[Public] Fare Leg Join Rules vs Fare Transfer Rules

Skander Chouchene

I. Why add transfer predicates to Fare Leg Join Rules?

We want to add transfer predicates to `fare_leg_join_rules.txt`, mainly: `duration_limit`, `duration_limit_type`, and maybe a variant of `transfer_count` later on.

`fare_leg_join_rules.txt`was initially defined to consider multiple legs as one effective leg, for the purposes of matching in `fare_leg_rules.txt`. This would help with:

- Multi-leg sub-journeys where the only determinant of the fare is the first and last stop/zone.
- Rail sub-journeys where the transfer occurs in a station with no turnstiles.
- Bus sub-journeys with silent transfers.

This was done initially without limits on the duration of the sub-journey. However, Fare Leg Join Rules were adopted with flexibility in mind. It can help define:

- Distance-base fares that are applied to the total distance travelled, and not applied leg-by-leg.
- Cumulative zonal fares that look at the zones travelled in the whole journey, regardless of how many times they are crossed.
- (*Pending relaxation of the single `network_id` constraint*) Integrated fare systems where one fare is paid for journeys spanning multiple agencies.

In addition, there are many cases where `fare_transfer_rules.txt` is either insufficient or inefficient in modeling fares:

- Insufficiency:
    - As long as `nonconsecutive_transfers_allowed` is not adopted, multiple transfers between legs with different fare amounts will always have the potential to result in double-counting.
- Inefficiency:
    - Fare structures with a large number of zones (even as low as 8), will result in a huge number of zone combinations that is neither easy to implement nor maintain.

→ Fare Leg Join Rules has the capability to model transfer rules to an extent, and to make them very simple. Therefore, we need to include at least `duration_limit` and `duration_limit_type` to be able to apply transfer time limits on leg joins.

II. When to use Fare Leg Join Rules vs Fare Transfer Rules

Use Fare Leg Join Rules for:

- When legs are treated as one leg in the real world. E.g.: No transfer validation needed in the transfer station, silent transfer in the same bus.
- (*Pending relaxation of the single `network_id` constraint*) With integrated or multi-agency fare systems where the final fare depends on predicates encountered during the whole journey, such as the agency with highest fare (e,g, Sound Transit), the first/last agency (e.g. PRESTO), the first and last stop/zone (e.g. MTA LIRR). → *Add this to Network Sets.*
- In some cases where it simplifies transfer rules. E.g. In an origin-destination fare structure with a huge number of zone combinations, it is easier to join the journey legs and look at the first and last stops of the whole journey, instead of pricing leg-by-leg and applying a plethora of transfer rules.

Use Fare Transfer Rules for:

- Simple fare structures where the main factors to define the fare are the duration of the journey and the number of journey legs taken. E.g. TriMet.
- If your system has fare products that do not support any transfer, since Fare Leg Join Rules inherently assumes a transfer.
- (Rare case) If your system has fare products with different transfer restrictions for the same route networks (e.g. Monthly pass allowing theoretical transfers for a whole month vs a no-transfer pass), it is preferable to use `fare_transfer_rules.txt`, as using `fare_leg_join_rules.txt` will join legs by default, which will show the no-transfer fare products as if they support multi-leg journeys.
    - You can mitigate this by not specifying `network_id` for legs of fare products that do not support transfers.

- If your fare system is too complex to just use `fare_transfer_rules.txt`, you have to consider a trade-off between `fare_transfer_rules.txt` and `fare_leg_join_rules.txt`.

III. Issues of overlap, mismatch and priority

1) Fare Leg Join Rules and Fare Transfer Rules defined jointly in the same feed

An overlap might occur in certain cases.

a) Transfer rule defined for legs that should be already joined

This happens if:

- `network_a` is joined to itself in `fare_leg_join_rules.txt`, then the feed also defines a transfer from `network_a` legs to `network_a` legs
- Or, `network_a` is joined to itself in `fare_leg_join_rules.txt` with `from_stop_id` and `to_stop_id` specified, then the feed also defines a transfer rule from `network_a` legs to `network_a` legs for the zones in which `from_stop_id` and `to_stop_id` exist respectively.

`fare_leg_join_rules.txt`

from_network_id	to_network_id	duration_limit	duration_limit_type
network_a	network_a	3600	1

`fare_transfer_rules.txt`

from_leg_rule_id	to_leg_rule_id	duration_limit	duration_limit_type
leg_network_a	leg_network_a	7200	1

In that case, we either:

- **Forbid this. Forbid only with `network_id` to avoid complicated interactions and edge cases with `from_stop_id` and `to_stop_id`.**
- **Make sure the `duration_limit` in `fare_leg_join_rules.txt` takes precedence over anything in `fare_transfer_rules.txt`.**

b) Legs joined, then transfer rule defined with another network

This happens if:

- `network_a` is joined to itself in `fare_leg_join_rules.txt`, then a transfer rule is defined from `network_a` legs to a leg of another network.
    - **No problem here**. The join happens then the transfer rule will apply between effective legs and other legs.

`fare_leg_join_rules.txt`

from_network_id	to_network_id	duration_limit	duration_limit_type
network_a	network_a	3600	1

`fare_transfer_rules.txt`

from_leg_rule_id	to_leg_rule_id	duration_limit	duration_limit_type
leg_network_a	leg_network_b	7200	1

- (*Pending relaxation of the single `network_id` constraint*) If `network_a` is joined to `network_b` in `fare_leg_join_rules.txt`, then a transfer rule is defined from `network_a` legs to a `network_b` legs.
    - **No problem here**, because the effective leg will not be matched through `network_id`, more through the upcoming `travels_through_network_set_id`.

`fare_leg_join_rules.txt`

from_network_id	to_network_id	duration_limit	duration_limit_type
network_a	network_b	3600	1

`fare_transfer_rules.txt`

from_leg_rule_id	to_leg_rule_id	duration_limit	duration_limit_type
leg_network_a	leg_network_b	7200	1

- (*Pending relaxation of the single `network_id` constraint*) If `network_a` is joined to `network_b` in `fare_leg_join_rules.txt`, then a transfer rule is defined from `network_a` legs to a `network_a` legs.
    - **No problem here**, because the effective leg will not be matched through `network_id`, more through the upcoming `travels_through_network_set_id`.

`fare_leg_join_rules.txt`

from_network_id	to_network_id	duration_limit	duration_limit_type
network_a	network_b	3600	1

`fare_transfer_rules.txt`

from_leg_rule_id	to_leg_rule_id	duration_limit	duration_limit_type
leg_network_a	leg_network_a	7200	1

2) Fare products that use join rules + fare products without join rules

What if a fare structure has different fare products, some that have transfers, and some that do not? How do we use Fare Leg Join Rules?

- Should the consumer look for all possible combinations?
- Should we specify in the spec if multiple combinations can be explored or not?

→ **Currently, it seems like once `fare_leg_join_rules.txt` is defined, the join will happen upstream, before any matching is done. Therefore, other options will not be considered. It might remain up to the consumer to decide if they want to look for fare options with join and without join.**

IV. Examples

1) MTA Long Island Rail Road (LIRR):

PS: We will disregard the time-based fares of LIRR.
There are 8 zones for the LIRR, with same-way transfers possible. The final fare is the fare between the first zone and last zone of the journey.

The conventional approach would be:

`fare_leg_rules.txt`

leg_rule_id	from_area_id	to_area_id	network_id	fare_product_id
leg_1_to_1	1	1	lirr	fare_1_to_1
leg_1_to_2	1	2	lirr	fare_1_to_2
…	…	…	…	…

`fare_transfer_rules.txt`

from_leg_rule_id	to_leg_rule_id	duration_limit	duration_limit_type
leg_1_to_1	leg_1_to_1	…	…
leg_1_to_1	leg_1_to_2	…	…
…	…	…	…

However, the 8 zones will result in 8*8 = 64 leg rules, which will result in at least 64*8 transfer rules. Which come with their own fare products to make up the fare upgrade from leg to leg.

This can be replaced with:
`fare_leg_join_rules.txt`

from_network_id_id	to_network_id	duration_limit	duration_limit_type
lirr	lirr	…	…

One single row that's easier to maintain and removes the need for transfer fare products.

2) Other examples can be found in this document
   [GTFS Fares-v2] Network sets real-world case research
   These examples will be valid pending relaxation of the single `network_id` constraint and the addition of Network Sets, which rely heavily on the addition of `duration_limit` and `duration_limit_type`.