Bus Tracing

You

January 24, 2018

Abstract

Your abstract.

1 Problem Statement

Given a system that crawls bus arrival data from the Singapore LTA public API (aka DataMall), we would like to reconstruct the complete trace (list of [timestamp,location] tuples) for each bus. The problem is that, while the API provides bus data containing timestamp and location fields, there is no 'Bus ID' field provided. So it is not possible to directly recover individual traces of multiple buses on a single route. This is illustrated in Fig.1. However, if we can separate out this data into *snapshots* of the bus positions at different times, then we can infer the individual traces as shown in Fig.2.

However, the separation into snapshots is not easy for the following reasons:

- Bus observations are reported by each bus stop and the crawler collects data sequentially from the bus stops. So we don't have all bus observations for a given timestamp. This is easily overcome by using the crawler *downloadID* to separate out snapshots.
- Each bus stop reports only the next 3 arriving buses for each service number, and sometimes the next bus could be the same for two or more bus stops. This means we can have multiple observations of the same bus. Moreover, since these bus stops are crawled at slightly different times, the bus positions for these *duplicate* observations may be slightly different. This is illustrated in Fig.3

2 Solution Approach

We break the problem of extracting individual bus traces into two parts:

1. *De-duplication* of the bus observations in a single snapshot
2. *Stitching* snapshots together to get a complete trace

3 De-duplication within snapshot

We present three different ways to de-duplicate the bus observations in a single snapshot. We name them as follows:

- Parametric Distance based algo
- Non-parametric Distance based algo
- Non-parametric ETA (estimated time of arrival) based algo

In all three algorithms, the de-duplication groups duplicate observations together by assigning them the same local *Bus_ID*. We then choose one observation from each group and discard the rest. NOTE: Distance is the distance traversed along the route from the starting point.

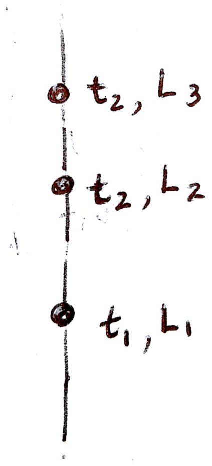

Figure 1: Bus observations with timestamp (t) and Location (L) but no 'Bus ID'.

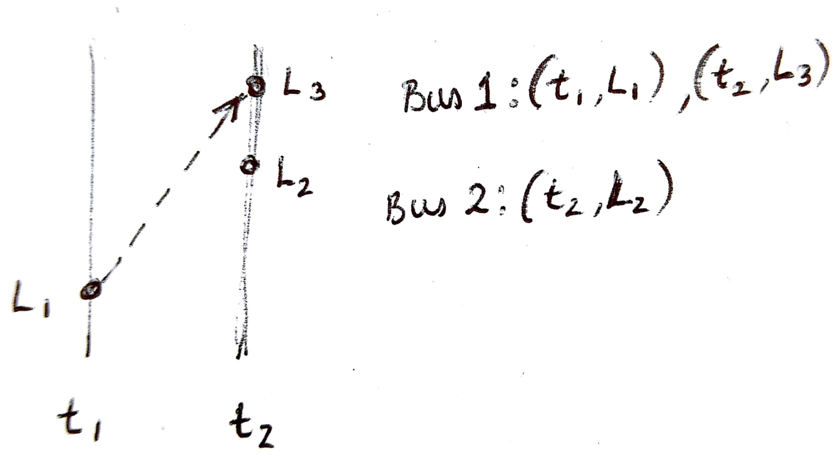

Figure 2: Observations from Fig.1 separated out as *snapshots* in time.

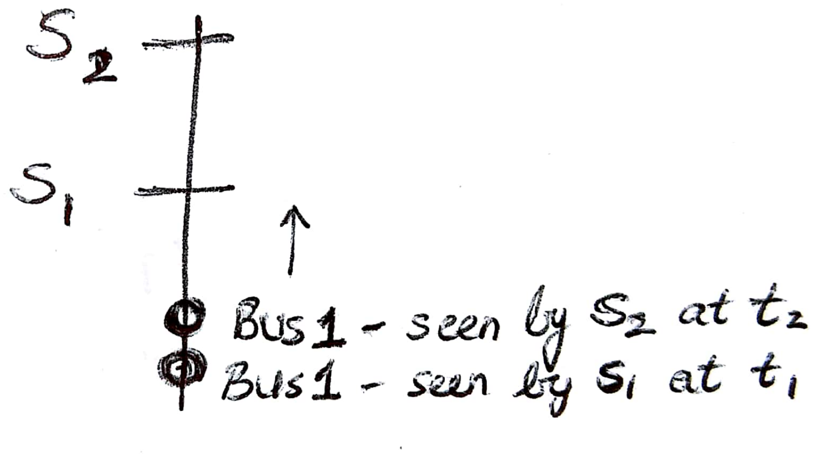

Figure 3: Two observations of the same bus collected in a single crawl through bus stops S_1 and S_2.

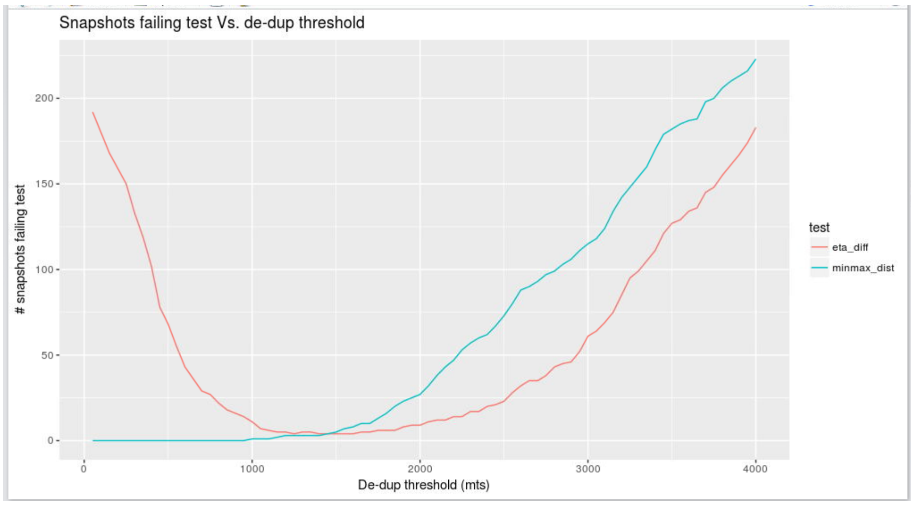

Figure 4: We set ϵ to the value that gives the lowest count of snapshots failing the ETA criterion (red curve). A similar curve using a distance-based test for errors is show in cyan.

3.1 Parametric Distance based algo

This algorithm simply groups together observations that are within ϵ distance of each other. However, too small a value of ϵ may result in duplicate observations being treated as distinct, and too large a value of ϵ will result in distinct observations being treated as duplicates. We then use the ETA field to check if the de-duplication is incorrect. Usually the ETA of a given bus at one bus stops will differ from the ETA at an adjacent stop by not more than 10min. We then count the number of snapshots that fail this criterion for a given value of ϵ. We repeat this over a range of ϵ values and use the value which gives the lowest number of failing snapshots as the threshold for grouping observations together.

This approach works but takes longer time to run, because we run the de-duplication repeatedly for a large range of ϵ values.

3.2 Non-parametric Distance based algo

If we plot distance vs. bus stops (the bus stop which reported an observation), for all observations of a single bus service for a single day we get a pattern like in Fig.5. Next bus is color-coded as red, SubsequentBus as green, and SubsequentBus3 as blue.

It is clear just by looking at this plot, that each horizontal line corresponds to duplicate observations (from different bus stops) of a single bus. The non-parametric distance algorithm assigns a Bus_ID to each observation by matching observations from one bus stop with observations from the next bus stop. If the matching results in a red-to-red, gree-to-green, blue-to-blue match, then it means that the three buses at a given bus stop are same as those at the next bus stop. However, if the matching results in a red-to-green, green-to-blue match then it means that ONE new bus has been added, and bus_IDs are assigned accordingly. This algorithm is called *non-parametric* because there is no parameter to tune like in the Parametric Distance based algo. The details of the matching is captured in Algorithm 1.

3.3 Non-parametric ETA based algo

This is exactly the same as the distance based algo except that it uses ETA values instead of distance. This should be clear after looking at Fig.6

3.4 Comparison of Approaches

We compare the percentage of snapshots passing two tests for each of the algorithms described above for a single service on a single day. The ETA test is the same as described in the section for parametric distance based algo. The threshold used for the ETA test is 10min. On the other hand, the distance test checks if the min. and max. distances of duplicate observations are within

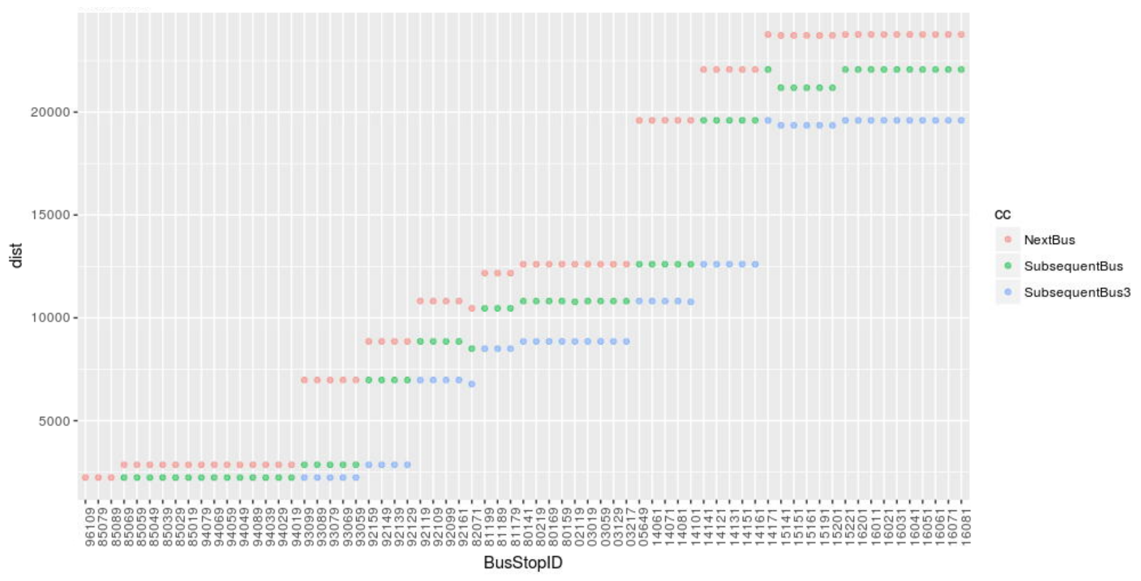

Figure 5: Plot of distance vs. Bus stop for a single snapshot.

Algorithm 1 Assign bus IDs to observations at stop S_j given bus IDs for observations at stop S_i

```
1: $X_i \leftarrow$ vector of bus distances observed at stop $S_i$ sorted desc.
2: $X_j \leftarrow$ vector of bus distances observed at stop $S_j$ sorted desc.
3: $I_i \leftarrow$ vector of IDs for buses observed at stop $S_i$
4: $cost\_rr\_gg\_bb = ((X_i[1] - X_j[1]) + (X_j[2] - X_j[2]) + (X_j[3] - X_j[3]))/3$
5: $cost\_rg\_gb = ((X_i[1] - X_j[2]) + (X_j[2] - X_j[3]) + (X_j[3] - X_j[3]))/2$
6: $cost\_rb = (X_i[1] - X_j[3])$
7: if $cost\_rr\_gg\_bb$ is least then
8:     $I_j = I_i$
9: else if $cost\_rg\_gb$ is least then
10:     $I_j = I_i + 1$
11: else if $cost\_rg\_gb$ is least then
12:     $I_j = I_i + 2$
13: end if
```

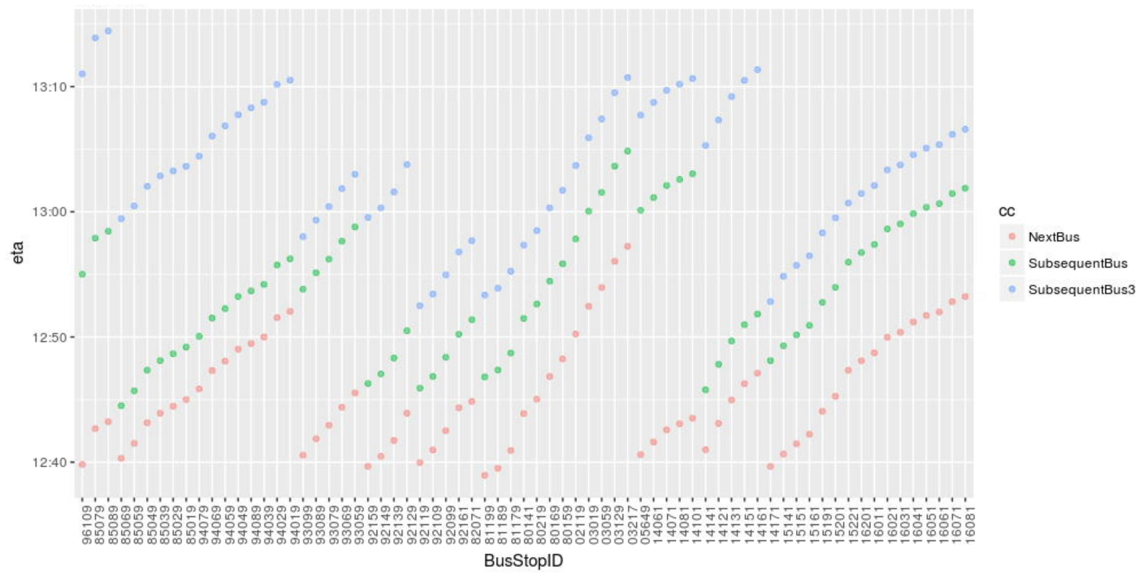

Figure 6: Plot of ETA vs. Bus stop for a single snapshot.

Table 1: Comparison of the three de-duplication algorithms.

	% Snapshots passing test	
Algorithm	Distance test	ETA test
Parametric (d=1450 mts)	98.7	98.7
Non-parametric Distance-based	96	96
Non-parametric ETA-based	95.7	95

Table 2: Performance of de-duplication algorithm for Service nos. 10, 14 & 16 over a 1-week period.

	% Snapshots passing test over 1 week	
Service Nos.	Distance test	ETA test
10	96.97	96.07
14	97.88	96.02
16	96.92	93.15

some X mts of each other. In our case we set X as ˜ 3km. For both these tests it is difficult to set a threshold which will catch all errors, but yet leave out the correct ones. So these tests are conservative in that they only catch gross errors in de-duplication. The results are shown in Table 1.

Subsequently we decided to use the Non-parametric distance-based algo as it was much faster than the parametric distance-based algo. We then evaluated the performance of this algo on multiple services over a 1-week period. The results for Service nos. 10,14 & 16 (Fig. 7) are shown in Table 2 and the results for Service nos. 131, 139 & 64 (Fig. 8) are shown in Table 3.

4 Stitching snapshots

4.1 Algorithmic approach

We can solve the snapshot stitching problem using the following simple algorithm which stitches two snapshots X_t and X_{t+1}. This algorithm is then applied iteratively over successive pairs of snapshots to get the complete trace.

The speed test in the above pseudo-code computes the speed of each bus given a matching, and if any bus speed is greater than some threshold (in our case we set it to 40kmph) then we clear that matching, skip the current snapshot and repeat the matching process.

This is illustrated in Fig.9

4.2 Optimization Approach

Let X_t be the set of buses observed in snapshot t (an individual observation can be identified as X_t^i) and let X_{t+1} be the set of buses observed in snapshot $t+1$. Let $E = (u, v)$ be the set of all edges with one end-point $u \in X_t$ and another $v \in X_{t+1}$. Then we can define a matching $M \subseteq E$ such that every observation in X_t and X_{t+1} is incident to at most one edge in M. For each matched pair of observations we can compute a cost of matching as the distance between the two matched observations i.e. $c_{ij} = d(j) - d(i)$. Unmatched observations in X_t are implicitly matched to the ending terminus and unmatched observations in X_{t+1} are matched to the starting terminus. Then

Table 3: Performance of de-duplication algorithm for Service nos. 131, 139 & 64 over a 1-week period.

	% Snapshots passing test over 1 week	
Service Nos.	Distance test	ETA test
131	98.78	95.27
139	98.73	96.6
64	41.19	41.35

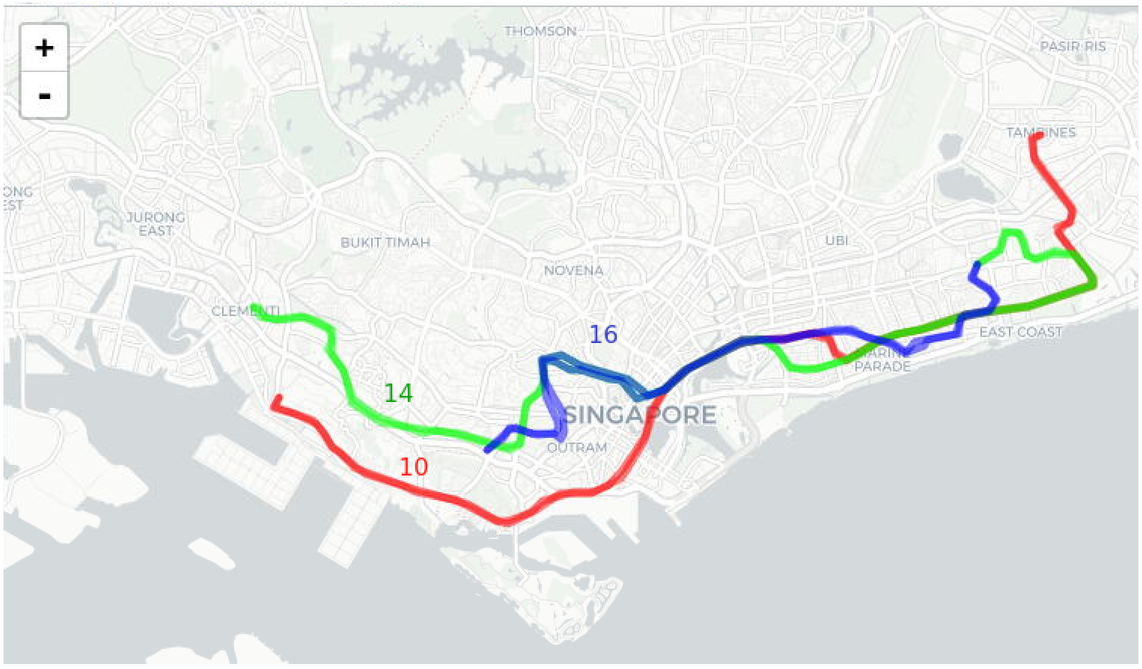

Figure 7: Routes 10,14 & 16.

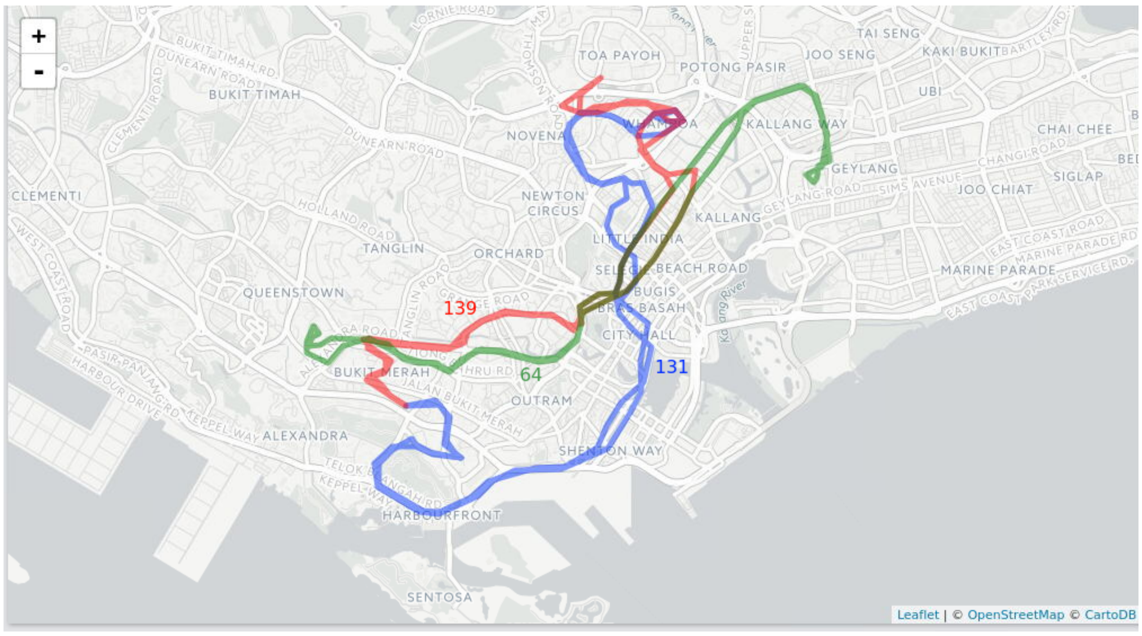

Figure 8: Routes 131,139 & 64.

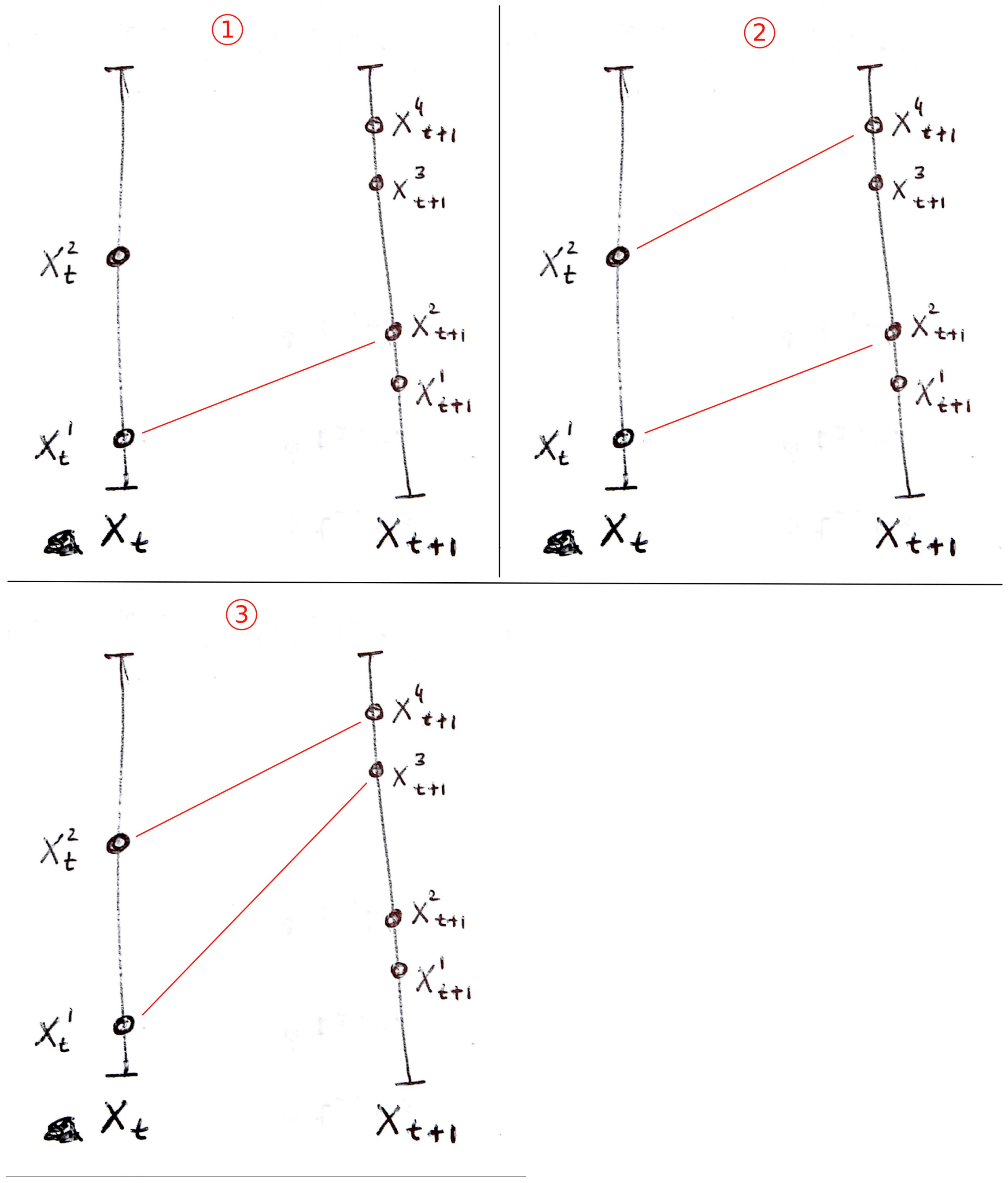

Figure 9: Stitching of two snapshots. In step 2, there is a single unmatched observation X_{t+1}^3, which is fixed by shifting all previous matchings up by one.

Algorithm 2 Assign bus IDs to observations in snapshot SS_j given bus IDs for observations in snapshot SS_i

```
 1: $i = 1$
 2: $j = 2$
 3: while j <= no. of snapshots do
 4:   $X_i \leftarrow i^{th}$ snapshot
 5:   $X_j \leftarrow j^{th}$ snapshot
 6:   for $k = 1$ to no. of observations in $X_i$ do
 7:     $C \leftarrow$ set of observations from $X_j$ between $(X_i^k, X_i^{k+1})$
 8:     if $C$ is not empty then
 9:       match $X_i^k$ to observation with largest distance, say $X_j^l$, in $C$
10:       move up RHS of all prev matchings by $|C| - 1$
11:       if any unmatched observations left below $X_j^l$, assign new bus IDs to them
12:     else
13:       // $C$ is empty
14:       match $X_i^k$ to observation from $X_j$ above and nearest to $X_i^{k+1}$
15:     end if
16:   end for
17:   if above matching violates speed check then
18:     // skip current and try to match to next snapshot
19:     clear matching and $j \leftarrow j + 1$
20:   else
21:     save matching
22:     $i \leftarrow j$
23:     $j \leftarrow i + 1$
24:   end if
25: end while
```

the total cost of matching M is :

$$C(M) = \sum_{(i,j)\in M} d(j) - d(i) + \sum_{i\in X_t - M} d_{end} - d(i) + \sum_{i \in X_{t+1} - M} d(i) - d_{start}$$

where $d_{start} = 0$ and $d_{end} = L$, the length of the route.

The goal is to find a matching M that minimizes the above cost, subject to the following constraints:
No reversing: for every $(i, j) \in M,\ d(j) >= d(i)$
No crossover: for every pair (i_1, j_1) and $(i_2, j_2) \in M$, if $d(i_1) > d(i_2)$ then $d(j_1) > d(j_2)$
Unmatched observations in X_t: $\min_{i\in X_t - M} d(i) > \max_{j \in M\cap X_t} d(j)$
Unmatched observations in $X_t + 1$: $\max_{i\in X_{t+1}-M} d(i) < \min_{j\in M\cap X_{t+1}} d(j)$

4.2.1 Optional - Including bus speed

The above approach does not account for bus speed, and it will prefer a matching where bus speed is slower, even if there is another matching satisfying the constraints but requiring a higher bus speed. For example, in Fig. 11 the 0 in-0 out matching will be picked. However, if the time interval between two snapshots is large then the 1 in-1 out matching might be more realistic.

We can account for this by adding a $(v_{bus} - v_{avg})^2$ term that will give high cost to matchings where bus speed required is much lower or higher than the average.

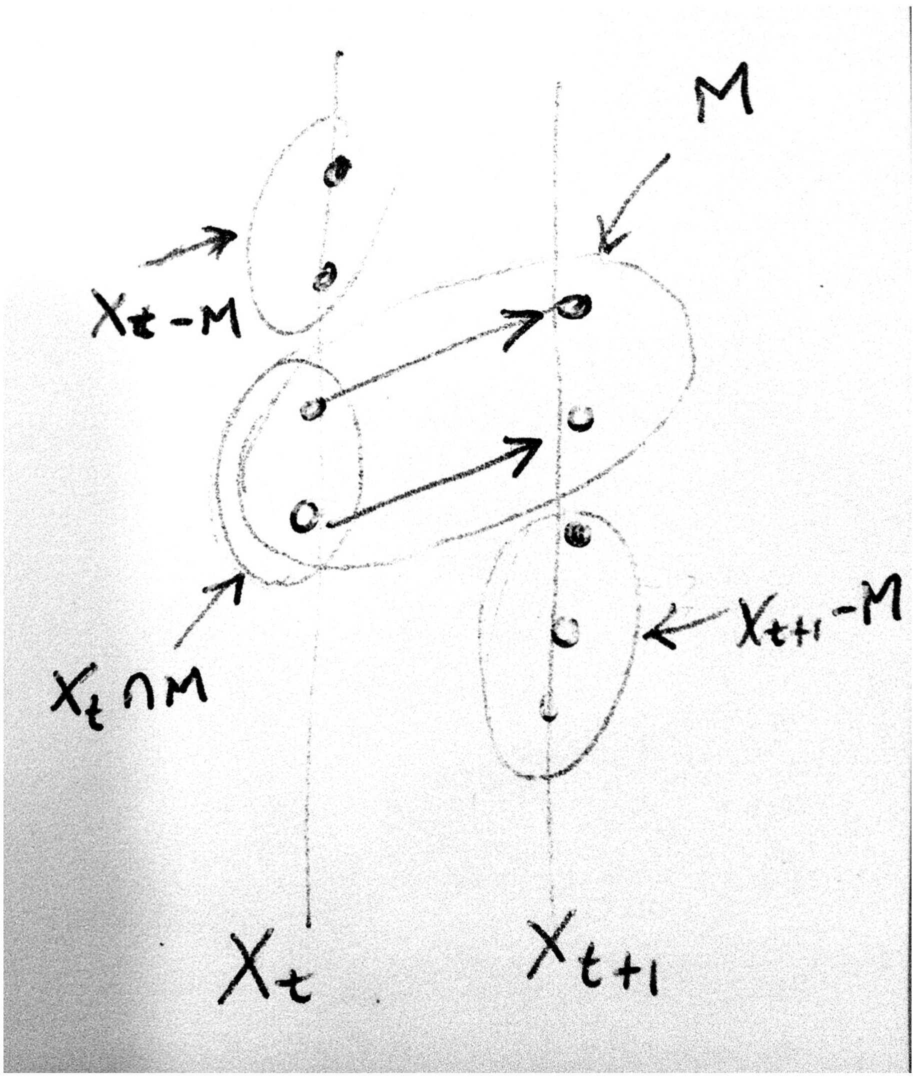

Figure 10: Illustration of various sets in the optimization formulation.

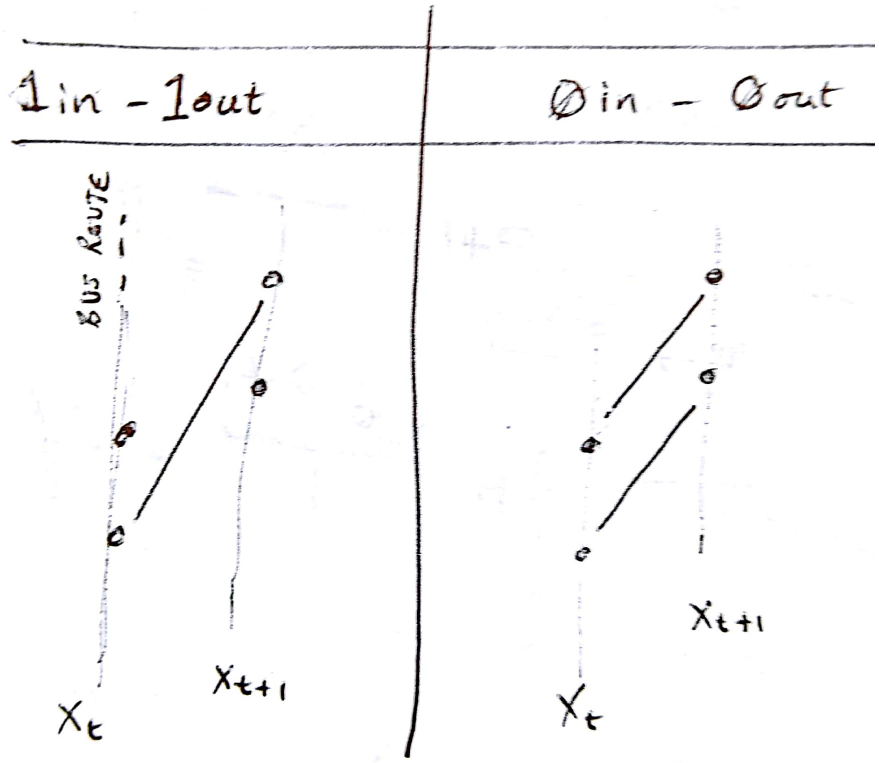

Figure 11: 1 in-1 out vs. 0 in-0 out scenario.