information

Article

Snapshot-Optimal Real-Time Ride Sharing

Afzaal Hassan [1], Mark Wallace [2], Irene Moser [1,*] and Daniel D. Harabor [2]

1 School of Science, Computing and Engineering Technologies, Swinburne University of Technology, Melbourne, VIC 3122, Australia; afzaalhassan@swin.edu.au
2 Faculty of Information Technology, Monash University, Melbourne, VIC 3800, Australia; mark.wallace@monash.edu (M.W.); daniel.harabor@monash.edu (D.D.H.)
* Correspondence: imoser@swin.edu.au

Abstract: Ridesharing effectively tackles urban mobility challenges by providing a service comparable to private vehicles while minimising resource usage. Our research primarily concentrates on dynamic ridesharing, which conventionally involves connecting drivers with passengers in need of transportation. The process of one-to-one matching presents a complex challenge, particularly when addressing it on a large scale, as the substantial number of potential matches make the attainment of a global optimum a challenging endeavour. This paper aims to address the absence of an optimal approach for dynamic ridesharing by refraining from the conventional heuristic-based methods commonly used to achieve timely solutions in large-scale ride-matching. Instead, we propose a novel approach that provides snapshot-optimal solutions for various forms of one-to-one matching while ensuring they are generated within an acceptable timeframe for service providers. Additionally, we introduce and solve a new variant in which the system itself provides the vehicles. The efficacy of our methodology is substantiated through experiments carried out with real-world data extracted from the openly available New York City taxicab dataset.

Keywords: ridesharing; ridematching; shared mobility

check for updates

Citation: Hassan, A.; Wallace, M.; Moser, I.; Harabor, D.D. Snapshot-Optimal Real-Time Ride Sharing. *Information* **2024**, *15*, 174. https://doi.org/10.3390/info15040174

Academic Editor: Katsuhide Fujita

Received: 10 February 2024
Revised: 17 March 2024
Accepted: 18 March 2024
Published: 22 March 2024

1. Introduction

Traffic congestion is a global problem. While travel delays are the most visible consequence of congestion, the economic toll it exacts reaches into the billions of dollars [1]. To mitigate congestion costs, promoting shared private vehicles and mass public transport is beneficial. These options are faster, more cost-effective, and have a lower carbon footprint. However, limitations include less flexibility, constrained routes, variable service frequencies, and limited availability. This research aims to develop better algorithmic techniques for intelligent trip sharing. In this context, we introduce a novel algorithmic approach, SnapPair, that optimises participant pairing in a ridesharing system. This adaptable approach can be applied in various settings and under different policies. We implement SnapPair in traditional role-based one-to-one matching scenarios, encompassing designated drivers and riders, as well as in scenarios with flexible roles. Furthermore, we propose a new type of ride matching scheme that involves a fleet of autonomous vehicles (AVs) or human-driven cars providing on-demand door-to-door transportation. Passengers can request trips on the spot; centralized algorithms then schedule vehicles and combine trips to maximise efficiency. The system aims to reduce the vehicle hours traveled (VHT), thereby mitigating congestion and associated costs. In this latter formulation, we assume the existence of a fleet operator responsible for providing the vehicles to serve all trips generated by SnapPair. These fleets can be operated by public or private entities, similar to UberPool or Lyft Shared Rides. The proposed formulation envisions a future where car ownership becomes obsolete and shared trips become the default mode of transportation for optimal vehicle utilisation.

We perform an experimental evaluation of SnapPair by applying it in an online fashion to imitate the behaviour of commuters who book their journeys ad hoc in everyday life.

New demand is included in the subsequent time window, and time windows are optimised at regular intervals.

In addition, we devise and implement several matching policies that differ in their level of eagerness to commit to a match. Our experiments evaluate their performance using both classic role-based matching schemes and our new formulation.

Considering the dynamic nature of link costs, which can experience rapid fluctuations throughout the day, we have taken an additional step by investigating an unconventional scenario in which the matching algorithm itself performs the shortest path calculations.

2. Definitions

- Rider: A participant in the ride sharing system who wishes to be transported from their origin to their destination within an announced time frame.
- Match: A pair of riders who share a vehicle on their trip.
- Snapshot Optimality: Attaining an optimal solution, defined as the maximum reduction in vehicle hours traveled (VHT), for a pairwise matching problem with a fixed demand.

3. Previous Work

Peer-to-peer ride matching offers shared journeys to the participants while maximising some global objective (usually savings in travel time) subject to the participants' spatiotemporal constraints [2]. There are multiple variants of this problem; some classifications are based on cardinality of matching (one-to-one, one-to-many, many-to-many), while others are based on the roles the participants play (riders, drivers). These roles can be either fixed or flexible [3].

One-to-one ride matching can be represented as a graph matching problem [3] in which each participant is a node, potential matches are edges, and the edge weight indicates the savings from the match. When roles are fixed, it becomes a maximum weighted bipartite matching problem, while if the roles are flexible it can be formulated as a maximum weight matching problem in general graphs. In the past, many polynomial-time algorithms have been proposed to solve these problems optimally [4–7]. Recent studies with exact solutions [8,9] have attempted the matching problem at a very small scale with up to few hundred participants and a small number of locations.

Unlike in the above studies, real-world ride sharing problems are large-scale and dynamic in nature [2] and require solutions in close to real time. In their landmark survey of a decade ago, Agatz et al. [10] noted the lack of fast optimal solutions for large metropolitan areas; since then, a substantial body of work has focused on developing fast solutions for real-time large-scale ride sharing systems, especially for the one-to-one version. Catering to the assumption of continuous travel requests, most studies are dynamic and use a rolling time window approach [11]. To reduce complexity, previous studies have tended to publish methods dividing the problem into smaller subproblems. Shen et al. [12] partitioned the road network into grids, with every participant only being matched within their grid. Xu et al. [13] made use of ellipses to bound possible locations for matches in order to avoid removing the optimal solution during pruning; however, this approach requires equal speed limits. A similar approach was taken by Masoud and Jayakrishnan [8]. A number of approaches have used graph partitioning techniques [11,14]. The ride sharing problem has been formulated as a linear program; at times this has been applied to a reduced search space [11], while in other cases authors have used a time-out period to provide answers within the specified time [15].

In other examples of heuristic approaches, Najmi et al. [16] used a clustering approach to investigate potential matches for riders in two sets of clusters: one based on the riders' origins, and the other on their destinations. Ketabi et al. [17] also employed a clustering method to find matches for one-to-one matching. In their study, Ta et al. [18] exclusively examined matches where the intersection of the driver and riders' journeys surpassed a predetermined threshold, typically set at 80%. However, they did not provide a rationale for the selection of this specific percentage. They chose matches using an approximate method focused on maximising the shared route percentage. Li et al. [19] approached the ridesharing problem by framing it as a vehicle routing problem, exploring both one-to-one and many-to-one variations. Stressing the NP-hard nature of the problem, they opted for a hybrid heuristic algorithm combining an insertion algorithm with tabu search for quick solution generation. Additionally, they incorporated clustering to pair participants within their respective clusters. Kleiner et al. [20] proposed an auction-based mechanism for matching drivers and riders. Their approach prioritised either maximising cost savings through reduced travel distance or achieving a high matching rate, but not both simultaneously. Nourinejad and Roorda [21] presented an auction-based model in which driver agents bid on riders and only willing riders accept the bids. Aissat and Oulamara [22] solved the fixed roles matching problem heuristically, and introduced the concept of intermediate meeting points to minimise detours for drivers.

Thus far, no published approach has solved dynamic one-to-one matching for large-scale demand in time for an operating service. Tafreshian and Masoud [11] have come the closest; however, their solution takes an impractically long time to compute an optimal match. For example, when working with another instance of similar size from the same dataset that we have utilised here, it takes roughly six minutes to solve for one-to-one matching with flexible roles for a demand contained in a one-minute time window. To address this shortcoming, they resorted to using a graph-partitioning heuristic to achieve real-time solutions. Another issue with their approach is that, to reduce the computational complexity, they introduce the concept of stations instead of providing door-to-door service, thereby reducing the number of graph nodes to be considered. Thus, passengers are expected to walk from their origin to the nearest station and board/disembark vehicles there. Similar approaches include Fiedler et al. [23], Fielbaum [24], and Wang et al. [25]. While computationally advantageous, station-based approaches present limitations in real-world scenarios. Notably, they might disadvantage individuals with mobility limitations, elderly individuals, and those who are unable or unwilling to walk to designated stations. Therefore, these methods represent a trade-off between computational efficiency and inclusivity.

Lu et al. [26] is the most recent work in pursuit of optimality that we have come across. They have proposed an exact methodology for addressing one-to-one matching with flexible roles; however, it is crucial to highlight that their approach achieves optimality only for very small instances. For problem instances involving ten participants their algorithm required only a few minutes for completion, whereas experiments with 35 participants required hours to solve. Consequently, it is evident that an optimal approach to one-to-one matching that can provide solutions within practical time frames is absent from the literature.

Furuhata et al. [27] have postulated that an optimal approach to solving one-to-one matching at large scale in an authentic setting would be a major breakthrough. In this paper, we propose and empirically test such an approach.

Contributions

In this paper, we publish the following contributions:

- A fully implemented and tested new algorithm, SnapPair, that guarantees a snapshot-optimal solution at each time point in a dynamic one-to-one matching problem. SnapPair is more than two orders of magnitude faster than the current state-of-the-art.
- A novel formulation of the dynamic one-to-one matching problem in which riders and vehicles are independent.

  This new formulation is more complex due to the increased number of possible matches. We call this formulation FreeMatch. We consider this formulation timely and significant, particularly in light of research indicating that a higher number of individuals have reported issues with crowded vehicles since the COVID-19 pandemic compared to the prepandemic period [28–30]. With FreeMatch, our objective is to establish a framework that optimally utilises standard commonplace vehicles with a maximum capacity capped at two, enhancing the appeal of ridesharing in accordance with travelers' altered preferences following the COVID-19 pandemic.

- An extended algorithm, including a rematching procedure that pairs both new riders and riders whose previous match has been dropped off, with the objective of optimising the use of vehicles available in the system.
- Experiments successfully applying the proposed algorithms to a problem size that is relevant to fleet operators.

4. Methodology

4.1. Overview

One-to-one matching is a well established problem which involves participants bringing their own vehicles, and can be viewed in two different versions. In the first version, certain individuals are designated as drivers and have vehicles, while others are riders in need of transportation. The second version assumes that all participants have vehicles, with the matching system determining which individuals assume the roles of driver and rider.

In FreeMatch, every participant is a rider looking to be transported. Riders are paired whenever possible. We assume the presence of a fleet operator who, based on their internal business logic, assigns vehicles (autonomous or human-driven) to serve paired and solo journeys facilitated by FreeMatch. Notably, FreeMatch finds particular relevance for ridesourcing companies seeking optimal passenger pairings for pooled rides, exemplified by services such as UberXShare [31]. This model, aiming to reduce customer fares through ridesharing, accommodates up to two passengers per vehicle. SnapPair can be envisioned as a tool that provides passenger pairings to a ridesourcing company while remaining agnostic to the vehicle assignment process. This approach aligns with real-world scenarios, as ridesourcing drivers have the option of accepting or rejecting trips. Consequently, it is more efficient to prioritise the optimal combination of pairs and solo journeys while leaving vehicle assignment to the fleet operator rather than potentially introducing inefficiencies by assigning them to individual drivers/vehicles who might reject the assignment.

To identify potential matches for a rider, the system generates reachability graphs that associate road network nodes with those riders who can pass through the node without reaching their destination late along with the time interval that they can spend at each node. The riders associated with the start node of rider r are the potential matches for r. This helps with the creation of a match graph, which maps the riders who can reach each other's origins and destinations without violating timelines. The match graph is then translated to a list of riders and their potential partners. These are translated into a linear program and solved to optimality by an optimisation engine that identifies the optimal matches.

4.2. Parameters and Variables

The road network is represented as a graph $G = (V, E)$, where $E \subseteq V \times V$ and where each edge $(i, j) \in E$ has an associated and positive edge cost ec_{ij} indicating the travel time. Based on these times, the system uses Dijkstra's algorithm [32] to compute the shortest paths between all pairs of nodes u, v in the graph, then records the time $w(u, v)$. In our experiments, we show that this can either be done once in advance or repeatedly for all relevant pairs at the beginning of every iteration, with the choice based on the current congestion-dependent edge costs.

Here, R is the set of riders; each rider $r \in R$ has an origin o_r, destination d_r, earliest possible departure time t_r^{ed}, and latest possible arrival time t_r^{la}.

For each rider r, the system computes the earliest possible arrival time $t_r^{ea} = t_r^{ed} + w(o_r, d_r)$.

A match $< j, k >$, where j and k are riders, is feasible if j can pick up k and if both riders depart after their earliest departure time and reach their destinations before their latest arrival time. If j is on the way to picking up k, then either j or k may have to wait at o_k until the other party is ready. Waiting times are permitted as long as both riders arrive at their destinations in time. For the sake of simplicity, pickup and dropoff are instantaneous and do not incur any delay. Formally, the route $route1(j,k) = < o_j, o_k, d_k, d_j >$ is feasible for riders j and k if

$$max(t_j^{ed} + w(o_j, o_k), t_k^{ed}) + w(o_k, d_k) \le t_k^{la}$$

and

$$max(t_j^{ed} + w(o_j, o_k), t_k^{ed}) + w(o_k, d_k) + w(d_k, d_j) \le t_j^{la}.$$

The cost of this match is

$$cost1(j,k) = w(o_j, o_k) + w(o_k, d_k) + w(d_k, d_j)$$

if $route1(j,k)$ is feasible and infinity otherwise.

The route $route2(j,k) = < o_j, o_k, d_j, d_k >$ is feasible for riders j and k if

$$max(t_j^{ed} + w(o_j, o_k), t_k^{ed}) + w(o_k, d_j) \le t_j^{la}$$

and

$$max(t_j^{ed} + w(o_j, o_k), t_k^{ed}) + w(o_k, d_j) + w(d_j, d_k) \le t_k^{la}.$$

Its cost is $cost2(j,k) = w(o_j, o_k) + w(o_k, d_j) + w(d_j, d_k)$ if $route2(j,k)$ is feasible and infinity otherwise.

Based on this, $< j,k >$ is a feasible match if $route1(j,k)$ or $route2(j,k)$ is feasible, and its cost is

$$cost(j,k) = min(cost1(j,k), cost2(j,k)).$$

For an unmatched rider, the cost is

$$cost(r) = w(o_r, d_r).$$

We write F for the set of all feasible matches.

To minimise VHT (vehicle hours travelled), the model can be formulated in terms of the variables M (the set of matched pairs) and U (the set of unmatched riders) as follows:

$$\begin{aligned}
\text{minimize} \quad & \left(\sum_{j \in U} \text{cost}(j) + \sum_{<j,k> \in M} \text{cost}(j,k) \right) \\
\text{subject to} \quad & 1.\quad M \subset F \wedge U \subset R \\
& 2.\quad R = U \cup \bigcup_{<j,k> \in M} \{j,k\} \\
& 3.\quad U \cap \bigcup_{<j,k> \in M} \{j,k\} = \emptyset \\
& 4.\quad \forall i,j,k \in R : < i,j > \in M \rightarrow (< j,k > \notin M \wedge (j \neq k \rightarrow < i,k > \notin M)),
\end{aligned} \tag{1}$$

where the objective function minimises the cost of all paired and solo journeys. Constraint 1 stipulates that all matched pairs are selected from feasible pairs, while the unmatched riders must belong to the set of riders. Constraint 2 stipulates that every rider in set R must be included in either a matched pair or the set of unmatched riders. Constraint 3 ensures that the matched pairs and unmatched riders are mutually exclusive sets, meaning that their intersection is empty. Constraint 4 stipulates that each rider can appear in at most one matched pair.

This model corresponds to both flexible roles and FreeMatch problems, where matched riders are picked up by a dedicated vehicle at their origin and taken to their destination using the shortest path. In FreeMatch, a vehicle is assumed to materialise instantly at the origin of its first rider and disappear when no longer needed, while in the version with flexible roles the vehicle is provided by one of the participants and stays with them.

The model is modified below to handle both driver and passenger roles. With fixed-role ride matching, the driver is responsible for initiating the trip and providing the vehicle for the passenger. In this variant, riders are divided into a set of drivers D and a set of passengers P. A match $< j,k >$ is now feasible only if $j \in D$ and $k \in P$. Consequently, it is computationally easier to generate the set F of feasible matches.

The model (1) for minimising VHT can be be modified as follows in order to cater to the scenario with fixed roles:

$$\begin{aligned}
\text{minimize} \quad & \left(\sum_{j \in U} \text{cost}(j) + \sum_{<j,k> \in M} \text{cost}(j,k) \right) \\
\text{subject to} \quad & 1. \quad M \subset F \wedge U \subset R \\
& 2. \quad R = U \cup \bigcup_{<j,k> \in M} \{j,k\} \\
& 3. \quad U \cap \bigcup_{<j,k> \in M} \{j,k\} = \emptyset \\
& 4. \quad \forall i,k \in D \wedge \forall j,l \in P : < i,j > \in M \rightarrow (< i,l > \notin M \wedge < k,j > \notin M).
\end{aligned} \tag{2}$$

Constraint 4 enforces the exclusivity of rider-driver pairings, implying that a driver assigned to a matched pair cannot be paired with another rider and vice versa.

4.3. Reachability Graphs

Many existing ridesharing studies use pruning mechanisms to speed up the matching of travellers. Approaches include the use of grids [33,34] and geometric shapes [8,13]. These approaches are approximations, and may miss candidates with longer journeys and more distant origins.

Here, we propose a pruning mechanism that preserves all possible matches. The reachability graph algorithm identifies and records the nodes that a rider r can reach without exceeding their deadline to reach their destination. Using the identified reachable nodes of each rider, the road graph is annotated with the arrival and departure times of those riders who can traverse each node of the road graph without missing their destination arrival deadline t_r^{la} for d_r. This provides a fast mechanism to identify candidates for matches.

Algorithm 1 illustrates the simple procedure that is called for each rider r. The network graph, origin o_r, destination d_r, earliest departure time t_r^{ed}, and latest arrival time t_r^{la} are passed to the procedure. It begins to traverse the road graph at o_r and examines the nodes in its immediate neighbourhood. For each node, the algorithm tests whether the destination can be reached within the required time considering the need to travel to the node from o_r and reach d_r after departing from the node (line 5). If the node is included in the reachability graph, then children need to be examined as well. If a node does not qualify, then its neighbours are not examined. This renders the algorithm very efficient.

If a node is reachable, the information of the rider is stored along with and their earliest arrival and latest departure times. The time interval that can be spent on the current node is determined based on the time (cost) $c(o_r, n)$ that it takes to reach it added to the earliest possible departure time t_r^{ed}. The end of the interval can be calculated by subtracting the time required to travel from the current node to the destination node $c(n, d_r)$ (line 6). This method guarantees the preservation of the optimal solution within the search space, as any additional deviation would result in r failing to meet the deadline.

4.4. Construction of the Match Graph

The construction of the match graph builds on the reachability graphs and road graph, using annotated nodes to indicate which riders can traverse at specific times. The goal is to compute all possible pairwise matches and store them as a graph in which every node represents a rider, every edge is a possible match, and the direction of an edge denotes the order of pickup. This graph builds on similar approaches used in other studies [15,35].

Algorithm 1: Creating a Reachability Graph

```
Data: $G$, (graph of road network), $r$, $o_r$, $d_r$,$t_r^{ed}$, $t_r^{la}$
Result: $N$, set of reachable nodes
1 $N \longleftarrow \emptyset$                                  /* Set initialisation */
2 $P \longleftarrow \{o_r\}$                 /* Set for nodes to process.  Start with $o_r$ */
3 for $n \in P$ do
4     $P \longleftarrow P \setminus \{n\}$
5     if $t_r^{ed} + c_{o_r,n} + c_{n,d_r} \leq t_r^{la}$ then /* $t_r^{la}$ can be met via $n$ */
          /* Store time and rider with the node */
6         $n \longleftarrow r, t_r^{ed} + c_{o_r,n}, t_r^{l} - c_{n,d_r}$
7         $N \longleftarrow n$
          /* Store neighbours of $n$ for examination */
8         $P \longleftarrow neighbours(n)$
```

Algorithm 2 initially builds the nodes of the match graph from the list of riders; then, for every rider r it retrieves all riders passing through r's origin o_r. This is a simple lookup operation, as every node holds the list of trips passing through it along with the corresponding times (line 4). To establish a possible match, the next step verifies which of the riders in these trips can either drop r to its destination or be dropped by r to their destination (line 6). When a match has been established, it is added as an edge between the corresponding riders, with the direction of the edge representing the order of pickup and the weight of the edge representing the savings the match will provide.

Algorithm 2: Creating the Match Graph

```
Data: $R$, (set of all riders), $N$, (reachability graphs)
Result: $MG$, match graph
1 $MG \longleftarrow R$                          /* Riders become nodes of $MG$ */
2 $C \longleftarrow \emptyset$                           /* Set for candidate matches */
3 for $r \in R$ do
      /* Store riders that can reach $o_r$ in candidate set $C$ */
4     $C \longleftarrow R \in N(o_r)$
5     for $c \in C$ do
6         if $c \in N(d_r)$ OR $r \in N(d_c)$ then /* $c$ can reach $r$'s destination or $r$
           can reach $c$'s destination */
              /* Add edge from c to r */
7             $MG \longleftarrow edge_{c,r}$
```

As we repeat these steps for every rider, all possible matches are found and stored. For every rider r's turn, only those matches are found where r is picked up. The other riders' turns take care of any such matches where r performs the pickup.

4.5. Optimality

Our optimisation model includes information on all potential rider pairings and all individual journeys, including their respective minimum costs. From this, we argue that an optimal mathematical solver will select a combination of paired rides and solo journeys that ensures the lowest overall cost.

Lemma 1. *SnapPair records all possible pairings of riders.*

Proof. We show that if it is possible for $r1$ to give a ride to $r2$, then $r2$ will appear in the list of matches for $r1$.

For our road graph G, we use Dijkstra's algorithm [32] to compute the shortest paths between all pairs of nodes u, v in the graph and to record the optimal time $w(u, v)$. Our reachability graph method records all possible nodes that $r1$ can visit en route to d_{r1}. This is determined by starting at o_{r1} and expanding (a) as follows: all neighbours of all locations which can be reached with time $t^{ed}_{r1} + w(o_r, n) + w(n, d_r) \leq t^{la}_{r1}$, where n denotes a node that is expanded during this traversal. From the correctness and completeness of Dijkstra's algorithm, all locations L satisfying (a) are returned and the returned set includes all locations that can be reached by r_1 en route to d_1. If rider $r2$ starts at any other location $n' \notin L$, then $t^{ed}_{r1} + w(o_r, n') + w(n', d_r) > t^{la}_{r1}$, $r1$ arrives late, and $r1$ cannot feasibly pick up $r2$ (i.e., $r2$ does not appear in the list of matches for $r1$). $\square$

Lemma 2. *SnapPair records the minimum cost for each individual journey or potential pairing.*

Proof. For any possible pairing between two riders $r1$ and $r2$, we show that our algorithm calculates the lowest-cost journey for both riders to reach their destinations in time. Suppose that rider $r1$ picks up another rider $r2$; when set L for $r1$ has been ascertained, SnapPair checks for each possible rider $r2$ that starts at a location $n \in L$, whether it is feasible for $r1$ to pick up $r2$, and if so, at what cheapest cost. The cost is determined by the route chosen for the journey. For each potential match $\langle r1, r2\rangle$, three routes are possible:

1. $route1(r1, r2) = \langle o_{r1}, o_{r2}, d_{r2}, d_{r1}\rangle$, having a cost of $cost_{route1}(r1, r2) = w(o_{r1}, o_{r2}) + w(o_{r2}, d_{r2}) + w(d_{r2}, d_{r1})$;
2. $route2(r1, r2) = \langle o_{r1}, o_{r2}, d_{r1}, d_{r2}\rangle$, having a cost of $cost_{route2}(r1, r2) = w(o_{r1}, o_{r2}) + w(o_{r2}, d_{r1}) + w(d_{r1}, d_{r2})$ and only if $(d_{r1} = o_{r2})$;
3. $route3(r1, r2) = \langle o_{r1}, d_{r1}, o_{r2}, d_{r2}\rangle$, having a cost of $cost_{route3}(r1, r2) = w(o_{r1}, d_{r1}) + w(o_{r2}, d_{r2})$.

The cost of $route3$ is equal to the cost of the two direct routes $route_{r1} = \langle o_{r1}, d_{r1}\rangle$ incurring $cost(r1)$ and $route_{r2} = \langle o_{r2}, d_{r2}\rangle$ with $cost(r2)$, and as such is subsumed by the solution in which $r1$ and $r2$ travel alone. For this reason, it is not considered. If neither $route1(r1, r2)$ or $route2(r1, r2)$ is feasible, $r2$ is not added to the list of possible riders for $r1$ to pick up. If either route is possible, then the corresponding cost is recorded and the match $\langle r1, r2\rangle$ is included in the match graph. If both routes are possible, then the minimum cost for it, denoted as $cost(r1, r2) = min(cost_{route1}, cost_{route2})$, is recorded along with shortest route. $\square$

Theorem 1. *The computed solutions are globally optimal.*

Proof. Lemmas 1 and 2 establish that the mathematical optimiser is provided with every conceivable individual journey and pairing, each accompanied by its lowest cost. The model's constraints guarantee that every rider is either chosen for a solo trip or included in a matched pairing. The optimiser returns the arrangement that best minimises costs, yielding the optimal solution. This solution is derived using a branch-and-bound approach, ensuring the return of an optimal combination, though achieving this guarantee may entail exploring all potential combinations. $\square$

4.6. Dynamic Optimisation and Rematching

Dynamic optimisation handles new demands within a rolling time horizon. When a new demand is introduced at time t, the system simulates the situation up to t and determines which existing riders can be considered for the optimisation iteration.

We introduce the notion of *slack* for riders, which is the difference between the latest and earliest possible arrival times. The *slack* for an unmatched rider is

$$slack_r = t^{la}_r - t^{ea}_r. \tag{3}$$

For each unmatched rider r, reoptimisation is possible if $t - t_r^{ed} < slack_r$. In this case, t_r^{ed} is updated to t.

A vehicle currently in transit with matched riders $\langle j, k \rangle$ may be eligible for multiple matches if the remaining rider is matched again after dropping off the first rider. Thus, the departure location and earliest departure time for matched riders $\langle j, k \rangle$ in transit are updated according to the current trip leg and position, as follows:

- If the current leg ends with d_j arriving there at time t_{arr}, then o_k is updated to d_j and t_k^{ed} is updated to t_{arr}; rider k is added to the set of drivers D and rider j is dropped from R.
- If the current leg ends with d_k arriving there at time t_{arr}, then o_j is updated to d_k and t_j^{ed} is updated to t_{arr}; rider j is added to the set of drivers D and rider k is dropped from R.

Riders may request a ride with advance notice of τ minutes, where τ denotes the time before their earliest departure time t_r^{ed} of a rider r becomes known to the system.

Newly arrived riders are added to R. The earliest departure time and origin for in-transit riders is updated to the current time t and the rider's current location. When the participants in the iteration have been determined, the reachability and match graphs are reconstructed while enforcing the constraint that

$$\forall \langle j, k \rangle \in R : k \in R_t \rightarrow \langle j, k \rangle \notin F,$$

where R_t denotes the riders in transit.

This implies that a modification of the match graph construction outlined in Algorithm 2 is implemented for riders already in transit. These riders are exempt from the need to seek potential candidates who can pick them up, as they are already in transit; thus, they are only considered for such matches where they will be performing the pickup.

For matching variants in which the participants provide vehicles, i.e., fixed roles and flexible roles, an additional constraint is enforced when constructing the match graph:

$$\begin{aligned} \forall \langle j, k \rangle \in R : j \in R_t \ \wedge \langle j, k \rangle \in F \rightarrow \\ route(j, k) = \langle o_j, o_k, d_k, d_j \rangle. \end{aligned}$$

This ensures that the vehicle stays with the participant who provided it. The constraint disallows routes such as $\langle o_j, o_k, d_j, d_k \rangle$, where the last person to be dropped off was not the one who brought the vehicle. Accordingly, the optimal pairing of riders that minimises the total cost, as in the model (1), results in a snapshot optimal solution.

4.7. Eager and Lazy Optimisation

Regardless of the type of matching, practitioners can choose whether matched riders should leave immediately or wait in situ for as long as their schedule allows. This "lazy optimisation" opens up the possibility of improving the match for better savings of VHT (Vehicle Hours Travelled).

In case "eager optimisation" is chosen for a match $\langle j, k \rangle$, the time of arrival at o_k is $t_j^{ed} + w(o_j, o_k)$. Similarly, the pair departs from o_k at $max(t_j^{ed} + w(o_j, o_k), t_k^{ed})$ and arrives at the first dropoff location by the shortest route.

However, for "lazy optimisation" the time of departure from o_j is the latest possible time consistent with arriving at both destinations in time. For example, on route $r1(j, k) = \langle o_j, o_k, d_k, d_j \rangle$, the departure time from o_j would be

$$t_j^{dep} = min(t_k^{la}, t_j^{la} - w(d_k, d_j)) - w(o_k, d_k) - w(o_j, o_k).$$

When reoptimising at time t, if $t < t_j^{dep}$, then o_j remains as before; however, if $t > t_j^{dep}$, then the match $\langle j, k \rangle$ cannot be changed. Thus, j and k are dropped from the set of riders R and begin their combined trip.

4.8. Dynamically Changing Edge Costs

Traffic congestion varies during the day, causing prerecorded costs for specific time periods to potentially become outdated. Thus, we examine the impact of link costs changing after each time window. While updating edge costs at each reoptimisation is slower, it provides optimal solutions within an acceptable timeframe for fleet operators. Dijkstra's algorithm [32] is used for all shortest path calculations, irrespective of whether the costs are precomputed or generated during the matching algorithm. While the majority of studies favour the first option, a few have incorporated shortest path calculations within the matching algorithm. However, these studies have typically conducted matching on a smaller scale or relied on heuristics [18,34].

In addition to recalculating all shortest paths, changing edge costs require recomputing the slacks for unmatched riders and the predicted departure and arrival times for matched riders. For lazy optimisation, dynamically changing edge costs may inevitably cause some riders who started their trips as late as possible to arrive late. However, for other riders the changed edge costs result in reoptimisations that enable them to arrive on time. Naturally, if the edge costs increase, the total VHT increases as well.

5. Experiments

5.1. Case Study

In our study, we used the New York City taxicab dataset [36] as a case study. This dataset is favoured for shared mobility studies due to its provision of GPS coordinates for requests instead of zones [37]. The road network in Manhattan, New York City consists of 4484 nodes and 8839 edges with an average edge distance of 186 m. Unlike previous studies that employed designated pickup/drop-off points [11], SnapPair considers every node for pickup and dropoff. For edge costs, we compute free-flow travel times in seconds by dividing edge length by the maximum allowed speed, following similar methods in other research [38]. Our travel demand consisted of 23,981 requests covering the hour-long morning peak from 8–9 a.m. on 1 May 2013, a working Wednesday. Under free-flow conditions, commuters traveling independently in their own vehicles would spend 1800 h in total transportation time. In our simulation, we introduced new demand every minute during a one-hour period. The notice period for all requests was set to one minute, mimicking the behavior of ride-hailing service users who request a ride when they need it. On average, each minute in the simulation included around 400 new riders, with a maximum of 444 new riders occurring in any minute. All requests originated and terminated within Manhattan. Each request was assumed to have one rider, following assumptions in previous work [15]. Vehicle capacity was capped at two passengers in all experiments. For experimental evaluation of the system, we first applied SnapPair to the classic one-to-one matching scenario, then to our new formulation called FreeMatch with vehicles provided. We produced results for both online and offline settings and with different slack levels achieved by modifying riders' latest arrival times. The resulting slack levels are percentages of the shortest rider journey. In the offline experiments, we assumed full advance knowledge of travel demand, which is a rarity in real-world situations. We followed prior research [39] to set a baseline optimum for our ridesharing system. Additionally, we assessed the following matching policies in order to present their impact on the system's performance:

- **Eager departure without rematching:** In this approach, as soon as the optimal set of matches for a specific time window is identified, they are immediately put into action; the initial riders in these matches begin their journey right away, and the successfully matched riders are not subject to rematching even if they become available after their match is dropped off. Unmatched riders, however, stay in the queue, awaiting future rounds of matching for as long as their deadlines allow.

- **Lazy departure without rematching:** Matched participants delay committing to the first optimal match offer if it remains valid in the next iteration, allowing them to potentially be paired with someone else for greater system-wide savings. If waiting would invalidate the match, participants commit and depart. Riders are not rematched even if they become available after dropping the other rider in the match.
- **Eager departure with rematching:** Under this policy, participants commit to the first presented optimal match, but can be rematched after the first rider in the match is dropped off (potentially multiple times). Unmatched riders wait until their slack runs out before departing on their own without being matched.
- **Lazy departure with rematching:** Matched participants delay committing to a match as long as it remains valid in the next iteration. They can be rematched after the first match. Unmatched riders wait until their slack is fully utilised while waiting to be matched, then depart on their own.

We ran the experiments on a Lenovo laptop with 16 GB RAM and a 1.80 GHz processor. The implementation was carried out in Java using the JgraphT library [40] for graph representations. To solve the linear programming formulations, we used Minizinc [41] and Gurobi [42].

5.2. One-to-One Ridematching with Fixed Roles

First, we solve the more established formulation where the roles of driver and passenger are fixed. Rematching is avoided to match the formulations in prior studies.

Table 1 shows the result of this matching approach. The slack column indicates how much extra time a traveller is prepared to spend on their trip as a percentage of the shortest travel time possible given their origin and destination. The time column indicates the average time in milliseconds for optimising a time window, calculated as the average optimisation time of the 60 time windows of one minute each in an hourly dataset, including the preoptimisation steps. Matches denotes the number of matches made between pairs of travellers and VHT the vehicle hours travelled.

Table 1. Eager departure compared to lazy departure; roles are fixed.

Slack	Time (Milliseconds)		Matches		VHT (h)	
	Eager	Lazy	Eager	Lazy	Eager	Lazy
10%	**325**	334	**356**	**356**	**1781**	**1781**
20%	**372**	388	**1494**	**1494**	**1707**	**1707**
30%	**447**	467	**3151**	3146	**1607**	**1607**
40%	**554**	642	**4803**	4774	**1521**	**1521**
50%	**690**	860	**6132**	6112	1464	**1461**

Lazy departure leaves matched drivers waiting at their respective origins until their slack runs out. During this time, the pairs may be split up and matched again if this leads to greater savings than the original match. It is no surprise that the eager approach takes less time to optimise, as fewer riders remain for the next iteration. The VHT results for the lazy and eager approaches are nearly identical here. This similarity is primarily due to the comparable number of potential matches between the two approaches, especially at lower levels of slack. However, a significant difference emerges at 50% slack, where the eager policy results in an average of 443 potential matches per minute while the lazy approach offers 688 potential matches every minute to choose from, as shown in Table 2, which shows the average numbers of riders and potential matches for every optimisation window.

Table 2. Average number of riders and potential matches during a time window; roles are fixed.

Slack	Average Riders		Potential Matches	
	Eager	Lazy	Eager	Lazy
10%	**409**	**409**	**7**	**7**
20%	519	**520**	**36**	**36**
30%	612	**621**	109	**120**
40%	660	**709**	244	**308**
50%	681	**749**	443	**688**

5.3. One-to-One Ride-Matching with Flexible Roles

Tables 3 and 4 present the results of the experiments that match travellers one-to-one while assuming that the rider one who starts the journey provides the vehicle (i.e., "flexible roles").

Table 3. Eager departure compared to lazy departure; the driver is decided after the match and there is no rematching after the first drop-off.

Slack	Time (Milliseconds)		Matches		VHT (h)	
	Eager	Lazy	Eager	Lazy	Eager	Lazy
10%	**371**	397	**1316**	**1316**	**1724**	**1724**
20%	**497**	509	4112	**4113**	**1547**	**1547**
30%	**708**	751	**6819**	6803	1405	**1404**
40%	**1016**	1362	**8501**	8463	1337	**1328**
50%	**1413**	2985	**9553**	9451	1307	**1287**

Table 4. Eager departure compared to lazy departure; the driver is decided after the match and may be rematched after dropoff.

Slack	Time (Milliseconds)		Matches		VHT (h)	
	Eager	Lazy	Eager	Lazy	Eager	Lazy
10%	**362**	395	**1321**	**1321**	**1722**	**1722**
20%	**470**	478	**4147**	4146	**1534**	1535
30%	**696**	738	**6953**	6925	1389	**1384**
40%	**1049**	1373	**8800**	8661	1317	**1304**
50%	**1518**	2742	**10,016**	9715	1287	**1253**

Table 3 shows the results of the algorithm that does not match a driver with a new passenger after they have dropped off the previous passenger. At up to 30% slack, the optimisation of a time window takes less than a second for an average of over 400 travellers per time window. Although eager departure produces slightly (at most 1%) more matches, lazy departure provides better cost savings, suggesting that the quality of the matches is better. The differences between the gains in VHT is meager at up to 30% slack, with meaningful differences only observed above 40% slack. Compared to fixed roles, in the case of 50% slack the policy of flexible roles sometimes provides more than three times the number of matches, and at worst one and half times the number of matches. The VHT is between 3% and 12% shorter when compared to the flexible policy without rematching.

Table 4 shows the results when the algorithm differs from Table 3 in that it rematches the drivers after they have dropped off the rider they were previously matched with. While rematching does not appear to affect the runtime, it leads to significant (2–34 h) savings in VHT. Overall, rematching leads to a 3% increase in the number of matches at 50% slack when comparing the lazy option with rematching to the lazy option without rematching (9715 vs. 9451). This saves 34 h of system travel time (1253 compared to 1287).

5.4. Ridematching with Vehicles Provided (FreeMatch)

Tables 5 and 6 contain the same result columns as Tables 3 and 4, except under the assumption that the vehicle is provided by the system rather than belonging to either traveller.

Assuming vehicle provision allows for more matches by permitting the same traveler to start and end their journey first. On average, this results in 2.13 times as many potential matches per iteration compared to the version with flexible roles. Consequently, it is expected that both eager and lazy departure will have longer running times. A comparison of the run times in Table 5 (compared to Table 3) and Table 6 (compared to Table 4) illustrates this.

The assumption that vehicles appear when needed without the need to designate a driver leads to significant further savings in travel time. As a consequence, the eager policy creates slightly more savings in travel times for 20% and 30% slack, suggesting that there are plenty of good options to choose from without reoptimising existing pairs of riders.

Table 5. Eager departure compared to lazy departure; the vehicle is provided and there is no rematching after the first dropoff.

Slack	Time (Milliseconds)		Matches		VHT (h)	
	Eager	Lazy	Eager	Lazy	Eager	Lazy
10%	**391**	397	**1929**	**1929**	**1692**	**1692**
20%	**587**	612	**5339**	5245	**1489**	1494
30%	**995**	1134	**7966**	7743	**1357**	1361
40%	**1621**	2455	**9420**	9119	1300	**1297**
50%	**2424**	6067	**10,271**	9764	1279	**1269**

Table 6. Eager departure compared to lazy departure; the vehicle is provided and the remaining rider may be rematched after the first is dropped off.

Slack	Time (Milliseconds)		Matches		VHT (h)	
	Eager	Lazy	Eager	Lazy	Eager	Lazy
10%	**390**	394	**1966**	**1966**	**1689**	**1689**
20%	**607**	614	**5594**	5479	**1474**	1478
30%	**1064**	1181	**8536**	8221	**1335**	1339
40%	**1819**	2735	**10,338**	9706	1274	**1267**
50%	**2723**	7147	**11,519**	10,529	1251	**1226**

The best results (a VHT of 1226) when the vehicle is provided as a service are achieved when allowing 50% slack and applying a lazy policy with rematching. The second-best option only differs in the way the vehicle is provided; requiring a driver while applying 50% slack and a lazy policy with rematching achieves a VHT that is 27 h, or 2% longer (1253 h), whereas providing a vehicle and refraining from rematching takes 43 h longer. These results suggest that the most significant contributors to the savings are: (1) rematching; (2) providing a vehicle; and (3) lazy departure.

5.5. Comparison with Static Formulation

Approaches using a dynamic rolling time window typically yield inferior results compared to static offline approaches that have complete information; however, offline optimisation may not always be feasible in ridesharing due to last-minute travel decisions by riders. Nonetheless, offline optimisation results can be used as a baseline. Figure 1 displays a comparison between a subset of policy combinations and 40% slack against the results of offline optimisation using the same dataset. All results apply the lazy departure approach and demonstrate small differences compared to offline optimum across the four settings (left to right). For flexible roles without rematching, the online formulation is only 7.18% costlier than the offline optimum. When the vehicle is provided by a service and there is no rematching, this difference is 7.74%. For flexible roles with rematching, this value is 5.24%. Finally, when the vehicle is provided by a service and rematching is used, the online performance is only 5.23% from the true optimum.

5.6. Impact of Shortest Path Calculations on the Matching Algorithm's Performance

The previous experiments used prestored shortest path calculations. This is a reasonable approach in real-world settings, where average travel times on urban roads during specific hours are predictable. However, we now consider a scenario where shortest path calculations are necessary due to unavailability of prestored link costs and the cost of each trip must be predicted at the start using the current link cost information. For these simulations, we increased the cost of half the links by 0.5% every minute. In this scenario, the reachability and match graphs are constructed using a travel function that accounts for real-time travel time rather than relying on initial shortest path calculations. Compared to the other experiments with an overall transport cost of 1800 h, the altered link costs result in an increase to 1928 h.

Table 7 shows that the matching algorithm performs well with the eager departure policy, even under these challenging conditions. It would arguably be considered acceptable for real-life matching service providers, taking a maximum of 19.5 s to optimally match a minute's worth of demand.

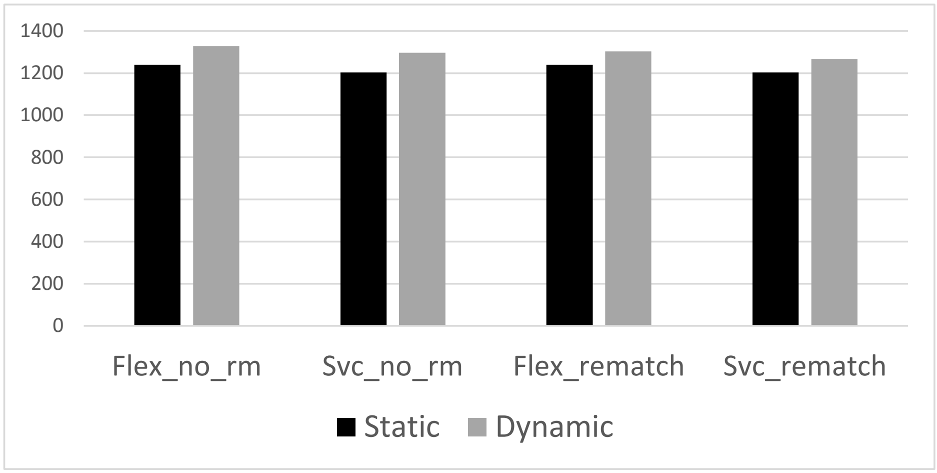

Figure 1. Comparison with static.

Table 7. Eager departure applied to changing link costs; the vehicle is provided and the remaining rider may be rematched after the first is dropped off. Shortest path calculations are performed within the matching algorithm.

Slack	Time (Milliseconds)	Matches	VHT (Hours)
30%	13,044	6654	1499
40%	18,118	9156	1357
50%	19,575	10,731	1293

6. Conclusions

Matching drivers to riders through a ridesharing service presents a scalability challenge. Previous solutions have broken the problem down into distinct regions, reduced the number of pickup points, or used incomplete search methods. Having precomputed the shortest path cost between each pair of points in the ridesharing area, this paper presents an algorithm that links each rider to all of their possible matches. Consequently, even when new ridesharing requests arise every minute, optimal pairings can be computed in seconds. Indeed, the computation time is so short that it is possible to recompute the shortest paths dynamically at each time point and successfully handle dynamic link costs while retaining optimality computed within 20 s. This highly efficient implementation supports extensions to the basic ridesharing matching service, with experimental results showing a potential contribution to congestion reduction on city streets. First, if riders are flexible about using their own vehicle or riding as a passenger, savings can be increased by up to 12%. Second, when the "slack time" overhead for sharing increases from 10% to 50%, the number of matches increases five-fold, which reduces the total vehicle hours travelled (VHT) by 27%. Third, if the vehicle used for transport is not linked to any of the passengers (by using an automated vehicle, for example), the same vehicle can match sets of up to seven riders even with a limit of two occupants at any time. In this case, the VHT cost is reduced by over a third for lower slack times. For 50% slack there is some additional reduction, though at most only 10% more.

Finally, we compared two separate policies and the presence of absence of rematching:

- Eager or Lazy
  A trip can either be started "eagerly" as soon as a match is found, or delayed "lazily", in which case the slack time enables the trip to be delayed until the next update of demands (after one minute) to check for a better option.
- Rematching
  When one of a pair of travellers has been dropped off, the remaining traveller can be paired again with a new traveller.

Among these improvements, the rematching approach has the most beneficial impact on the total hours travelled. Providing a vehicle from an outside service was the next most impactful measure. Our experiments showed significant benefits of larger slack times for all studied ridesharing variations. Automated vehicles offer substantial advantages with low slack times. Additionally, lazy starting and rematching yield comparable benefits, with rematching showing slightly better savings in terms of vehicle hours travelled when using automated vehicles.

Author Contributions: Conceptualization, A.H., M.W., D.D.H. and I.M.; methodology, A.H., M.W., D.D.H. and I.M.; software, A.H.; validation, A.H.; formal analysis, A.H., M.W. and I.M.; investigation, A.H., M.W., D.D.H. and I.M.; resources, A.H., M.W. and I.M.; data curation, A.H.; writing—original draft preparation, A.H., M.W., D.D.H. and I.M.; supervision, M.W. and I.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Australian Government through the Australian Research Council's Discovery Projects funding scheme (project DP190100013).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Goodwin, P. *The Economic Costs of Road Traffic Congestion*; UCL (University College London), The Rail Freight Group: London, UK, 2004.
2. Martins, L.d.C.; de la Torre, R.; Corlu, C.G.; Juan, A.A.; Masmoudi, M.A. Optimizing ride-sharing operations in smart sustainable cities: Challenges and the need for agile algorithms. *Comput. Ind. Eng.* **2021**, *153*, 107080. [CrossRef]
3. Tafreshian, A.; Masoud, N.; Yin, Y. Frontiers in service science: Ride matching for peer-to-peer ride sharing: A review and future directions. *Serv. Sci.* **2020**, *12*, 44–60. [CrossRef]
4. Hopcroft, J.E.; Karp, R.M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.* **1973**, *2*, 225–231. [CrossRef]
5. Fredman, M.L.; Tarjan, R.E. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* **1987**, *34*, 596–615. [CrossRef]
6. Edmonds, J. Maximum matching and a polyhedron with 0, 1-vertices. *J. Res. Natl. Bur. Stand. B* **1965**, *69*, 55–56. [CrossRef]
7. Gabow, H.N.; Tarjan, R.E. Faster scaling algorithms for general graph matching problems. *J. ACM* **1991**, *38*, 815–853. [CrossRef]

8. Masoud, N.; Jayakrishnan, R. A real-time algorithm to solve the peer-to-peer ride-matching problem in a flexible ridesharing system. *Transp. Res. Part B Methodol.* **2017**, *106*, 218–236. [CrossRef]
9. Chen, W.; Mes, M.; Schutten, M.; Quint, J. A ride-sharing problem with meeting points and return restrictions. *Transp. Sci.* **2019**, *53*, 401–426. [CrossRef]
10. Agatz, N.; Erera, A.; Savelsbergh, M.; Wang, X. Optimization for dynamic ride-sharing: A review. *Eur. J. Oper. Res.* **2012**, *223*, 295–303. [CrossRef]
11. Tafreshian, A.; Masoud, N. Trip-based graph partitioning in dynamic ridesharing. *Transp. Res. Part C Emerg. Technol.* **2020**, *114*, 532–553. [CrossRef]
12. Shen, B.; Huang, Y.; Zhao, Y. Dynamic ridesharing. *Sigspatial Spec.* **2016**, *7*, 3–10. [CrossRef]
13. Xu, Y.; Qi, J.; Borovica-Gajic, R.; Kulik, L. Geoprune: Efficiently matching trips in ride-sharing through geometric properties. In Proceedings of the 32nd International Conference on Scientific and Statistical Database Management, Vienna, Austria, 7–9 July 2020; pp. 1–12.
14. Pelzer, D.; Xiao, J.; Zehe, D.; Lees, M.H.; Knoll, A.C.; Aydt, H. A partition-based match making algorithm for dynamic ridesharing. *IEEE Trans. Intell. Transp. Syst.* **2015**, *16*, 2587–2598. [CrossRef]
15. Alonso-Mora, J.; Samaranayake, S.; Wallar, A.; Frazzoli, E.; Rus, D. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proc. Natl. Acad. Sci. USA* **2017**, *114*, 462–467. [CrossRef] [PubMed]
16. Najmi, A.; Rey, D.; Rashidi, T.H. Novel dynamic formulations for real-time ride-sharing systems. *Transp. Res. Part E Logist. Transp. Rev.* **2017**, *108*, 122–140. [CrossRef]
17. Ketabi, R.; Alipour, B.; Helmy, A. Playing with matches: Vehicular mobility through analysis of trip similarity and matching. In Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, 6–9 November 2018; pp. 544–547.
18. Ta, N.; Li, G.; Zhao, T.; Feng, J.; Ma, H.; Gong, Z. An efficient ride-sharing framework for maximizing shared route. *IEEE Trans. Knowl. Data Eng.* **2017**, *30*, 219–233. [CrossRef]
19. Li, Y.; Chung, S.H. Ride-sharing under travel time uncertainty: Robust optimization and clustering approaches. *Comput. Ind. Eng.* **2020**, *149*, 106601. [CrossRef]
20. Kleiner, A.; Nebel, B.; Ziparo, V. A mechanism for dynamic ride sharing based on parallel auctions. In Proceedings of the 22th International Joint Conference on Artificial Intelligence (IJCAI), Barcelona, Spain, 16–22 July 2011.
21. Nourinejad, M.; Roorda, M.J. Agent based model for dynamic ridesharing. *Transp. Res. Part C Emerg. Technol.* **2016**, *64*, 117–132. [CrossRef]
22. Aissat, K.; Oulamara, A. Dynamic ridesharing with intermediate locations. In Proceedings of the 2014 IEEE Symposium on Computational Intelligence in Vehicles and Transportation Systems (CIVTS), Orlando, FL, USA, 9–12 December 2014; pp. 36–42.
23. Fiedler, D.; Čertickỳ, M.; Alonso-Mora, J.; Pěchouček, M.; Čáp, M. Large-scale online ridesharing: The effect of assignment optimality on system performance. *J. Intell. Transp. Syst.* **2022**, *28*, 189–210. [CrossRef]
24. Fielbaum, A.; Bai, X.; Alonso-Mora, J. On-demand ridesharing with optimized pick-up and drop-off walking locations. *Transp. Res. Part C Emerg. Technol.* **2021**, *126*, 103061. [CrossRef]
25. Wang, J.; Cheng, P.; Zheng, L.; Chen, L.; Zhang, W. Online Ridesharing with Meeting Points. *Proc. VLDB Endow.* **2022**, *15*, 3963–3975. [CrossRef]
26. Lu, W.; Quadrifoglio, L.; Lee, D.; Zeng, X. The ridesharing problem without predetermined drivers and riders: Formulation and heuristic. *Transp. Lett.* **2023**, *15*, 969–979. [CrossRef]
27. Furuhata, M.; Dessouky, M.; Ordóñez, F.; Brunet, M.E.; Wang, X.; Koenig, S. Ridesharing: The state-of-the-art and future directions. *Transp. Res. Part B Methodol.* **2013**, *57*, 28–46. [CrossRef]
28. Jabbari, P.; MacKenzie, D. Ride sharing attitudes before and during the COVID-19 pandemic in the United States. *Transp. Find.* **2020**, *26*. [CrossRef]
29. Shokouhyar, S.; Shokoohyar, S.; Sobhani, A.; Gorizi, A.J. Shared mobility in post-COVID era: New challenges and opportunities. *Sustain. Cities Soc.* **2021**, *67*, 102714. [CrossRef] [PubMed]
30. Hansen, T.; Sener, I.N. Strangers On This Road We Are On: A Literature Review of Pooling in On-Demand Mobility Services. *Transp. Res. Rec.* **2023**, *2677*, 1368–1381. [CrossRef]
31. Uber Technologies, Inc. UberX Share. Available online: https://www.uber.com/gb/en/ride/uberx-share/ (accessed on 17 March 2024).
32. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
33. Ma, S.; Zheng, Y.; Wolfson, O. T-share: A large-scale dynamic taxi ridesharing service. In Proceedings of the 2013 IEEE 29th International Conference on Data Engineering (ICDE), Brisbane, QLD, Australia, 8–12 April 2013; pp. 410–421.
34. Thangaraj, R.S.; Mukherjee, K.; Raravi, G.; Metrewar, A.; Annamaneni, N.; Chattopadhyay, K. Xhare-a-ride: A search optimized dynamic ride sharing system with approximation guarantee. In Proceedings of the 2017 IEEE 33rd International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19–22 April 2017; pp. 1117–1128.
35. Santi, P.; Resta, G.; Szell, M.; Sobolevsky, S.; Strogatz, S.H.; Ratti, C. Quantifying the benefits of vehicle pooling with shareability networks. *Proc. Natl. Acad. Sci. USA* **2014**, *111*, 13290–13294. [CrossRef] [PubMed]
36. Donovan, B.; Work, D. *New York City Taxi Trip Data (2010–2013)*; Technical Report; University of Illinois Urbana-Champaign: Champaign, IL, USA, 2014.

37. Qin, Z.T.; Zhu, H.; Ye, J. Reinforcement learning for ridesharing: An extended survey. *Transp. Res. Part C Emerg. Technol.* **2022**, *144*, 103852. [CrossRef]
38. Mahéo, A.; Zhao, S.; Hassan, A.; Harabor, D.D.; Stuckey, P.J.; Wallace, M. Customised Shortest Paths Using a Distributed Reverse Oracle. In Proceedings of the International Symposium on Combinatorial Search, Gugangzhou, China, 26–30 July 2021; Volume 12, pp. 79–87.
39. Agatz, N.; Erera, A.L.; Savelsbergh, M.W.; Wang, X. Dynamic ride-sharing: A simulation study in metro Atlanta. *Procedia-Soc. Behav. Sci.* **2011**, *17*, 532–550. [CrossRef]
40. Michail, D.; Kinable, J.; Naveh, B.; Sichi, J.V. JGraphT–A Java Library for Graph Data Structures and Algorithms. *ACM Trans. Math. Softw.* **2020**, *46*, 1–29. [CrossRef]
41. Nethercote, N.; Stuckey, P.J.; Becket, R.; Brand, S.; Duck, G.J.; Tack, G. MiniZinc: Towards a standard CP modelling language. In Proceedings of the International Conference on Principles and Practice of Constraint Programming, Providence, RI, USA, 23–27 September 2007; pp. 529–543.
42. Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*; Gurobi Optimization, LLC: Beaverton, OR, USA, 2022.