1 Transaction Generator

First of all it is important to remark that we are first generating ordinary, regular transactions that are not fraud kinds of transactions. Therefore the transactions generated for each card object have to be generated in such a way that they will not produce any fraud pattern alert. Later in the process we will poison our system by generating those transactions to produce fraud pattern alerts.

1.1 Regular transactions

Considerations:

- Particular ATM usage frequency
- ATM selection for the generated transactions of each card
- Distribution of the genetated transactions on the decided considered time

Some ideas to explore:

- ATM - Pre-selection of closed card ATM subset.
- Time - Uniform distribution.
- Time - Poisson process distribution.
- Random walks.

1.1.1 ATM closed subset + Uniform time distribution

- → ATM selection: closed ATM subset.
- → Time distribution: uniform distribution of the *num_tx* transactions for a card on a certain day (*num_tx* is drawn from a Poisson distribution of $\lambda =$ `withdrawal_day`, based on the behavior of the *Wisabi* clients).

More detailed explanation follows.

The transaction generator is done to be able to generate transactions for each of the cards based on the gathered client transaction behavior of each of the cards for a customisable `d` number of days starting in a `start_date`.

For a card, the idea is to create a certain number of transactions per day, by linking the card to a certain ATM that is no farther than `max_distance` kms from the residence location of the client of the card. Also, we will limit the time distance between two consecutive transactions so that the final set of created transactions can not produce a potential fraud related with having two transactions in different ATM locations with an insufficient feasible time distance.

For each card:

- **ATM subset**: Create a subset of ATMs that are considered to be *usual* for the card client, so that they are all at a distance inferior or equal to `max_distance` kms to the residence location of the client of the card. We limit the size of this subset to be of `max_size_atm_subset`, so that we take only a maximum of `max_size_atm_subset` of the closest ATMs. In addition, among the ATMs in this subset, preference is given to those that are the closest to the residence location of the client and those that belong to the same bank company as the client's card. The transactions generated for this card will be linked only to ATMs of this subset called `Neighborhood`.

$$\texttt{Neighborhood} = \{\texttt{ATM}\ |\texttt{dist(ATM, residence\_loc)} \leq \texttt{max\_distance}\}$$

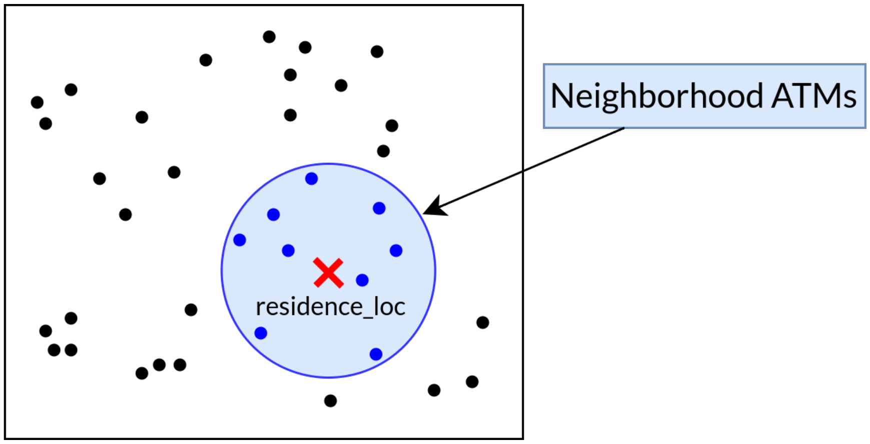

Figure 1: `Neighborhood` ATM subset

- `t_min`: Minimum threshold time between any two consecutive transactions of the client. That is, the minimum time distance between the end of a transaction and the start of the next consecutive transaction of a card.

$$t_{min} = \frac{2 * \texttt{max\_distance}}{\texttt{max\_speed}}$$

  For the calculation of this minimum time distance:

  - `max_distance`: two options:
    - In general: taking $2 * \texttt{max\_distance}$ kms as the upper bound on the maximum distance between 2 ATMs of the ATM subset, set the `t_min` to be the time needed to traverse that distance at a selected `max_speed`. (* So far done like this).

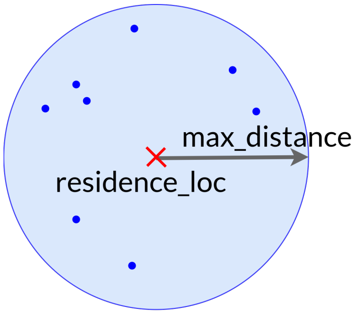

Figure 2: max_distance upper bound for the calculation of t_min

    - Specifically: Taking the specific maximum distance between all the ATM pairs in the ATM subset to get the t_min.
  - max_speed: maximum speed at which it is possible to travel (by any possible means of transport) between any pair of ATMs $\in$ Neighborhood.

- For each day generate num_tx transactions, random number drawn from a Poisson distribution of $\lambda =$ withdrawal_day.

- Distribution of the num_tx transaction times (transaction_start), doing a uniform distribution of the num_tx for each of the days:

  We create an ordered list of num_tx start moments in seconds in a day (in the range $[\texttt{t\_min}/2, 86400 - (\texttt{t\_min}/2) - \texttt{max\_duration}]$) so that all of them are at a minimum time distance of $\texttt{t\_min} + \texttt{max\_duration}$. See Figure 3. *max_duration limits the maximum duration of an ordinary transaction. For the moment it was set to be of 10 minutes (600s).*
  Note that the interval bounds for the transaction start moments are designed in such a way that the t_min minimum time distance between the end of a transaction and the start of the next transaction is also respected between transactions belonging to different consecutive days. See Figure 4. Note that this way of generation (day by day) we are not allowing transactions to occur in the interval marked among the two red dotted lines of the Figure 4. NOTE/TODO: This could be fixed by looking to the start time of the last transaction of the previous day.... The start moments are therefore taken to define the transaction_start of each of the transactions of that day. transaction_end is assigned a shifted time difference with the respective transaction_start, in particular the difference is drawn from a normal distribution $\mathcal{N}(300, 120)$ that defines the duration of a transaction to be of mean of 5 minutes (300s) and a standard deviation of 2 minutes (120s), bounding it to be of a maximum time of max_duration of 10 minutes (600s) and setting it to the mean if the distribution sample was negative.

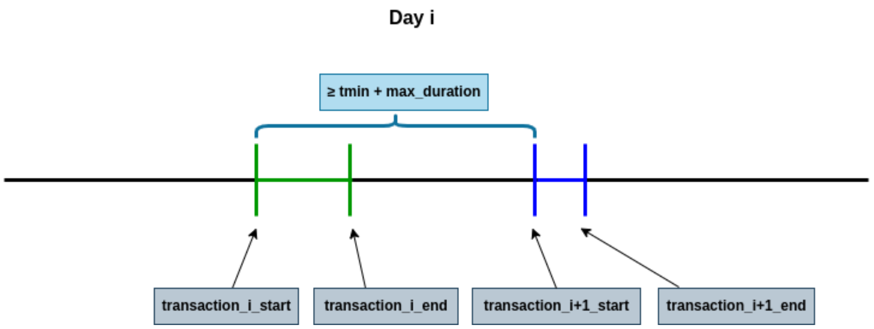

Figure 3: Time distance limit between two consecutive transactions

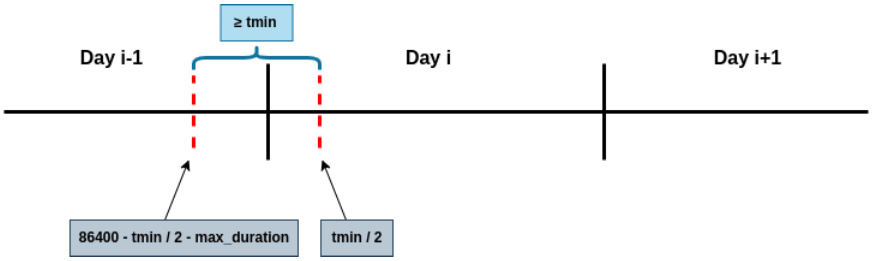

Figure 4: Time distance limit between two consecutive transactions between two days

Note that a limit was set on the duration of the transaction so that we can have the control avoiding time overlapping transactions, which will be producing irregular undesired fraud pattern alerts.

- `transaction_amount`: based on card behavior parameters, it is drawn from a normal distribution $\mathcal{N}(\texttt{amount\_avg}, \texttt{amount\_std})$. If negative amount, drawn from a uniform distribution $\mathcal{U}(0, \texttt{amount\_avg} * 2)$.

Note that this approach is done with the focus on avoiding the production of potential fraud scenarios. By taking into account the previous generated transaction: both for the linked ATM of the new transaction and its transaction time (to avoid transactions that are overlapped or that come one directly after the other, since this may be fraudulent!).

1.1.2 Poisson Process

1.1.3 Random Walk