Ride Technical Report TR-2. May 2015.
CONFIDENTIAL. Please do not distribute.

Unconstrained Route Matching Algorithm

Gleb Chuvpilo[1]*, Oscar Salazar[2], Sanny Liao[3], Sergio Botero[4], Nicolás Hock[5]

Abstract
This technical report describes the *Unconstrained Route Matching Algorithm*, a proposed replacement of the *Route Matching Algorithm* [1] currently used in production at Ride. The main difference between the two algorithms is that *Unconstrained Route Matching Algorithm* replaces the concept of the *neighborhood* that limits the search of potential matches to an area close to the origin of the trip with a *channel* along the route from driver's origin to destination. This opens up the search space and thus allows to feed the matching funnel with a greater number of potential matches. The new algorithm preserves the rest of the features of the *Route Matching Algorithm*, specifically: takes into consideration varying amounts of overlap between a potential driver and a potential passenger's desired travel schedule, produces optimal routes that originate at a driver's home location, visit all of the driver's passengers in the most efficient order, and end at the work location, all while providing global optimality. Finally, we expand on the metrics for measuring algorithm quality. They define objective criteria to allow apples-to-apples comparisons across algorithms, and enable future research in the space.

Keywords
Matching — Clustering — Scheduling

[1] *Vice President of Research and Development, Ride*
[2] *Chief Product and Technology Officer, Ride*
[3] *Data Scientist, Ride*
[4] *Software Engineer, Ride*
[4] *Software Engineer, Ride*
***Corresponding author**: gleb@ride.com

Contents

Introduction

We have seen vast success of the *Route Matching Algorithm*[1] implemented and deployed in production at Ride, especially compared to *Simple Agglomerative Clustering Algorithm* used in *Ride Marketplace Alpha*. *Route Matching Algorithm* forms globally-optimal pickup routes that drivers would themselves form had they been aware of each others' existence: the algorithm takes into consideration varying amounts of overlap between a potential driver and a potential passenger's desired travel schedule, produces optimal routes that originate at a driver's home location, visit all of the driver's passengers in the most efficient order, and end at the work location, all while providing global optimality. *Route Matching Algorithm* showed success and started forming rides from Day 1 of the public launch. However, it comes with an intentional simplification that allowed for fast research and development time. This simplification limits matches to passengers who (1) live in the driver's immediate vicinity called *neighborhood* and (2) go to a destination within a short walking distance of the driver's destination. In other words, the algorithm is highly constrained, and what we really want is to relax these constraints. This Technical Report describes a route matching algorithm with these constraints removed, and we call it the *Unconstrained Route Matching Algorithm*. The discussion that follows first addresses the overall algorithm structure, then dives into the building blocks, and finally discusses how we can measure algorithm quality.

1. Algorithm Design

1.1 Overview

The *Unconstrained Route Matching Algorithm* has two main components: the *Matcher* specified in pseudocode in algorithm 1, and the *Global Optimizer* specified in algorithm 2. The role of the *Matcher* is to produce all possible match pairs of a user u_i with other users u_j: any new user can get matched to multiple other users, with u_i getting assigned either the role of a driver or a passenger. The role of the matcher is to find all these matches, and record them in a matrix. In other words, *Matcher* produces *locally-optimal matches*. These matches are then submitted to the *Global Optimizer*. The role of the *Global Optimizer* is to analyze all possible passenger-driver match pairs, and produce *globally-optimal matches*. This dual structure is needed, because we no have the luxury of *one-to-many* search by driver's destination as constrained by the *Route Matching Algorithm*. Now we have to deal with world of *many-to-many* matches instead.

1.2 Matcher

Algorithm 1: Matcher

```
input  : $U$ Users
output : $S$ Score Matrix
1 begin
2     for $\forall u_i \in U$ do
3         $C \longleftarrow \emptyset$
4         $t_i \longleftarrow ComputeTopologyConstraints(u_i)$
5         $s_i \longleftarrow ComputeScheduleConstraints(u_i, U)$
6         $c_i \longleftarrow ComputeNetwork(u_i, U)$
7         $S_i \longleftarrow CombineScores(t_i, s_i, c_i)$
8         $return(S)$
```

We feed the *Matcher* with any new user information, i.e. creation of a new user or an update of the existing user (line 2). On this new information, we first initialize the list of candidates to an empty set (line 3). Then we compute topology constraints (line 4), schedule constraints (line 5), and network constraints, such as limiting matching to a single employer or constraining by user's social graph (line 6). We then combine the aforementioned constraints into a combined score (line 7), and finally produce (line 8) and return (line 9) a *Score Matrix*, in Table 1.

u_1	u_2	…	u_n
u_2			
…			
u_n			

Table 1. Score Matrix: rows are driver roles, columns are passenger roles

1.3 Global Optimizer

Algorithm 2: Global Optimizer

```
input  : $S$ Score Matrix
output : $M$ Matches, $S'$ Updated Score Matrix
1  begin
2      $M \longleftarrow \emptyset$
3      for $\forall s_{i,j} \in S$ do
4          $O \longleftarrow Optimize(S)$
5          $D \longleftarrow AssignDriverRole(O)$
6          for $d_i \in D$ do
7              $v_i \longleftarrow VehicleCapacity(d_i)$
8              while $TotalPassengers(d_i) < v_i$ do
9                  $W \longleftarrow SolveTSPWaypoints(D)$
10                 $M \longleftarrow AddMatch(W)$
11     $return(M)$
```

We feed the *Score Matrix* arriving from the matcher to the *Global Optimizer*. We initialize this algorithm with an empty set of matches (line 2), and start the critical section with the *for* loop (line 3). Next is the core of global optimization, where we optimize roles and waypoints in the system by taking into account all new user and match information (line 4). We then assign driver roles (line 5). Then, for each designated driver (line 6), we calculate available vehicle capacity (line 7) and attempt to fill this capacity with new passengers (lines 8-10), which includes solving the Traveling Salesman Problem (line 9). We then produce (line 10) and return (line 11) resulting matches.

2. Algorithm Details

2.1 Reachability Channel

We define a term *Reachability Channel* as the bounding polygon around any given user's route from *origin* to *destination*, subject to the following constraint: any point inside the channel should be reachable as an additional waypoint for that user while still complying with *business rules*, such as, for instance, staying within 20% extra driving time. Precomputing and storing the geometry of *Reachability Channels* allows to calculate all possible channels that any give user belongs to in time complexity of $O(n)$.

2.2 Distance Score

To proxy for route optimality, we construct a distance score for each potential passenger relative to a driver. Given destination address D_d, passenger address D_i, and driver address D_j, the distance score for each potential passenger i with respect to driver j is:

$$DistanceScore_j(passenger_i) = \omega(D_d, D_i, D_j) \quad (1)$$

2.3 Schedule Score

2.3.1 Overview

Schedule mismatch poses an equally important challenge to carpooling. When users have non-recurring trips, the algorithm accounts for schedule mismatch using a mask to find users with compatible anchor times for the trips concerned, where anchor time can either be arrival time or departure time. When users have recurring trips in a given time span, in addition to schedule mismatch for a single trip, the algorithm also takes into consideration the frequency at which two travel schedules are compatible. Optimizing for schedule mismatch necessities a compromise in route optimality, we describe a simple method to parametrize the tradeoffs.

2.3.2 Anchor Time

Depending on the direction of travel, each traveler has an anchor time in the form of either desired arrival time or departure time. Suppose a driver has an anchor time of A_j, we fix the driver's anchor time and look for a set of passengers whose anchor time falls within x minutes of the driver's anchor time. In other words, driver j has a mask M_j:

$$M_j = [A_j - x, A_j + x] \quad (2)$$

For each driver j, we evaluate each user who can be a passenger against the mask M_j to createa a consideration set of passengers C_j. For non-recurring trips, C_j can be used directly by the *Route Matching Algorithm* to construct optimal carpools around each driver.

2.3.3 Computing the Schedule Score

For users with recurring trip, we construct an overlap score to capture the degree to which a passenger's desired set of trip overlaps with that of a potential driver.

Let T_i be passenger i's desired schedule, and T_j be driver j's desired schedule, where

$$T_i = [s_i(t_1), s_i(t2), ..., s_i(t_S)] \quad \text{for a passenger with S desired trips} \quad (3)$$

$$T_j = [s_j(t_1), s_j(t_2), ..., s_j(t_T)] \quad \text{for a driver with T desired trips} \quad (4)$$

The overlap of passenger i's schedule to that of driver j's can be measured as:

$$ScheduleScore_j(passenger_i) = \chi(T_i, T_j) \quad (5)$$

2.4 Network Score

We are introducing *Network Score*, a new score that did not exist in the *Route Matching Algorithm*. This score allows to describe the following use cases:

- Limiting matching to a single organization
- Introducing a bias to match within a user's organization or within a user's social graph

2.5 Combining Scores

We parametrize the tradeoff between schedule overlap, route optimality and network score using a function that computes a custom function of these three scores, e.g., a weighted average:

$$EligibilityScore_j(passenger_i) = \psi(ScheduleScore_j(i), DistanceScore_j(i), NetworkScore_j(i)) \quad (6)$$

Eligibility score is increasing in both overlap score and distance score. The relative effect of either overlap score or distance score on eligibility score may be fixed, or may vary based on outside factors, such as travel times, relationship between two carpoolers, or other individualized characteristics. Using the eligibility score, we rank all users in the consideration set C_j for each driver, and use the top n potential passengers as inputs for the *Unconstrained Route Matching Algorithm.*

3. Measuring Algorithm Quality

3.1 Overview

We would like to be able to quantify improvements in each algorithm update going forward. There are three main ways in which we would like to measure algorithm quality:

1. Post-match trips in a day
2. Probability of match acceptance
3. Expected trips in a day

We propose to start using metric (1) and a simplified version of metric (2) right away. In the meantime, we should start collecting data on other components for (2) and (3) for future improvements.

3.2 Post-match trips in a day

The macro goal here is to reduce the number of cars on the road. We want to measure that. Let's define a trip as one driver who drives from point A to point B in a day, irrespective of whether a driver has passengers or not. The following example provides three scenarios[1] :

Table 2. Scenario Analysis

	Travel Alone	Algo 1	Algo 2
Number of Trips	T_0	T_1	T_2

In this example:

[1] *Travel Alone* is a base case when none of the employees carpool

- Let T_0 be the number of trips that would be taken for a fixed number of employees to commute from home to work and back when traveling alone
- When Ride pools employees into carpools, T_s trips are formed, where $T_s \leq T_0 \forall s \neq 0$
- Algorithm 2 is superior to Algorithm 1 iff $T_2 < T_1$

3.3 Probability of users accepting a match

We want to maximize the probability that users will accept the matches that we have formed for them. To do this, we need to consider a few factors:

- Utility of saving money to the user, $U(dollar)$
- Utility of driving or riding with a fellow carpooler, this can include everything from companionship to environment impact, $U(carpool)$
- Utility of driving or riding a different car over his/her own, $U(cartype)$
- Utility of the inconvenience that carpooling produces, $U(inconvenience)$

Putting them together, we have:

$$\begin{aligned} &Prob(\text{user i accepts match}) = \\ &Prob(U_i(dollar) + U_i(carpool) + U_i(cartype) > \\ &\qquad U_i(inconvenience)) \end{aligned} \tag{7}$$

For the time being, we will stay agnostic toward how each of our users value dollar, carpool, car type, and inconvenience. However, we will try to measure inconvenience as a function of difference in schedule and travel time. Generically, inconvenience can be modeled as:

$$\begin{aligned} &Inconvenience = \\ &\qquad \alpha * F(\Delta traveltime) + \beta * G(\Delta schedule), \end{aligned} \tag{8}$$

where $\alpha + \beta = 1$
Functions F and G are defined as:

$$F(\Delta traveltime) = \ln(\tfrac{TTWC - TTWDA}{TTWDA} + 1) \tag{9}$$

where

- $TTWC$ is *travel time when carpooled*
- $TTWDA$ is *travel time when driving alone*

$$G(\Delta schedule) = \ln(\tfrac{\text{DAT-CAT}}{\text{(60 minutes)}} + 1) \tag{10}$$

where

- DAT is *desired anchor time*
- CAT is *carpool anchor time*

In summary, instead of comprehensively modeling the probability that a user will accept a match, we will use a metric that simply captures the inconvenience that a match would have causes a user. We will operate under the assumption that the less inconvenient a match is, the more like he/she will accept the match.

3.4 Expected trips in a day

Given T_s, the number of trips that a matching algorithm forms in a day, and the probability that each user accepts the match formed for her, we can comprehensively capture the quality of an algorithm as the expected number of trips to take place each day:

$$\begin{aligned} &\text{Quality of algorithm m} = \\ &\text{Expected number of trips in a day} = \\ &\qquad \sum_{1}^{T_m} 1 \times Prob(\text{users in ride i accept match}) \end{aligned} \tag{11}$$

where $T_m =$ the total number of trips formed by algorithm m

4. Discussion

One of the potential improvements to the app is multi-match functionality, so that users can see multiple matches simultaneously as cards that they can scroll through. In *Unconstrained Route Matching Algorithm* we compute a *Score Matrix* for all the users, so we can produce multi-matches out of the box.

References

[1] Gleb Chuvpilo, Sanny Liao, Oscar Salazar, and Sergio Botero. Route matching algorithm. Technical Report RIDE-TR-1, Ride, New York, NY, October 2014.