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

- **By card**: Generation of the transactions for each of the cards independently. **We have control** to avoid anomalous scenarios when selecting the ATMs and distributing the transactions along time.
- **In general**: Linking ATM and client composing (card,ATM) pairs, and distributing these pairs along time according to a certain distribution. **No control / More difficult to control the possible derived anomalous scenarios produced among same card pairs**.

Therefore, the generation by card option it is considered to be the best so to be able to have the control on the possible anomalous scenarios for the generated transactions of each of the cards. Some ideas to explore:

- Selection of ATMs:
  - Neighborhood / Closed ATM subset.
  - Random walk. To do the selection of the sequence of ATMs for the generated transactions.
- Distribution of the transactions along time:
  - Uniform distribution.
  - Poisson process distribution.
- Other options:
  - Random walk for both the ATM and the transaction time selection, in the same algorithm together.

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`.

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

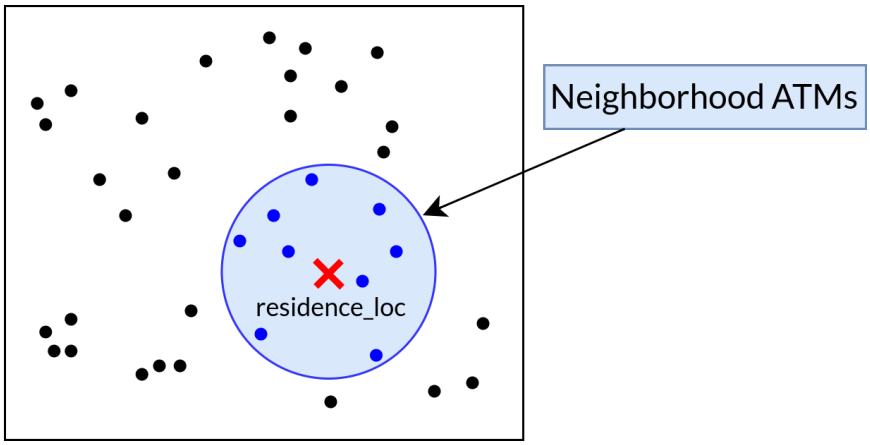

Figure 1: `Neighborhood` ATM subset

- `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).

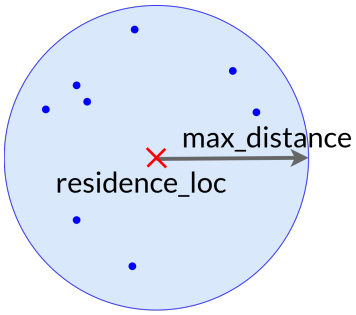

Figure 2: `max_distance` upper bound for the calculation of `t_min`

    - 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 ∈ `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 **??**. ***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 **??**. 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 **??**. 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.

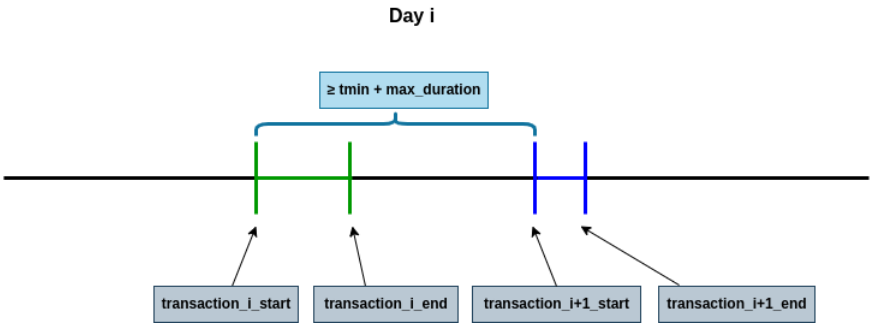

Figure 3: Time distance limit between two consecutive transactions

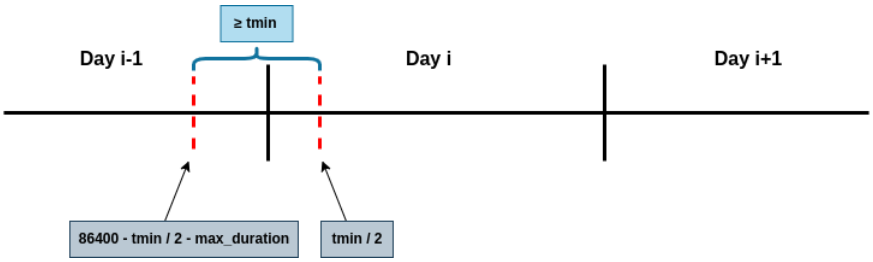

Figure 4: Time distance limit between two consecutive transactions between two days

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 $\mathcal{N}$(`amount_avg`, `amount_std`). If negative amount, drawn from a uniform distribution $\mathcal{U}(0,$ `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 ATM closed subset + Poisson Process

→ **ATM selection**: Closed ATM subset.

→ **Time distribution**: Poisson process distribution of num_tx transactions for each of the cards.

Generate `num_tx` transactions for a selected period of time `t`. Distribution following a Poisson process distribution along $[0, t]$.

- $\lambda =$ `avg_tx` on a day for the client if `t = 24h`. Otherwise, decide a specific λ for the considered `t`.
- Inter-arrival times $X_1, X_2, \cdots$ are distributed following an exponential distribution: $X_i \sim \text{Exp}(\lambda)$. They represent the time elapsed between two of these events, e.g. X_2 represents the time elapsed between the first and the second arrival. Note that in this case, since we need to respect the minimum required time between two consecutive transactions (t_{min}) so to avoid introducing anomalous undesired scenarios, we have to impose that: $X_i \geq \Delta_{i-1} + t_{min}, \forall X_i$, where:

- X_i: time elapsed between the i-1 and i-th transaction.
- Δ_{i-1}: time duration of the i-1 transaction. This duration will be upper bounded by Δ_{max}, which will be considered the maximum possible duration of a transaction.
- t_{min}: minimum calculated time distance between any 2 consecutive transactions of the client.

With these interarrival times, we obtain the arrival times T_i. Note that they are not independent, in particular: $T_1 \leq T_2 \leq T_3 \cdots$.

Therefore, to generate the Poisson process with rate λ:

1. Generate i.i.d. random variables $X_1, X_2, \cdots$ where $X_i \sim \text{Exp}(\lambda)$.
2. Obtain the arrival times as:
   - $T_1 = X_1$
   - $T_2 = X_1 + X_2$
   - $T_3 = X_1 + X_2 + X_3$
   - $\cdots$

Note that, having imposed the previous we will have that:

$$\begin{cases} T_i = T_{i-1} + X_i \\ X_i \geq \Delta_{max} + t_{min} \end{cases} \quad \forall X_i \tag{1}$$

which implies that:

$$T_i \geq T_{i-1} + \Delta_{max} + t_{min}, \forall X_i \tag{2}$$

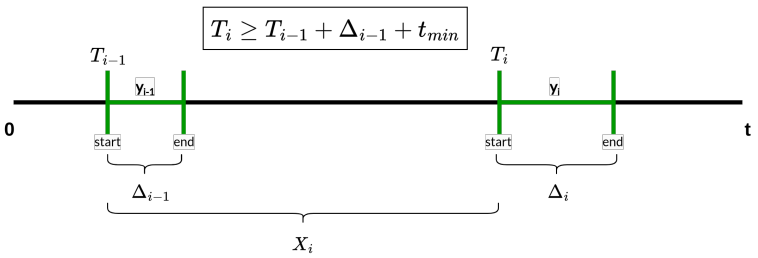

Figure 5: Schema of the interarrival and arrival event times on the poisson process.

References:

- Poisson process modeling - Theoric description

- Poisson process modeling - Python implementation
- Less formal explanation with Python code
- More theory
- Wikipedia entry

1.1.3 Random Walk / Markov Chain

Idea is to model the sequence of transactions for each of the clients as a markov chain through the network of ATMs. The network is a fully-connected graph with ATMs as nodes and edges are weighted with the distance between each pair of ATMs as weights.

The idea is to obtain the sequence of transactions for each client as a markov chain, in which, for each step a transaction is generated in that specific ATM node, at a certain datetime respecting the constraint of the minimum time distance with the previous transaction, so that no undesired anomalous fraud scenarios are produced.

Some considerations:

- Initially, we will compute the transition matrix obtaining all the respective transition probabilities. The probability of transition to another ATM-node will be inversely proportional to the distance to the considered ATM-node.
- Transition matrix P_t: containing, for each state, the probability of transitioning between states. For example, the entry i, j: $(P_t)_{i,j} = \mathcal{P}(X_{t+1} = j | X_t = i)$ contains the probability of transitioning to state j when being in state i. Note that since we assume a complete graph, and also the posibility to transition to the same state, all entries of this matrix will be distinct to 0: $(P_t)_{i,j} \neq 0 \ \forall i, j$.
- Finally, once P_t is computed, we perform the simulation to obtain the sequence of transactions for the specific card/client.

References:

- Markov Chains
  - MC - Theory
  - Simulation of Markov Chains - Theory
  - MC - Implementation example
  - MC - Implementation video example
- Random Walks
  - Theory:

    - RWs: A Review of Algorithms and Applications
    - RWs on Graphs
    - RW - More theory
- Implementation examples:
    - Simple RW on a Graph - Implementation example
    - A RW on a graph - Implementation
    - Simulate Random Walks Through Markov Chain - Matlab Implementation example