way that variable routes must transport all literal routes corresponding to either their positive or their negative literal.
In that graph, literal routes go from each of the vertices that represent this literal to the subsequent vertex representing this literal. This graph can be divided into the following subgraphs:
The first subgraph G_{Init} and the last subgraph G_{End} are both directed paths of length $2m$, where each vertex represents one of the literals. The order used for this is $u_1, \overline{u_1}, u_2, \overline{u_2}, ..., u_m, \overline{u_m}$. To distinguish these vertices, we reference the structure to which they belong in the superscript. In G_{Init}, we call the vertices ${u_1}^{\text{Init}}$ to $\overline{u_m}^{\text{Init}}$, and in G_{End}, we call the vertices ${u_1}^{\text{End}}$ to $\overline{u_m}^{\text{End}}$.
The cost of the edges in G_{Init} and G_{End} is defined as follows: For edges going from the respective positive literal to its negative literal, the cost of the edges is 1. For edges going from a negative literal to the next positive literal, the cost of the edges is 3.
Figure 3.1 shows how G_{Init} and G_{End} are structured.
Now, to define the clause routes, there is one subgraph G_{k_i} for each clause $k_i \in K$: Every G_{k_i} is a directed path that consists of a start vertex s_i, which is the beginning of the associated clause route. That vertex is followed by $2m$ vertices that we call in-vertices, where each vertex represents a unique literal. These vertices are followed by $2m$ vertices that we call out-vertices, where again, each vertex represents a unique literal. Finally, the path ends at an end vertex t_i, where the associated clause route ends. For each clause subgraph, there is exactly one clause route. For the vertices representing literals, we use the order $u_1, \overline{u_1}, u_2, \overline{u_2}, ..., u_m, \overline{u_m}$ again. For distinctiveness, we call the in-vertices ${u_1}^{k_i,\text{in}}$ to $\overline{u_m}^{k_i,\text{in}}$ and the out-vertices ${u_1}^{k_i,\text{out}}$ to $\overline{u_m}^{k_i,\text{out}}$.
For the edge costs in G_{k_i}, the idea is that the literal routes inside the clause all have the same length $2m \cdot 3 = 6m$, unless the associated literal is used in the clause. In that case, the path should be slightly longer $(6m+1)$. Therefore, it is a better solution if the clause route transports

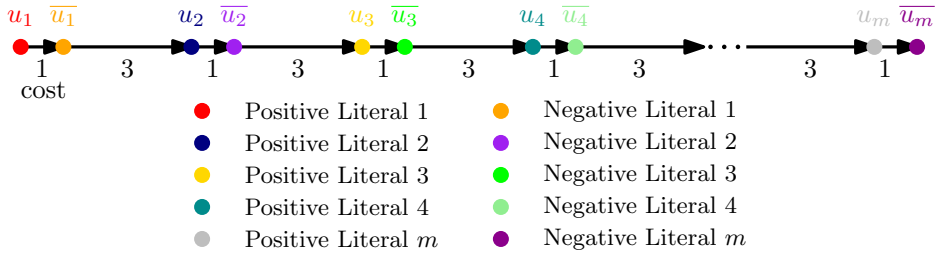

Figure 3.1: Structure of G_{Init} and G_{End}. The edges have different costs, so that the 1 cost edges right at the start or at the end of a variable route are the only ones that can be skipped when transporting literal routes, while achieving the maximum sharing value.

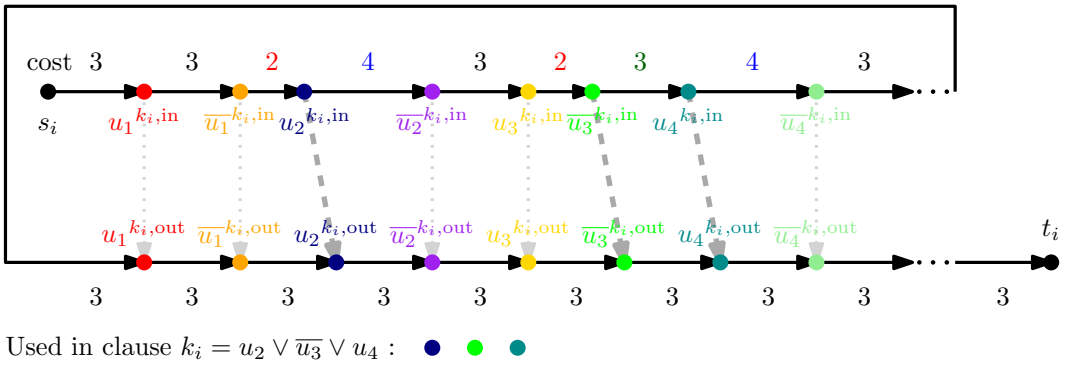

Figure 3.2: Structure of a G_{k_i}. In this example clause it can be observed how the edge cost is modified based on whether a literal is used in a clause. Literal routes for literals used in the clause are slightly longer than the other literal routes. Therefore, the clause route that starts at s_i and ends at t_i achieves a higher sharing value, when transporting one of these slightly longer literal routes.

a route corresponding to a literal that is used in this clause.
The cost of an edge going from a to b inside G_{k_i} is defined as follows: If neither a nor b are in-vertices for literals used in k_i, then the cost of the edge is 3. If only a is an in-vertex for a literal used in k_i, then the cost of the edge is 4. If only b is an in-vertex for a literal used in k_i, then the cost of the edge is 2. If both a and b are in-vertices for literals used in k_i, then the cost of the edge, once again, is 3, as their effects cancel out.
Effectively, we maintain an average edge cost of 3 but shorten the edges that go to in-vertices for literals used in the clause by 1 and lengthen the subsequent edge by 1. Figure 3.2 visualizes this with weights based on an example clause $k_i = u_2 \vee \overline{u_3} \vee u_4$.
We can combine these paths into the directed path $G = (V, E)$ by linking these subgraphs in the order $G_{\text{Init}}, G_{k_1}, ..., G_{k_n}, G_{\text{End}}$. For the edges that link these subpaths, the cost is defined as 3. You can see the complete structure of G in Figure 3.3.
For every variable u_i, we define a variable route that starts at the first instance of its positive literal u_i^{Init} and ends at the last instance of its negative literal $\overline{u_i}^{\text{End}}$. Figure 3.4 illustrates which routes can be transported by a variable route, so that they transport a route along all but one edge.

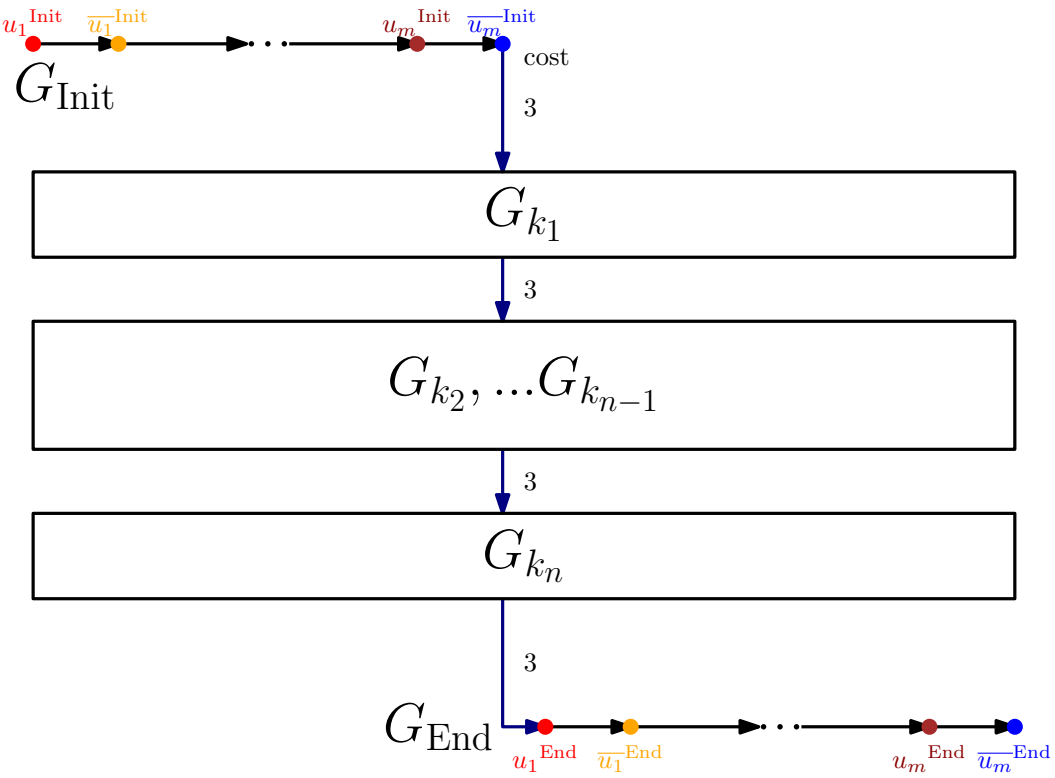

Figure 3.3: Structure of G. The edges connecting all the subgraphs cost 3 each. See Figure 3.1 for the edge costs in G_{Init} and in G_{End}. See Figure 3.2 for how the clause subgraphs are structured.

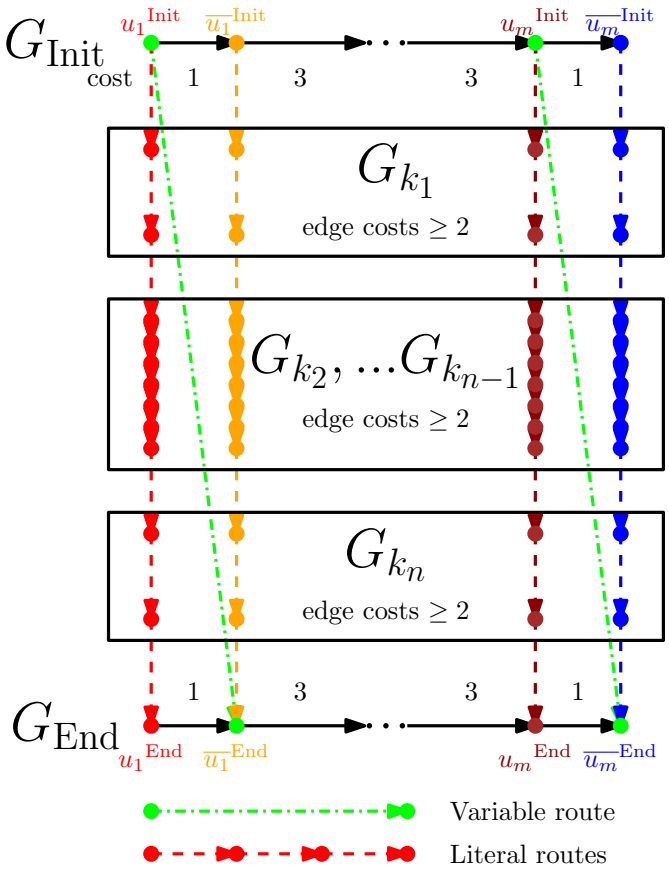

Figure 3.4: Visualization of variable routes. Refer to Figure 3.3 for the more precise graph structure. This figure illustrates the opportunities variable routes have to transport other routes, while there is at most one edge where they do not transport any route. For there to be only one edge, along which they transport no route, they have to first transport positive literal routes and then eventually swap over to negative literal routes.

Now that an instance of the Vehicle Sharing Fixed Routes Problem has been constructed, we want to show how we can decide whether the SAT instance (U, K) has a solution if we can solve the Vehicle Sharing Fixed Routes Problem. The proof is divided into multiple lemmas, making it easier to follow. Instead of directly calculating vehicle operation time, we aim to determine how much vehicle operation time is saved when a route transports other routes compared to all of them using separate vehicles. We define the sharing value of a route as the cost of all routes that are transported by it, so the vehicle operation time that is saved by this assignment. Because for every literal route there is no other route that is a subwalk, only clause routes and variable routes can transport other routes. We begin by calculating which sharing value variable routes can achieve. For that, we calculate the cost of a variable route and then count how expensive the edges are along which no route can be transported in a solution.

Lemma 3.1: *All variable routes have the cost* $n(12m + 6) + 4m + 1$.

Proof. All variable routes use the $n(4m + 2) - 1$ edges inside and connecting the clause subpaths. The cost of the edges inside a clause subpath is either 2, 3, or 4. By construction, there is an edge costing 4 for every edge costing 2 and vice versa. This results in a cost of $3(n(4m + 2) - 1) = n(12m + 6) - 3$. They also use the edge connecting the last clause subpath G_{k_n} to G_{End}, as well as the edge connecting G_{Init} and the first clause subpath G_{k_1}, which both cost 3. The i-th variable route uses $m - i$ edges with cost 3, as well as $m - i + 1$ edges with cost 1 in G_{Init}, and $i - 1$ edges with cost 3, as well as i edges with cost 1 in G_{End}. So, in total, we get a cost of

$$\begin{aligned}
&(n(12m + 6) - 3) + 2 \cdot 3 + 3(m - i) + (m - i + 1) + 3(i - 1) + i \\
&= n(12m + 6) + 3(m - i + i - 1 + 2 - 1) + (m - i + 1 + i) \\
&= n(12m + 6) + 3m + m + 1 \\
&= n(12m + 6) + 4m + 1.
\end{aligned}$$

∎

Now, we show that for every variable route, there must be at least one edge along which they transport no route. We show that the only edge along which no route is transported can cost 1, which is only achieved if the variable route transports either all its respective negative literal routes or all its respective positive literal routes. As the respective negative and positive literal routes are different for all variable routes, there is no competition between them. This means that, in an optimal solution, variable routes correctly block either all their positive or all their negative literal routes from being transported by a clause route.

Lemma 3.2: *The highest achievable sharing value for a variable route is* $n(12m + 6) + 4m$. *They achieve that sharing value if and only if they either transport all respective negative literal routes or all respective positive literal routes.*

Proof. By Lemma 3.1, all variable routes have the cost $n(12m + 6) + 4m + 1$. The lowest cost of any edge in the graph is 1. Therefore, to achieve a sharing value higher than $n(12m + 6) + 4m$, the route would have to transport another route along every edge. The only route starting at the beginning of a variable route is the first route corresponding to its positive literal. Whenever any literal route ends, the only other route that begins at that vertex is the next literal route corresponding to the same literal. At the same time, the only route ending at the end of a variable route is the last route corresponding to its negative literal. Therefore, there

must be at least one edge where no route is transported by a variable route. See Figure 3.4 for an illustration. To achieve a sharing value of $n(12m + 6) + 4m$, we need exactly one edge, which must cost 1 and along which no route is transported. Therefore, the first route that is transported by the variable route must be the first positive literal route or the first negative literal route, and the last route that is transported must be the last positive literal route or the last negative literal route. Otherwise, there would be more edges, besides the first or last edge of the variable route, along which no route is transported. Because the only edges that cost 1 are in G_{Init} and G_{End}, the edge along which no route is transported must be in G_{Init} or G_{End}. If we decide that the first route we transport is the first negative literal route corresponding to the variable, then we have already used up that edge in G_{Init}, and we cannot skip any further edges. Therefore, we must transport all respective negative literal routes to achieve a sharing value of $n(12m + 6) + 4m$ for the variable route. If we instead decide that the first route we transport is the positive literal route corresponding to the variable, then we still have one edge available along which no route is transported. However, because the first positive literal route ends outside of G_{Init}, the edge we skip must be in G_{End}, and we must continue transporting all the subsequent positive literal routes until the last positive literal route ends in G_{End}. At that point, there is only the last edge of the route left, which costs 1 and along which we cannot transport another route. Thus, to achieve a sharing value of $n(12m + 6) + 4m$, a variable route must either transport all respective negative literal routes or all respective positive literal routes. ∎

Now we know which sharing value the variable routes can achieve. As noted earlier, literal routes cannot transport other routes. Thus, we only need to determine which sharing value clause routes can achieve to know what the highest possible cumulative sharing value across all routes is. We show that clause routes can transport only one route, which must be a literal route. We show that the literal routes inside a clause cost $6m + 1$ if the literal is part of the corresponding clause and only $6m$ if the literal is not part of the corresponding clause. This effect can also be observed in Figure 3.2, where edge costs are based on an example clause $k_i = u_2 \vee \overline{u_3} \vee u_4$. The literal routes corresponding to literals that are part of the clause are marked with dashed gray arrows, while the literal routes corresponding to literals that are not part of the clause are marked with dotted arrows in a lighter shade of gray.

Lemma 3.3: *Clause routes can transport at most one route, which must be a literal route. The highest achievable sharing value for a clause route is* $6m + 1$. *They achieve that sharing value if and only if the literal corresponding to the literal route is part of the clause.*

Proof. Consider the clause route corresponding to k_i. The only routes that start and end between s_i and t_i are literal routes going from $a^{k_i,\text{in}}$ to $a^{k_i,\text{out}}$, for some literal a. Because all these literal routes end after all these other literal routes have started, at most one literal route can be transported. All these literal routes use exactly $2m$ edges. We now calculate the cost of these literal routes based on whether they are part of the clause. For that, we pair edges costing 2 with edges costing 4 to show that the average cost of an edge inside these literal routes is 3, unless the literal is part of the clause. In the following, we say that a vertex has the property *Used* if and only if the vertex is an in-vertex for a literal used in this clause. The cost of the edges is either 2, 3, or 4. If an edge goes from a vertex without *Used* to a vertex with *Used*, then its cost is 2. If the next edge also enters a vertex with *Used*, its cost is 3; if instead it goes to a vertex without *Used*, its cost is 4. Because, by construction, only in-vertices can have *Used*, we can find an edge that costs 4 for every edge that costs 2 for each literal route, as all literal routes end with an out-vertex. Similarly, we can find an edge that costs 2 for every

edge that costs 4 for each literal route, unless the literal route starts with a vertex with *Used*. In that case, the edge that costs 2 and would pair with the first edge that costs 4 would be in the path before the literal route starts. Otherwise, the cost of all edges balances out to 3 per edge. Therefore, literal routes starting with an in-vertex that has *Used*, which is the case for literals that are part of k_i, cost $2m \cdot 3 + 1 = 6m + 1$. Meanwhile, literal routes starting with an in-vertex that does not have *Used*, which is the case for literals that are not part of k_i, cost $2m \cdot 3 = 6m$. ∎

We call the total sharing value W the sum of the sharing values of all routes in an optimal solution.
The maximum sharing value $M = m(n(12m + 6) + 4m) + n(6m + 1)$ is the sum of the highest achievable sharing values, which we have just calculated, of all routes. In the following lemma, we prove that M behaves as intended.

Lemma 3.4: *$W \leq M$, and in case $W = M$, every variable route has a sharing value of $n(12m + 6) + 4m$, and every clause route has a sharing value of $6m + 1$.*

Proof. For all literal routes, there is no route that is a subpath of the literal route. Thus, they can never transport another route. Assuming all variable routes and clause routes achieve their maximum sharing value, the maximum sharing value for each of the m variable routes is $n(12m + 6) + 4m$ by Lemma 3.2 and $6m + 1$ for each of the n clause routes by Lemma 3.3. In total, this results in a maximum sharing value of $m(n(12m + 6) + 4m) + n(6m + 1)$. ∎

Now we use the previous lemmas to prove that the Vehicle Sharing Fixed Routes Problem is NP hard. To do so, we show that the SAT instance (U, K) has a solution if and only if the total sharing value W is equal to M. Therefore, we can decide whether the SAT instance has a solution if we have a solution to the instance of the Vehicle Sharing Fixed Routes Problem that we have just constructed.

Theorem 3.5: *The Vehicle Sharing Fixed Routes Problem is NP hard, even if we restrict the graph to a directed path with unit distance edges for $C = 2$.*

Proof. We now prove the equivalence: SAT instance (U, K) has a solution $\iff$ $W = M$.
“ $\implies$ ”
Assume the SAT instance has a solution. Then, there is an interpretation that satisfies the SAT instance. Because no literal route can transport another route, routes can only be transported by clause routes and variable routes. For every variable $u_i \in U$: If u_i is false, let the variable route for u_i transport all the literal routes representing the positive value u_i. If u_i is true instead, let the variable route for u_i transport all the literal routes representing the negative value $\overline{u_i}$. By Lemma 3.2, each of the variable routes achieves a sharing value of $n(12m+6)+4m$ by transporting all respective negative or all respective positive literal routes, so combined, they achieve a sharing value of $m(n(12m+6)+4m)$. Since this interpretation is a solution, for each of the clauses k_i there is at least one literal a_i that is fulfilled. Thus, the literal routes for this literal a_i are not being transported by any of the variable routes yet, and the literal route from $a_i^{k_i,\text{in}}$ to $a_i^{k_i,\text{out}}$ can be transported by the clause route for k_i. By Lemma 3.3, each of the clause routes achieves a sharing value of $6m + 1$ by transporting a literal route corresponding to a literal used in the clause, so combined, they achieve a sharing value of $n(6m + 1)$. As we avoided transporting the same literal route multiple times, this is a valid assignment. If we combine the sharing values of the variable routes and the clause routes, we get a value of $m(n(12m + 6) + 4m) + n(6m + 1) = M$. By Lemma 3.4, W cannot be higher than M. Thus,

there is no better solution and $W = M$.
" $\Longleftarrow$ "
Assume $W = M$. By Lemma 3.4, every variable route has a sharing value of $n(12m + 6) + 4m$, and every clause route has a sharing value of $6m + 1$. By Lemma 3.2, every variable route transports either all its respective negative literal routes or all its respective positive literal routes. Because, for every variable, either all positive or all negative literal routes are already occupied by the respective variable route, these routes can no longer be transported by any clause route, and clause routes can only transport literal routes representing literals with the opposite value. Because by Lemma 3.3 every clause route transports a literal route that is part of the clause, the interpretation that assigns every variable the value that is not being transported by its variable route is a solution to the SAT instance.

Thus, we can determine whether the SAT instance (U, K) has a solution based on the total sharing value W obtained from an optimal solution to the associated Vehicle Sharing Fixed Routes Problem instance. ∎

Now we show how to extend the previous construction to an arbitrary vehicle size $C > 2$. First of all, we add $\ell = C - 2$ additional variables to the variable set, so $U' = U \bigcup \{u_{m+1}, ..., u_{m+\ell}\} = \{u_1, ..., u_{m'}\}$. The idea is that we add the positive literal for ℓ new variables to each clause so that the additional capacity in the clause route's vehicle can be occupied by the positive literals of the newly added variables while achieving a higher sharing value than if they transported literals not part of the clause. That way, we avoid the problem that satisfying one clause three times and another clause zero times is better than satisfying both clauses only once. Therefore, each clause $k_i \in K$ is transformed into $k'_i = k_i \vee u_{m+1} \vee ... \vee u_{m+\ell}$, so $K' = \{k'_1, ..., k'_n\}$.
We also add ℓ additional artificial literals for every variable. These artificial literals are not literals in the traditional sense of SAT but behave like them in the graph construction. The idea is that every variable route can fill the additional capacity in their vehicle using the new artificial literal routes corresponding to their variable, so that it is not beneficial for them to suddenly start transporting literal routes corresponding to different variables or even to start transporting clause routes. These artificial literals are not part of any clause, so clauses benefit less from transporting an artificial literal than from transporting a literal used in the clause. For each variable, the artificial literals are placed between their positive and their negative value: Thus, the order of the vertices in G' is now $u_1, u_{1:1}, .., u_{1:\ell}, \overline{u_1}, u_2, u_{2:1}, .., u_{2:\ell}, \overline{u_2}, ..., u_{m'}, u_{m':1}, .., u_{m':\ell}, \overline{u_{m'}}$, where $u_{i:j}$ refers to the j-th artificial literal for the variable u_i.
We extend $G_{\text{Init}}, G_{k'_1}, ..., G_{k'_n}, G_{\text{End}}$ accordingly to $G'_{\text{Init}}, G'_{k'_1}, ..., G'_{k'_n}, G'_{\text{End}}$ with the new variables and artificial literals. New literal routes are introduced for the artificial literals in the same way that the other literal routes operate.
We also must modify the edge costs, so a sequence of $C - 1$ edges with a cost of 1 in G'_{Init} and G'_{End} is cheaper than any other edge in that graph. The newly added edges in G'_{Init} and G'_{End} between literals corresponding to the same variable all cost 1. For the other edges, we use the same formula that we used previously. However, edges that cost 2 now cost C, edges that cost 3 now cost $C + 1$, and edges that cost 4 now cost $C + 2$.
Figure 3.5 contains an example of how additional artificial literals are placed between the positive and negative literals in G_{Init}. In that example, $C = 4$, so two artificial literals are added per variable.
Figure 3.6 contains an example of how a clause is extended by newly introduced variables. Here $k_i = u_2$ is transformed into $k'_i = u_2 \vee u_{m+1}$. Because $C = 3$, the positive literal for $C - 2 = 1$

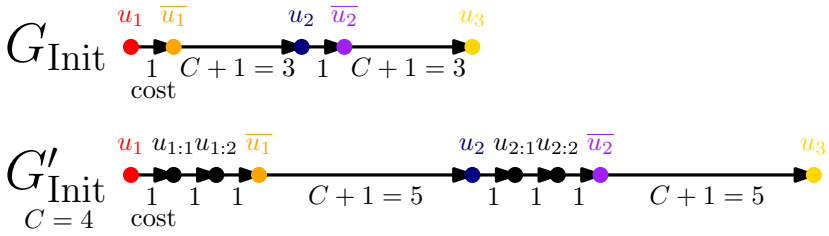

Figure 3.5: Example how to adjust G_{Init} to G'_{Init} for $C = 4$

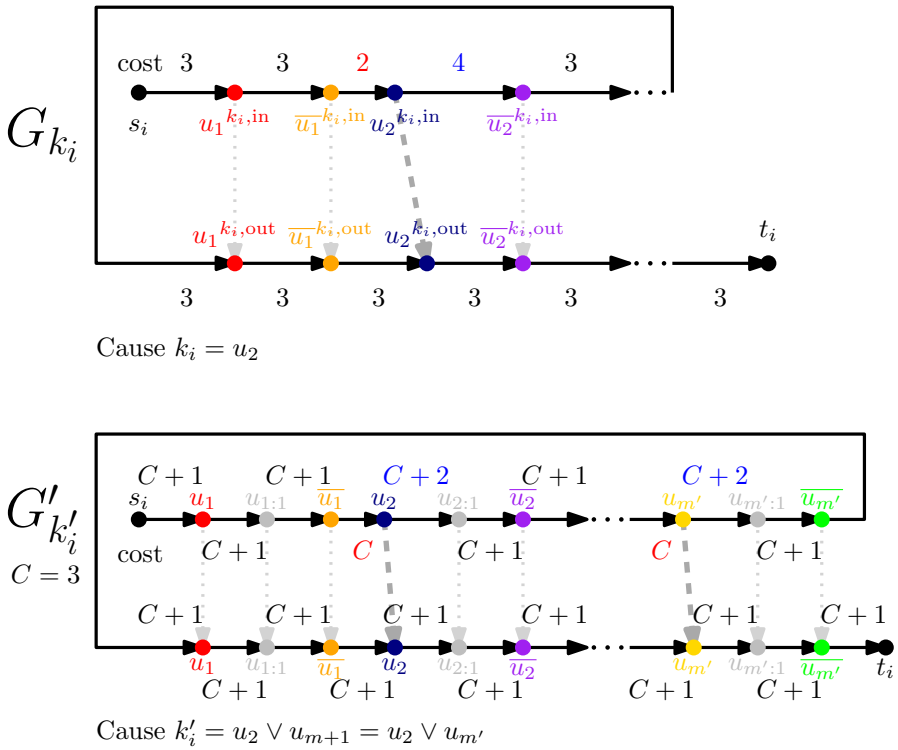

Figure 3.6: Example how to adjust G_{k_i} to $G'_{k'_i}$ for $C = 3$ and $k_i = u_2$

new variables has been added to each clause.

Next, a sketch is provided on how the proof for Theorem 3.5 must be modified to fit the revised instance of the Vehicle Sharing Fixed Routes Problem.

Theorem 3.6: *The Vehicle Sharing Fixed Routes Problem is NP hard, even if we restrict the graph to a directed path with unit distance edges for $C \geq 2$.*

Proof Sketch. In Lemma 3.1, variable routes now use $n(2Cm' + 2) - 1$ edges inside and connecting the clause subpaths. The average cost of these edges is $C + 1$, which results in a cost of $n(C + 1)(2Cm' + 2) - (C + 1)$. The edges connecting G'_{Init} to $G'_{k'_1}$ and $G'_{k'_n}$ to G'_{End} cost $C + 1$

each. The i-th variable route uses $m' - i$ edges with cost $C + 1$, as well as $(C - 1)(m' - i + 1)$ edges with cost 1 in G'_{Init}, and $i - 1$ edges with cost $C + 1$, as well as $(C - 1)i$ edges with cost 1 in G'_{End}. In total, the cost per variable route is:

$$\begin{aligned}
&n(C+1)(2Cm'+2) - (C+1) + 2 \cdot (C+1) \\
&+ (C+1)(m'-i) + (C-1)(m'-i+1) + (C+1)(i-1) + (C-1)i \\
&= n(C+1)(2Cm'+2) + (C+1)(m'-i+i-1+2-1) + (C-1)(m'-i+1+i) \\
&= n(C+1)(2Cm'+2) + (C+1)m' + (C-1)m' + C - 1 \\
&= n(C+1)(2Cm'+2) + 2Cm' + C - 1.
\end{aligned}$$

In Lemma 3.2, the highest achievable sharing value is $(C-1)(n(C+1)(2Cm'+2)+2Cm'+C-1)-(C-1)(C-1) = (C-1)(n(C+1)(2Cm'+2)+2Cm')$ instead. It is achieved by transporting all literal routes corresponding to $C-1$ respective literals, which again leaves only one literal per variable for the clause routes. Per transported literal, there are $C - 1$ edges costing 1 in G'_{Init} and G'_{End} along which the literal routes corresponding to that literal do not use the slot. Note that it is important here that we previously modified the edge costs, so every edge outside of G'_{Init} and G'_{End} has a cost of at least C.
In Lemma 3.3, clause routes can instead transport at most $C - 1$ literal routes. Per transported literal route, they achieve a sharing value of $(C + 1)Cm' + 1$ if and only if the literal is part of the clause and only $(C + 1)Cm'$ otherwise. Thus, the highest achievable sharing value is $(C - 1)((C + 1)Cm' + 1)$, which is achieved if and only if the clause route transports $C - 1$ literal routes for literals that are part of the clause.
The maximum sharing value M' is instead $m'((C-1)(n(C+1)(2Cm'+2)+2Cm'))+n((C-1)((C+1)Cm'+1))$, where M' is the upper bound for W. In the case of $W = M'$, every variable route has a sharing value of $(C-1)(n(C+1)(2Cm'+2)+2Cm')$, while every clause route has a sharing value of $(C-1)((C+1)Cm'+1)$.
In Theorem 3.5, given that there is an interpretation that satisfies the SAT instance (U, K), for every variable $u_i \in U$: If u_i is false, let the variable route for u_i transport all the literal routes representing the positive value u_i, as well as the artificial literal routes representing $u_{i:1}, ..., u_{i:\ell}$. If u_i is true instead, let the variable route for u_i transport all the literal routes representing the negative value $\overline{u_i}$, as well as the artificial literal routes representing $u_{i:1}, ..., u_{i:\ell}$. For the additional variables we added to U to obtain U', we treat them as if they were true, so that the clause routes can transport the positive literal route inside their clauses, as the positive literals corresponding to these variables are part of each modified clause. Therefore, the clause routes can transport a literal route corresponding to each of the positive literals for the newly added variables, as well as one literal route corresponding to a literal that fulfills the clause in this interpretation. It exists because this interpretation is a solution.
If we assume that $W = M'$, then we find an interpretation that fulfills the SAT instance by assigning every variable the value that is not being transported by its variable route. If a variable route decides to transport both the positive and negative literals, then there is one artificial literal it does not transport, and we can choose an arbitrary value for this variable, as both literals are blocked from all clause routes anyway. ❑

4 Algorithm for One Driver Scenario

We have just observed that the problem is NP hard, even if we constrain the underlying graph to be a directed path with unit distance edges. Therefore, instead of only placing constraints on the graph, it might be interesting to impose constraints on which routes are allowed to transport other routes as well.
We consider the scenario in which there is only one route, which we call the driver, that can transport other passengers. This route may not include repetitions of the same vertex, and we do not allow edges with a cost less or equal to 0. Therefore, the route must be a path. If the driver's vehicle has the capacity C, then the driver can transport up to $k = C - 1$ passengers at a time. We assume that all n potential passengers p_i want to be transported along a specific subsection of the driver's route, starting at s_i and ending at t_i. For vertices along the driver's route, we measure the distance from the starting point of the driver's route, so the cost of the i-th passenger's route is $c(p_i) = t_i - s_i$. We call this scenario the One Driver Path Sharing Problem.

We first consider the case where $C = 2$, so the driver can only transport one passenger at a time. This problem can be solved in $O(n \log n)$ using Algorithm 4.1.

Algorithm 4.1: Algorithm that solves the One Driver Path Sharing Problem for $C = 2$

Input: n routes with a starting point s_i, endpoint t_i and cost $c(i) = t_i - s_i, i \in [n]$
Output: Set of routes with maximal cost that can be transported by the driver

```
1  Sort all starting points and endpoints by increasing distance from the beginning of the
     driver's route. If an endpoint and a starting point have the same distance, place the
     endpoint before the starting point
2  last ⟵ ⊥
3  w ⟵ 0
4  p ⟵ Array of length n, stores the index of the predecessor
5  v ⟵ Array of length n, stores the combined value of this route and its predecessors
6  for each point ∈ input do
7      if point = s_i then
8          p[i] ⟵ last
9          v[i] ⟵ w + c(i)
10     else if point = t_i then
11         if v[i] > w then
12             last ⟵ i
13             w ⟵ v[i]
14 r ⟵ {}
15 while last ≠ ⊥ do
16     r.insert(last)
17     last ⟵ p[last]
18 return r
```

The idea is that because routes can be transported by only one driver, the only way to decrease vehicle operation time is for other routes to be transported by that driver. The vehicle operation time is minimized if the combined cost of the routes transported by the driver is as large as possible. Due to the limited vehicle size of $C = 2$, we can only transport routes that do not overlap. The algorithm effectively travels along the path of the driver's route. It stores the index of the last route transported in the currently best-known solution in *last* and the associated value of that solution in w. Whenever the algorithm traverses a starting point s_i, it stores *last* as the predecessor of the i-th route in $p[i]$ and the combined value of transporting the i-th route as well as its predecessors $w + c(i)$ in $v[i]$. Whenever the algorithm traverses an endpoint t_i, it checks whether transporting the i-th route is better than the currently best-known solution. In case transporting the i-th route is better, then *last* and w are updated accordingly. The correctness relies on a starting point s_i only being traversed after all endpoints with equal or less distance have been traversed, so the best-known solution is optimal for the section $[0, s_i]$ whenever s_i is traversed.

Theorem 4.1: *Algorithm 4.1 correctly solves the One Driver Path Sharing Problem for $C = 2$. It runs in $O(n \log n)$.*

Proof. The reconstructed solution consists of the route that is the last route in the best-known solution, as well as the predecessors that are found by iteratively following $p[\text{last}]$. Therefore, we must show that the algorithm processes the points in the correct order, so that whenever a starting point s_i is processed, the best-known solution is optimal for the section $[0, s_i]$. We must also show that *last* correctly contains the index of the last route in the currently best-known solution, and that w stores the cost of the currently best-known solution.
Due to the sorting of the points, whenever a starting point s_i is processed, all endpoints with the same or less distance have already been processed, and all endpoints with higher distance have not been processed yet.

We now show by induction that the invariant, that *last* at any point contains the index of the last route in the currently best-known solution, and that w stores the associated cost of that solution, is maintained whenever a point is processed. We also show that once a starting point s_i has been processed, $p[i]$ is correctly initialized to the predecessor index in an optimal solution containing the i-th route and $v[i]$ to the associated combined cost.
Base case. Initially we have not found the endpoint of any route yet, so there is no route we can transport on the traversed section, which means that *last* correctly points to no route with an associated cost $w = 0$. We have not traversed any starting points yet, so no array entries must be initialized.
Inductive step. Whenever we find an endpoint t_i, we check whether transporting the i-th route and its predecessors yields a better solution than the cost of the best-known solution stored in w. Only if that solution yields a better solution for the section $[0, t_i]$, we update *last* and w accordingly. By induction we correctly initialized the entries $p[i]$ in the predecessor array and $v[i]$ in the combined value array, when we previously traversed s_i, so we correctly update the best-known solution for the section $[0, t_i]$ when traversing t_i. Note that we can find a better solution for the section $[0, t_i] = [0, t_j]$ right afterwards if the i-th route and the j-th route end at the same vertex.
Whenever we find a starting point s_i, by induction, *last* contains the index of the last route in the best solution for the section $[0, s_i]$, and w contains the associated cost. Therefore, the best solution that includes the i-th route uses *last* as its predecessor and the combined cost of that route is $w + c(i)$, which we correctly store in $p[i]$ and $v[i]$.

Thus, we have shown that whenever a starting point s_i is processed, we currently store an optimal solution for the section $[0, s_i]$, which means that the solution we store in $p[i]$ and $v[i]$ is optimal for the section $[0, t_i]$ if we are forced to include the i-th route.
When reconstructing the solution, we first find the last route p_j transported in an optimal solution. Because we cannot transport multiple routes at the same time, the combination of the optimal solution for the section $[0, s_j]$ and the j-th route, is an optimal solution for the section $[0, t_j]$. Therefore, we correctly reconstruct the solution by iteratively following $p[\text{last}]$ and the algorithm is correct.
Sorting the starting points and endpoints is possible in $O((2n)\log(2n)) = O(n \log n)$. We then traverse $2n$ points, performing a constant number of $O(1)$ operations for each, so traversing all points takes $O(n)$ time. The starting point of a predecessor of a route always appears before the starting point of that route, so we find at most $2n$ predecessors that we insert into the result, which means that this step once again requires $O(n)$ time. Thus, the runtime is dominated by the sorting step, resulting in an overall complexity of $O(n \log n)$.

■

Now that we know how to solve the One Driver Path Sharing Problem for $C = 2$, it might be interesting to know how to solve it for an arbitrary vehicle size $C \geq 2$. Solving it for an arbitrary vehicle size $C \geq 2$ is closely related to solving the Maximal Weight Fixed Interval Scheduling problem. Maximal Weight Fixed Interval Scheduling tries to schedule n independent and non-preemptive jobs for processing in a parallel machine environment with k identical machines. Each job has a fixed start time, a fixed end time, and a weight. Each machine can process at most one job at a time, and a job can be executed by at most one machine at a time. The goal is to find a subset of jobs with maximal total weight that can feasibly be scheduled [AS87 | CL95 | BE96 | KNC07].
The One Driver Path Sharing Problem is identical to Maximal Weight Fixed Interval Scheduling if we turn every passenger p_i into a job starting at s_i, ending at t_i, with a weight of $c(p_i) = t_i - s_i$ and use $k = C - 1$ processors. Fortunately, there exist algorithms that solve Maximal Weight Fixed Interval Scheduling in $O(kn \log n)$ [CL95 | BE96]. The algorithms are based on finding the Minimum Cost Flow of Size k. In the Minimum Cost Flow of Size k problem, we are given a directed graph $G = (V, E)$, where each edge has an associated cost, as well as a nonnegative capacity. We are also given a source vertex $s \in V$ and a sink vertex $t \in V$, as well as an integer k. A flow from s to t is a function $f : E \to \mathbb{R} \geq \mathbb{0}$ satisfying the capacity and flow conservation constraints:
(1) Capacity constraint: For every edge $e \in E$: $f(e) \leq \text{cap}(e)$.
(2) Flow conservation: For every vertex besides s and t the outgoing flow is equal to the incoming flow.
The value of the flow is equal to the flow outgoing from s and incoming in t. This problem asks to find a flow of size k such that the cost of the flow $\sum_{e \in E} \text{cost}(e) f(e)$ is minimized [Bin+18 | Tar83].

Algorithm 4.2: Algorithm for Maximal Weight Fixed Interval Scheduling (Bouzina and Emmons) [BE96]; renamed jobs to routes to fit the One Driver Path Sharing Problem

Input: An integer k
Input: n routes with a starting point s_i, endpoint t_i and cost $c(i)$, $i \in [n]$
Output: Set of routes with maximal cost that can be transported by the driver using a vehicle with capacity $C = k + 1$

```
1  Index routes in chronological order of starting points
2  Construct a directed graph G = (V, E_1 ⋃ E_2), where edges have a cost and a capacity,
     with V = {v_1, ..., v_{n+1}}
3  Add edges (v_j, v_{j+1}) with cost 0 and capacity k for j ∈ [n] to E_1
4  Add edge (v_j, v_k) where k is the first route not overlapping with route j for j ∈ [n]. If
     no such route k exists, instead add the edge (v_j, v_{n+1}). These edges have cost −c(j)
     and capacity 1 and are added to E_2
5  Find the Minimum Cost Flow of Size k on G with s = v_1 and t = v_{n+1}
6  r ⟵ {}
7  for each (v_j, x) ∈ E_2 do
8      if flow((v_j, x)) = 1 then
9          r.insert(j)
10 return r
```

Algorithm 4.2 [BE96] constructs a Directed Acyclic Graph (DAG) with two groups of edges. The edges in E_1 are skip edges that can be used to not fill up a vehicle slot for some time. The edges (v_j, v_k) in E_2 represent the j-th route being transported, which yields the reward $c(j)$. Finally, using a solution for the Minimum Cost Flow of Size k on that graph, all routes whose representing edges have a flow of 1 are added to the result set.

Theorem 4.2: *Algorithm 4.2 correctly solves the One Driver Path Sharing Problem for $C \geq 2$. It runs in $O(Cn \log n)$.*

Proof. For correctness refer to Bouzina and Emmons [BE96]. Finding the Minimum Cost Flow of Size k dominates the runtime, which can be solved in $O(km \log n)$ on DAGs with integer capacities [Tar83]. Here, m is in $O(n)$, which means that the total runtime is in $O(kn \log n)$. With $k = C - 1$, we obtain a runtime of $O(Cn \log n)$. ∎

If we want to place some additional time constraints on the routes in the One Driver Path Sharing Problem, such that each route p_i requires the driver to start driving within a certain time window $[a_i, b_i]$, then we can solve the problem by running the respective algorithm with the eligible routes once for each time window. There can be up to $2n$ time windows, which means we must run the algorithm $O(n)$ times. Afterwards, we pick the time window where the algorithm's result provides the best solution. For $C = 2$, we can improve the performance of running Algorithm 4.1 $O(n)$ times to $O(n^2)$ by sorting all points only once. For arbitrary $C > 2$, running Algorithm 4.2 $O(n)$ times yields a performance of $O(Cn^2 \log n)$.

5 Conclusion

We have shown that solving the Vehicle Sharing Fixed Routes Problem is NP hard, even if the underlying road network has a very simple structure and consists only of a directed path with unit distance edges. Then, we have considered the scenario in which there is only one driver who can serve other passengers and obtained an algorithm that solves this scenario in $O(nC \log n)$ by reducing it to the Maximal Weight Fixed Interval Scheduling problem. As this version of the problem is very hard to solve optimally, even though we did not account for any temporal constraints passengers might have for their journey, it might be interesting to consider whether such temporal constraints make the otherwise trivial version of the problem, where transferring between vehicles is allowed for free, difficult. Another slightly different version of this problem that might be interesting to consider is one in which no passenger who effectively owns the vehicle must remain in the vehicle during the entire journey of the vehicle.

Bibliography

[AS87] Esther M. Arkin and Ellen B. Silverberg. “Scheduling jobs with fixed start and end times”. en. In: *Discrete Applied Mathematics* Volume 18 (Sept. 1987), pp. 1–8. ISSN: 0166218X. DOI: *10.1016/0166-218X(87)90037-0*.

[BE96] Khalid I. Bouzina and Hamilton Emmons. “Interval Scheduling on identical machines”. en. In: *Journal of Global Optimization* Volume 9 (Dec. 1996), pp. 379–393. ISSN: 0925-5001, 1573-2916. DOI: *10.1007/BF00121680*.

[Bin+18] Timo Bingmann, Johannes Fischer, Robert Geisberger, Moritz Kobitzsch, Peter Sanders, Dennis Schieferdecker, Christian Schulz, and Johannes Singler. “Algorithmen II (WS 18/19)”. en. In: (Oct. 2018).

[Blä+25] Thomas Bläsius, Adrian Feilhauer, Markus Jung, Moritz Laupichler, Peter Sanders, and Michael Zündorf. *Synergistic Traffic Assignment*. en. Feb. 2025. DOI: *10.48550/arXiv.2502.04343*.

[CL95] Martin C. Carlisle and Errol L. Lloyd. “On the k-coloring of intervals”. In: *Discrete Applied Mathematics* Volume 59 (May 1995), pp. 225–235. ISSN: 0166-218X. DOI: *10.1016/0166-218X(95)80003-M*.

[DB22] Stacy C Davis and Robert G Boundy. *Transportation Energy Data Book: Edition 40*. en. Oak Ridge, TN: Oak Ridge National Laboratory, 2022.

[KNC07] Mikhail Y. Kovalyov, C. T. Ng, and T. C. Edwin Cheng. “Fixed interval scheduling: Models, applications, computational complexity and algorithms”. In: *European Journal of Operational Research* Volume 178 (Apr. 2007), pp. 331–342. ISSN: 0377-2217. DOI: *10.1016/j.ejor.2006.01.049*.

[SEL19] David Schrank, Bill Eisele, and Tim Lomax. *Urban Mobility Report 2019*. College Station, TX: Texas Transportation Institute, Aug. 2019.

[Tar83] Robert Endre Tarjan. *Data structures and network algorithms*. en. 10. print. Philadelphia, Pa: Soc. for Industrial and Applied Mathematics, 1983. ISBN: 978-0-89871-187-5.