RouteAligner Project Report

Leonardo Giammarella

1 INTRODUCTION

Transportation problems have become increasingly prevalent in contemporary scenarios, especially within logistics companies responsible for the efficient movement of goods between various locations. As companies strive to optimize their transportation processes, challenges arise in maintaining adherence to planned routes, resulting in deviations by truck drivers.

In response to these challenges, organizations often utilize specialized software to generate standard routes, each associated with a specific price. However, the reality is that drivers frequently deviate from these predetermined routes, leading to a notable difference between the assigned "standard routes" and the actual routes executed, often influenced by drivers' personal choices.

Efficient route planning is essential for resource management. By providing accurate route recommendations and ideal routes, the system aids in allocating resources more effectively, such as managing truck availability and merchandise inventory. Moreover, timely and reliable deliveries are fundamental to customer satisfaction. Ensuring that drivers adhere closely to assigned routes enhances delivery punctuality, contributing to customer trust and loyalty.

However, the processing of historical actual and standard routes involves dealing with vast and complex datasets, managing diverse types of merchandise, varying quantities, and city sequences, that add a layer of intricacy to the analysis. Additionally, the dynamic nature of routes i.e., routes evolving over time based on factors such as changes in customer demand, traffic conditions, and company policies, requires the system to adapt using robust and flexible algorithms.

Accurately recommending standard routes demands a deep understanding of driver behaviors and preferences. Striking a balance between recommending routes that are both cost-effective for the company and acceptable to the drivers is a non-trivial task.

To tackle these transportation-related challenges, a systematic approach is required. Developing solutions to align assigned routes with drivers' preferences has become a crucial focus. As transportation problems continue to evolve, there is a growing need for systems that can analyze historical data, provide route recommendations, prioritize standard routes for individual drivers, and generate ideal routes aimed at minimizing deviations.

The proposed approach to recommend new standard routes aims to refine existing standard routes to increase their average similarity to corresponding actual routes. The method involves examining each existing standard route, analyzing associated actual routes, and using frequent itemsets mining methods to generate a modified version, which replaces the original standard route. The approach balances company requirements and driver preferences by modifying existing routes instead of creating new ones. Additionally, the method enables parallel processing by categorizing actual routes into batches, enhancing overall optimization efficiency for extensive datasets. This method has demonstrated satisfactory effectiveness, particularly in scenarios where drivers deviate significantly from the assigned standard routes.

When identifying the five standard routes from which drivers are likely to deviate the least, first, a basic algorithm has been introduced. It recommends to each driver the five standard routes with the highest average similarity to the driver's historical implementations. This algorithm is then enhanced, adding a preliminary filtering step that uses minhashing for the estimation of Jaccard similarities. This step efficiently avoids the need to compute similarity for each actual route against its corresponding standard route. The filtering process selects only those standard routes that are likely to exhibit minimal deviation from the driver's historical routes. This approach excels in scenarios where drivers' actual routes show little deviation from the assigned standard routes.

Lastly, to generate ideal routes for each driver, a similar approach to updating the standard routes was employed. However, it did not prove to be very effective, possibly due to the dataset generation process used.

2 RELATED WORK AND TECHNOLOGIES

In this section, we review essential concepts and methodologies that have been used for this project and which we will refer to.

2.1 Jaccard similarity

The Jaccard similarity is a metric used to quantify the similarity between two sets. It is calculated as the size of the intersection divided by the size of the union of the two sets.

2.2 Jaccard similarity for multisets

A **multiset** is a mathematical concept that extends the idea of a set by allowing elements to appear more than once, unlike a traditional set where each element is distinct. In a multiset, elements are associated with a count, indicating how many times they appear. In this project, the set of merchandise items for each trip is modeled as a multiset, where the count represents the quantity of each item.

Given two multisets A and B, their Jaccard similarity is computed using the formula:

$$J(A,B) = \frac{\sum_{i=1}^{n} min(a_i, b_i)}{\sum_{i=1}^{n} max(a_i, b_i)}.$$

Here, a_i and b_i represent the counts of the i-th element in sets A and B after mapping each element to integers $1, \dots, n$. If an element is absent in one of the two sets, the corresponding count is considered zero.

As an example, for $A = \{1,1,1,2,2,3,4\}$ and $B = \{1,1,3,3,5\}$, the counts a_i and b_i are given in the following table:

Element	**1**	**2**	**3**	**4**	**5**
# in set A	3	2	1	1	0
# in set B	2	0	2	0	1

The Jaccard similarity is calculated as:

$$J(A,B) = \frac{2+1}{3+2+2+1+1} = \frac{3}{9} = \frac{1}{3}.$$

2.3 Edit distance

Given two strings, the edit distance between these two strings is defined as the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into the other.

2.4 Minhashing

Minhashing is a technique used for estimating the similarity between two sets, particularly in the context of large-scale data where direct comparisons can be computationally expensive. The primary goal is to quickly determine the degree of overlap or similarity between two sets without exhaustively comparing all elements.

In detail, each set is represented by a a signature or hash value produced using a hash function. The key insight is to use multiple hash functions to generate a set of signatures for each set. The resulting signatures are then compared to estimate the Jaccard similarity, a measure of the intersection over the union of the sets.

This method will be used in this project to quickly estimate the similarity of two routes.

3 PROBLEM STATEMENT

Our scenario involves drivers participating in the transportation of various goods between different cities using trucks. A truck moves between two cities while carrying one or more types of merchandise, each with specified quantities. Notably, we assume the quantity, expressed as an integer, may be expressed in terms of items or kilograms, omitting a specific unit. Such a transportation process between two cities is commonly referred to as a **trip**. For instance, consider the following example:

Trip: Legnano to Monza - (milk: 3, pens: 10, butter: 20)

A truck can embark on a sequence of trips, with the endpoint of one trip serving as the starting point for the next. This sequence of trips is known as a **route**.

An example of a route comprising three trips follows:

(1) Trip: Rome to Milan - (milk: 3, pens: 10, butter: 20)
(2) Trip: Milan to Verona - (milk: 5, honey: 9, butter: 10, tomatoes: 20)
(3) Trip: Verona to Venice - (butter: 7, pens: 2, tomatoes: 10)

This route commences in Rome and, through Milan and Verona, concludes in Venezia.

The company, based on the frequent orders it receives, generates suggested routes, i.e., sequences of cities with their corresponding merchandise types and quantities for each trip. These routes are termed **standard routes** (SR).

However, it is assumed that truck drivers in charge of executing routes often deviate from the planned route, meaning they do not strictly adhere to the standard route they were initially hired to follow. They might load slightly more or less of a particular merchandise, add extra merchandise, or omit some altogether. Moreover, they may introduce additional cities into their route or omit some cities. These routes, which reflect the deviations, are referred to as **actual routes** (AR). Drivers do this due to personal preferences or agendas.

This discrepancy is a situation that the company wishes to minimize. They aim to reduce the disparity between what they assign to drivers (the standard routes) and what the drivers actually execute (the actual routes).

In light of these challenges and objectives, the development of a system is required. This system will, based on the existing set of standard routes and the historical actual routes completed by the drivers, accomplish the three following tasks:

- **Task 1** Provide recommendations to the company regarding which standard routes they should adopt.
- **Task 2** For each driver, create an ordered list of standard routes. The higher a standard route is placed in the list, the less likely the driver is to deviate from it. These routes are chosen from the pool of standard routes originally provided by the company and those recommended.
- **Task 3** For each driver, generate an ideal standard route that minimizes deviations from the driver's actual routes.

3.1 Routes similarity

3.1.1 Similarity between a standard and an actual route. In this section a measure of similarity between a standard route and an actual route is defined. Considering a standard route S and an actual route A, we denote the set of cities visited in the standard route as C_S and in the actual route as C_A. We assume that in the standard route each city is visited only once.

The calculation of similarity between these two routes involves a weighted average. This average is determined by the Jaccard similarity of the sets of cities visited in the two routes, coupled with a measure that captures the similarity of merchandise items in trips with shared destinations.

In a more detailed breakdown:

(1) Compute the Jaccard similarity between the sets of cities visited in the two routes and denote this value as $J(C_S, C_A)$.
(2) For each destination city in the standard route:
    - If the current destination city is not a destination city of any trip of the actual route, assign a similarity score of 0 for the current standard trip.
    - If the current destination city is a destination city of some trip in the actual route, compare the sets of merchandise items in the two trips using Jaccard similarity for multisets. Note that there may be multiple trips in the actual route whose destination city matches the destination city of the current standard route trip (if, for example, some cities are visited multiple times in the actual route). In such cases, retain the highest Jaccard similarity score for merchandise items.
(3) Compute the average of the calculated Jaccard similarities for merchandise sets and denote the final value as $J_{merch}(S, A)$.
(4) Evaluate the similarity between the standard and actual routes as follows:

$$Sim(S, A) = w_1 \cdot J(C_S, C_A) + w_2 \cdot J_{merch}(S, A),$$

where $w_1, w_2 \geq 0, w_1 + w_2 = 1$. Notice that $0 \leq Sim(S, A) \leq 1$. The weighting coefficients w_1 and w_2 allow for a flexible adjustment of emphasis between city and merchandise similarities. Unless specified differently, the weights will be $w_1 = w_2 = 0.5$.

To formalize the measure $J_{merch}(S, A)$, we establish some notation. Let S represent the set of trips in the standard route. For every trip $t \in S$, denote $M(t)$ as the (multi)set of merchandise items associated with trip t. Furthermore, let S^* be the set of trips in the standard route for which there exists a corresponding trip in the actual route, ensuring

that the destination cities of t and $a(t)$ are identical. As stated before, for some standard route trips t, multiple actual route trips may share the same destination city as t. In such cases, we define $a(t)$ as the trip in the actual route whose set of merchandise items $M(a(t))$ maximizes the Jaccard similarity with the set of merchandise items of t. If there are multiple actual route trips satisfying this criterion, one trip is randomly selected. The random selection process guarantees a one-to-one assignment, establishing a mapping from each standard trip t in S^* to a single corresponding actual trip $a(t)$ and ensuring the uniqueness of the value of $J(M(t), M(a(t)))$, irrespective of the specific (possibly random) choice of $a(t)$.
With these notations,

$$J_{merch}(S, A) = \frac{1}{|S|} \sum_{t \in S^*} J\,(\,M(t), M(a(t))\,).$$

Example. Consider the following routes:
Standard route S :

(1) Trip: Rome to Milan - (milk: 3, pens: 10, butter: 20)
(2) Trip: Milan to Verona - (milk: 5, honey: 9, butter: 10, tomatoes: 20)
(3) Trip: Verona to Venice - (butter: 7, pens: 2, tomatoes: 10)

Actual route A :

(1) Trip: Rome to Bologna - (water: 3, pens: 8, butter: 15)
(2) Trip: Bologna to Milan - (milk: 5, honey: 9, butter: 10, tomatoes: 20)
(3) Trip: Milan to Venice - (butter: 7, pens: 2, tomatoes: 10)

In this case,

$$J(C_S, C_A) = \frac{3}{5} = 0.6$$

and

$$J_{merch} = \frac{1}{3}\left(\frac{3+10}{5+9+20+10+20} + \frac{7+2+10}{7+2+10}\right) = 0.4.$$

Therefore, by choosing for example $w_1 = w_2 = 0.5$, the similarity of these two routes is 0.5.

The decision to calculate the average Jaccard similarities for sets of merchandise items among trips sharing the same destination city aims to prioritize trips in the actual route that successfully match the intended merchandise delivery to the correct destination. This approach acknowledges instances where the starting city in the actual route may differ from the standard route.

Focusing on the example provided above, the driver correctly delivers the designated merchandise to Venice, demonstrating a satisfactory execution of the trip. Consequently, this trip is considered successful, even if the starting city does not align with the one specified in the standard route. On the contrary, when the driver consistently fails to deliver goods in Verona, the Jaccard similarity score for that standard trip is rightfully assigned as 0. This emphasizes the importance of focusing on successful merchandise deliveries to the correct destination, regardless of deviations in the starting city.

However, the choice of this similarity measure is questionable and an alternative could involve opting for a more general yet stringent measure of similarity. Specifically, one might consider a similarity measure akin to the previous one, but focused on computing the average Jaccard similarity (for multisets) of merchandise items exclusively over trips in the actual route that **precisely** match the trips in the standard route (i.e., those trips that share both the start city and the destination city.)

4 SOLUTION

4.1 Task 1: Standard routes recommendation

To solve Task 1, I propose an approach that involves refining the existing standard routes, aiming to enhance their similarity to the corresponding actual routes on average. This approach avoids the creation of an entirely new set of standard routes, offering a more efficient and targeted optimization.

To implement this strategy, for each existing standard route, the following steps are undertaken:

- Examine standard route i;
- Analyze all actual routes associated with standard route i;
- Utilize frequent itemsets mining to generate a modified version of standard route i;
- Replace the original standard route i with the updated one.

This method ensures a more targeted optimization by refining each standard route in consideration of its historical actual routes, promoting a better fit between planned and executed transportation paths.

Furthermore, this approach strikes a balance between the company's requirements and the drivers' preferences. By modifying existing routes rather than creating new ones from scratch, it acknowledges the importance of maintaining a connection with established practices.

Another rationale for adopting this approach lies in its efficiency. Examining actual routes implementing standard route i to infer improvements on standard route j (with $i \neq j$) is deemed impractical, assuming that standard routes inherently differ from each other.

Lastly, this method facilitates parallel processing by categorizing actual routes into batches, each corresponding to the implementation of a particular standard route. This proves particularly advantageous when dealing with extensive datasets of actual routes, enhancing the overall efficiency of the optimization process.

4.1.1 Algorithm to recommend new standard routes. As anticipated, to update standard route i, we select all the actual routes associated with standard route i and create an ordered frequency table of all the trips, retaining only those trips with a frequency above a predefined threshold. Each trip in the current batch of actual routes is assigned a number, representing the number of times it appears in the batch (normalized by the total number of actual routes under consideration, if expressed as a percentage). Higher-frequency trips are positioned higher in the table. An illustrative frequency table is provided below:

Trip	**Frequency (percentage)**
Carpi - Baggio	16 %
Baggio - Verona	12 %
Genoa - Baggio	10 %
Verona - Baggio	8 %
Verona - Bologna	5 %
Bari - Baggio	5 %
Trieste - Cuneo	4 %

Subsequently, a new sequence of trips is generated using the following method: at each step, the most frequent trip is added to the new sequence, ensuring that the last city of the last trip matches the start city of the next trip. If this continuity is not feasible, a dummy link trip is added. Additionally, each time a new trip is included, all remaining available trips containing already-visited cities are removed.

Algorithm 1: Generating a New Trip Sequence, Avoiding Revisiting Cities

```
Input: Frequency table, MAXLENGTH
Output: Sequence of trips
Sequence ← empty sequence of trips;
Add the most frequently occurring trip in the table to Sequence;
Delete all trips containing the start city of the last added trip (already visited);
while length(Sequence) < MAXLENGTH and frequency table ≠ ∅ do
    if Start city of the first trip in the frequency table = destination of the last trip in Sequence then
        Add that trip to the sequence;
        Delete all available trips containing the start city of the last added trip;
    else
        Insert a dummy link trip with the start city as the destination of the most frequent available trip;
        Delete all available trips containing the start city of the dummy trip;
        Add the most frequently available trip (now with a start city matching the destination city of the dummy trip);
        Delete all available trips containing the start city of the last added trip;
    end
end
```

Examining the provided frequency table, the trip (Carpi - Baggio) is incorporated into the new sequence of trips. Subsequently, all trips containing the already-visited city Carpi are eliminated, resulting in an updated frequency table:

Trip	Frequency (percentage)
Baggio - Verona	12 %
Genoa - Baggio	10 %
Verona - Baggio	8 %
Verona - Bologna	5 %
Bari - Baggio	5 %
Trieste - Cuneo	4 %

At this stage, as the most frequently available trip shares the start city with the destination city of the last added trip in the sequence, the trip (Baggio - Verona) is included in the sequence. All trips containing Baggio are then removed from the table, resulting in:

Trip	Frequency (percentage)
Verona - Bologna	5 %
Trieste - Cuneo	4 %

Similarly, the trip (Verona - Bologna) is added to the sequence, and all trips containing Verona are removed from the table, resulting in:

Trip	Frequency (percentage)
Trieste - Cuneo	4 %

Lastly, as the destination city (Bologna) of the last added trip does not match the start city of the first available trip, a dummy trip (Bologna - Trieste) is introduced before adding (Trieste - Cuneo).

The final sequence of trips is (Carpi - Baggio), (Baggio - Verona), (Verona, Bologna), (Bologna, Trieste), (Trieste, Cuneo). In this example, we have considered no maximum length for the output sequence.

In general, the maximum length of the updated routes can be determined based on the average length of the examined actual routes, rounded to the closest integer. Alternatively, it can be adjusted to allow for longer or shorter lengths, serving as a hyperparameter. It can be also set to be equal to the length of the standard route we are updating. It is important to note that the output sequence is not guaranteed to have a maximum length of $MAXLENGTH$, in situations where, before adding the last trip to the output sequence, a dummy trip must be included. In these cases, the output sequence will have a length of $MAXLENGTH + 1$.

After generating the new sequence of trips, the merchandise for each trip is updated by examining the items present in the corresponding trips of the actual routes. Only the most frequently occurring items (based on a predetermined threshold, set to 40% in the implementation) are included in the updated merchandise. Regarding quantities, for each included item, the quantity is averaged based on the occurrences in the considered actual routes and then rounded to the closest integer.

Example. Assume we are updating standard route 1 and we want to generate the merchandise for the trip (Trieste, Cuneo). Suppose this trip appears four times in the actual routes that implement standard route 1, with the following merchandise. We choose to include items that appear in at least 30% of the considered trips:

Id	**From**	**To**	**Water**	**Butter**	**Milk**	**Beer**	**Bread**
a1_14	Trieste	Cuneo	**4**	0	**2**	0	0
a1_89	Trieste	Cuneo	**7**	**4**	**3**	0	0
a1_244	Trieste	Cuneo	**5**	0	**7**	0	0
a1_788	Trieste	Cuneo	**3**	0	**5**	0	**2**

The merchandise for the trip (Trieste, Cuneo) will be (Water: 5, Milk: 4).

Note that this algorithm involves calculating the frequencies of each trip for every set of actual routes related to a standard route. A trip, in this context, is treated as an ordered pair of two distinct cities. Therefore, for a set of n cities, there could be potentially $n \cdot (n-1) \sim n^2$ trips. If n is large, maintaining a counter in memory for each trip might become impractical.

In such cases, a technique similar to the one used in the Park, Chen, and Yu (PCY) algorithm could be beneficial. Rather than counting the frequency of each trip directly, trips in the current batch of actual routes can be hashed to specific buckets. A counter is then kept in memory only for those trips that are hashed to frequent buckets. A bucket is considered frequent if it contains a number of trips larger than a predetermined threshold s. This way, infrequent trips are not counted and added to the frequency table, reducing memory requirements.

However, it is worth noting that while this approach is beneficial for memory, it requires going through the trips of the considered actual routes twice, first to hash trips to buckets and then to count the trips.

4.2 Task 2: Tailored recommendation of standard routes to drivers

In this section, we outline a method to recommend, to each driver, the top five standard routes that align most closely with the driver's historical preferences, chosen from the pool of standard routes originally provided by the company and those recommended in the previous section.

An elementary algorithm can be outlined as follows:

(1) Group the actual routes by driver.
(2) Within each driver's set of routes, further categorize them based on the standard routes they implement.
(3) Calculate the average similarity between each standard route (that the current driver has implemented at least once) and the actual routes that implement that standard route. Repeat the same process also with the updated versions of the standard routes.
(4) Identify, for each driver, the five standard routes (from the original or updated set) with the highest average similarities to their corresponding actual routes.

In simpler terms, our goal is to determine, from the standard routes implemented by each driver in the past, which ones they have consistently adhered to the most (on average).

Figure 1 provides an illustrative example of this process for a specific driver.

However, this algorithm requires the computation of the similarity between each actual route and its corresponding standard route, using the previously introduced similarity measure. To address the computational cost associated with this basic algorithm, a refined version has been implemented. This improved approach begins with a preliminary step to filter standard routes, selecting only a limited number of routes per driver likely to exhibit high average similarity with their corresponding actual routes. Following this, the algorithm proceeds similarly to the basic version, computing, for each driver, the average similarity between each filtered standard route and the corresponding actual routes. Therefore, this process operates exclusively on the chosen standard routes, minimizing computational overhead.

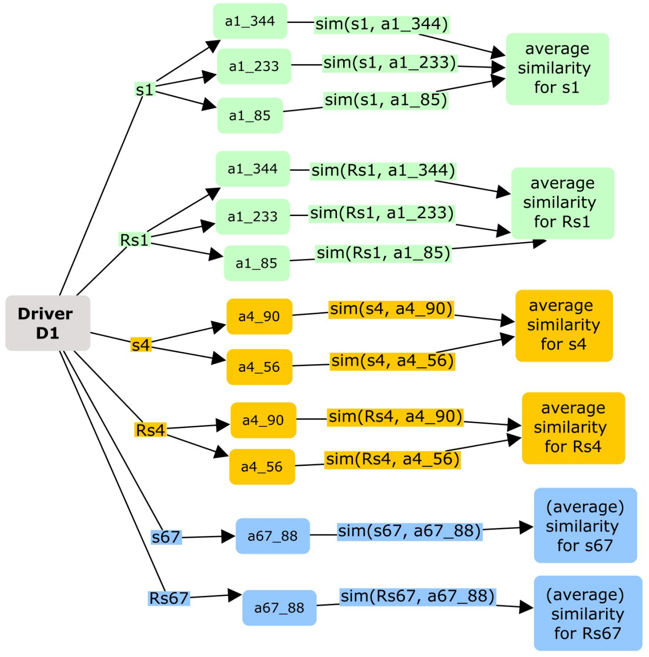

Figure 1: Average similarity of each SR implemented by driver D1

The algorithm's steps are outlined as follows:

(1) Transform each actual and standard route into a set, where the set representation includes all cities visited in the route and all merchandise items transported along that route.
(2) For each driver, employ minhashing (with a predetermined, potentially small, number of hash functions) to estimate the (average) Jaccard similarity between the set representation of each standard route (undertaken by the driver at least once) and the set representation of the corresponding actual routes.
(3) Select the n (> 5) standard routes with the highest average (estimated) similarity scores.
(4) For each of the n selected standard routes, compute the "true" average similarity - here, "true" refers to using the previously introduced similarity measure - with the corresponding actual routes of the current driver.
(5) Recommend the five standard routes, among the n selected, with the highest average similarities.

It is important to note that converting routes to sets and using minhashing for Jaccard similarity estimation is a preliminary step. This step allows for the rapid identification of routes that drivers are likely to follow closely. However, it is crucial to recognize that routes exhibiting high similarity in their set representations may not necessarily be similar according to the introduced measure. The introduced measure accounts for factors such as the start and destination city, as well as the trip in which each set of merchandise items is transported.

As an example, consider the two following routes:

Standard route:

(1) Trip: Rome to Milan - (milk: 3, pens: 10, butter: 20)
(2) Trip: Milan to Verona - (milk: 5, honey: 9, butter: 10, tomatoes: 20)
(3) Trip: Verona to Venice - (butter: 7, pens: 2, tomatoes: 10)

Actual route:

(1) Trip: Venice to Milan - (honey: 9, tomatoes: 20)
(2) Trip: Milan to Rome - (milk: 3, pens: 10, butter: 20)
(3) Trip: Rome to Verona - (pens: 8)

The set representations of these two routes are identical, implying a Jaccard similarity of 1. However, the similarity between these routes, as measured by the previously introduced similarity measure, is 0. Therefore, while minhashing serves as a valuable filter, the subsequent analysis ensures a more comprehensive evaluation of route similarity.

4.3 Task 3: Generating an ideal route for each driver

To address this task, a methodology similar to the one employed for Task 1 is applied, with the key distinction that the actual routes are now categorized based on the driver who implemented them, rather than the standard route.

In detail, the generation of an ideal route for driver D entails the selection of all actual routes attributed to this driver. Subsequently, an ordered frequency table of all the trips is established, considering only those trips surpassing a predefined frequency threshold. Using the Algorithm 1, introduced for Task 1, an ideal route is then generated, emphasizing trips with higher frequencies.

5 EXPERIMENTAL EVALUATION

5.1 Datasets

The datasets used for evaluating the methods presented in the preceding sections are synthetically generated using Python. The generation process involves creating a set of standard routes, with each standard route being used as a template for generating modified actual routes.

Here is an overview of the steps involved in the synthetic dataset generation process:

(1) **Initialization.** The process begins with a predefined set of cities, a collection of merchandise items, and a list of drivers.
(2) **Standard routes generation:**
  - For each standard route to be generated, a sequence of cities (to be visited along the route) is randomly chosen. The number of cities to visit is determined by sampling from a uniform distribution, resulting in a count between *MINNCITIES* and *MAXNCITIES*.
  - Merchandise items are assigned to each trip within the standard route. Items are randomly selected from the set of merchandise types, and random quantities are assigned within specified limits.
(3) **Actual routes generation:**
  - For each standard route generated, a random number of actual routes is created. The count is sampled from a normal distribution with a given mean and standard deviation.
  - Each actual route is slightly modified from its corresponding standard route. Modifications include replacing a fixed percentage of cities and adjusting merchandise items. Specifically, a percentage of the items of each trip are modified, with a subset having their quantities adjusted, and the remaining items being replaced with different ones.
  - Additionally, the route may undergo further modifications with probabilities P_ADD and P_REMOVE, determining the likelihood of adding or removing cities along the route.

The entire process is repeated using different random seeds to ensure reproducibility. Optionally, a smaller set of cities can be used for city replacements to simulate some repetition in the behavior of drivers. No routes have cities being visited multiple times.

For a more detailed explanation of the parameters that can be customized to generate a tailored dataset of routes, please refer to the documentation available in the file "*generate_randon_routes.py*" included in the "Dataset_Generation" directory of the project. Each utilized dataset is associated with a configuration file where all the parameters have been specified and customized.

Four datasets will be used, each with different characteristics. All these four datasets have 100 standard routes and for each standard route there are on average 400 actual routes.

(1) **Dataset 1:** This dataset starts from a set of 100 cities, a set of 48 types of merchandise and a set of 30 drivers. Each trip in the standard routes initially includes from 3 to 6 cities and a number of items between 1 and 10, with quantities between 1 and 30. In every actual route, 20% of the cities of the corresponding standard route are replaced. Then, for each trip in the actual routes, 50% of the merchandise items are left unmodified, while, among the remaining 50% of the items, 50% gets replaced by another item and the 50% left has its quantity adjusted. Lastly, with probabilities of 20% cities can be added or removed to the actual routes.
(2) **Dataset 2:** In this dataset, there are 93 cities available to generate standard routes. Furthermore, there is a separate set of 7 cities, from which to select replacements or additions when creating actual routes. The probability of choosing a city from this set is not uniform, simulating repetitive patterns in driver behavior. There are 20 drivers, and the number of items for each trip ranges from 1 to 8. In every actual route, 60% of the cities from the corresponding standard route are replaced. The probabilities of adding or removing a city are both set to 40%.
(3) **Dataset 3:** Similar to dataset 2, dataset 3 also features a distinct set of 7 cities from which replacements or additions are chosen when creating actual routes. However, in this case, each of the 7 cities has an equal probability of being selected. Trip lengths are allowed to be longer, ranging from 5 to 10 cities before any additions or removals. The key distinction lies in deliberately increasing the probabilities of adding or removing a city, both set at 60%, to simulate scenarios where drivers deviate significantly from the standard routes.
(4) **Dataset 4:** The variations introduced in Dataset 4, compared to Dataset 3, include an increase in the percentage of cities from the standard routes replaced in the actual routes to 80%. Additionally, 70% of the merchandise item sets are modified, and the probability of removing a city is raised to 70%.

For further details, see the configuration files "*config_dataset_i.py*" ($i = 1, \cdots, 4$) contained in the directory "Dataset_Generation".

5.2 Evaluation of Task 1

To evaluate the efficacy of the algorithm designed for Task 1, the following criteria will be used. For each standard route, the average similarity with its corresponding actual routes will be computed. This process will be performed twice: initially using the original standard routes and then with the recommended standard routes. The frequency threshold, determining the inclusion of trips (considering only start and destination cities) in the frequency table used to generate the updated sequence of trips, is set to 2%. The parameter $MAXLENGTH$ employed in the function generating the updated trip sequence for standard route i is set equal to the length of the original standard route i, incremented by 2.

The evaluation will consider:

- The average increase or decrease in similarity across all standard routes.
- The frequency of cases where the recommended standard routes led to an increase in similarity.
- The average increase over the standard routes that exhibited an increase in similarity.

For **dataset 1**, the average similarity for the original standard routes is 54.7%, reducing to 46.4% for the recommended standard routes. Consequently, the algorithm performs poorly, resulting in an average decrease in similarity of around 8.3%. The average similarity decreases for all 100 standard routes. However, this outcome should not be surprising. Consider that our algorithm identifies frequent trips in the behavior of drivers executing actual routes. If drivers consistently modify standard routes with random alterations, there will not be any prevalent modification that our algorithm can capture. In other words, if drivers make random modifications to standard routes without adhering to any patterns (as simulated in our dataset), there is no way to identify improved standard routes. This rationale underlies the choices made in creating the other datasets.

For **dataset 2**, the average similarity for the original standard routes is 23.4%, which decreases to 21.7% for the recommended standard routes. Again, the algorithm shows suboptimal performance, resulting in an average decrease in similarity of around 1.7%. However, there are 34 standard routes (out of 100) for which an increase in similarity is observed. The average increase over these 34 standard routes is approximately 1.2%.

For **datasets 3 and 4**, the algorithm's performance exhibits a substantial improvement.

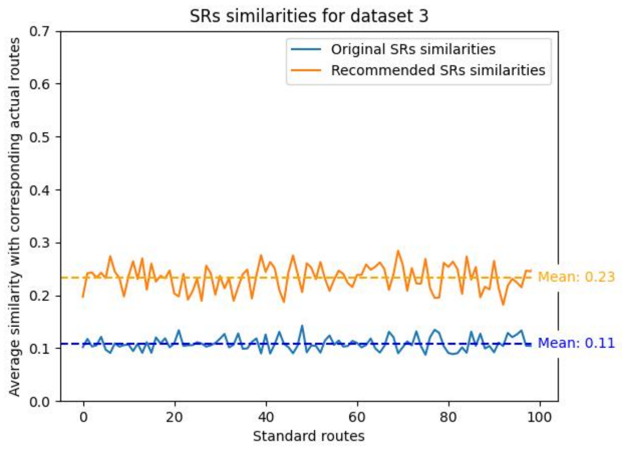

Figure 2: SRs average similarities on Dataset 3

As illustrated in the plot, the original standard routes start with an average similarity of around 10.8%, while the recommended ones show a similarity of 23.3%, indicating an average increase of around 12.5%. In this case, all 100 standard routes experienced an increase in similarity.

The original standard routes for **dataset 4** begin with an average similarity of around 5.3%, while the recommended ones achieve a similarity of 25.7%, resulting in a substantial average increase of around 20.4%. Similarly, all 100 standard routes in this case experienced an increase in similarity.

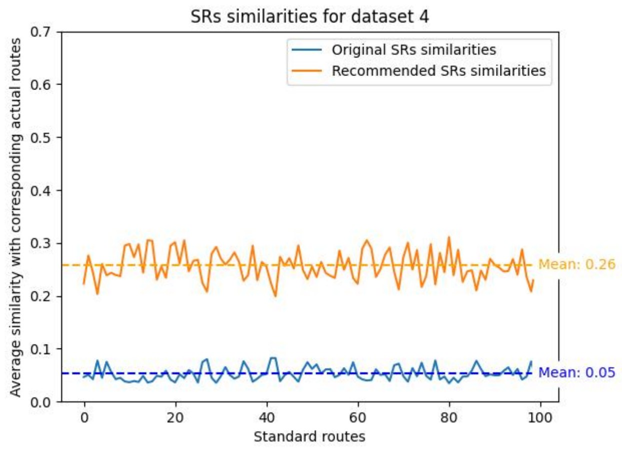

Figure 3: SRs average similarities on Dataset 4

Based on these experiments, it seems prudent to deploy the introduced algorithm primarily in cases where the average similarity between the actual routes and their corresponding standard route is approximately below 20%. Furthermore, the algorithm tends to yield better results when the actual routes exhibit less adherence to the corresponding standard routes.

5.3 Evaluation of Task 2

To assess the effectiveness of the algorithm developed for this task, I will conduct, for each driver, a comparison between the top 5 list recommended by the basic algorithm presented in Section 4.2 (which represents the optimal outcome achievable given a specific measure of similarity between a standard route and an actual route) and the top 5 list recommended by the algorithm using minhashing to filter standard routes.

In comparing the dissimilarity between two ordered lists of 5 items (items are strings denoting SR IDs in this case), two approaches will be considered:

(1) **Counting shared items**: One approach involves counting, for each driver, how many of the SRs included in the optimal top 5 list are also included in the suggested top 5 list. However, this method does not take into account the order of the routes in the lists, which is an important factor for this task.
(2) **Edit distance method**: A more robust and stringent alternative is to employ a modified version of the edit distance. Specifically, each standard route

ID in the two top 5 lists will be mapped to a character, and the edit distance will be computed for the resulting 5-character strings. If two top 5 lists exhibit an edit distance of 0, it indicates they are identical, both in terms of the routes contained and their order. Conversely, an edit distance of 5 with a count of 5 for the previous measure implies that the lists have the same SRs but in reverse order.

For all the 4 datasets, 200 hash tables have been used for the minhashing estimation of the Jaccard similarities.

For **Dataset 1**, in the preliminary filtering step, $n = 20$ standard routes have been selected for each driver. The top 5 lists recommended have an average shared count of 0.8 with the optimal top 5 lists and an average edit distance of 1.47. This suggests that, on average, the advanced algorithm correctly includes 4 out of 5 optimal SRs in the proposed top 5 lists, without considering the order. Notably, for 40% of drivers, both algorithms propose the same top 5 list. Moreover, out of the 150 standard routes suggested in the top 5 lists, 139 are from the original standard routes, while the remaining 11 are from the updated ones. This outcome is understandable, given that, for Dataset 1, the recommendation algorithm used in task 1 has shown poor performance, leading drivers to adhere more to the original standard routes on average. Additionally, it is worth mentioning that the first route, i.e., the standard route from which each driver deviated the least, was correctly identified for 27 out of 30 drivers.

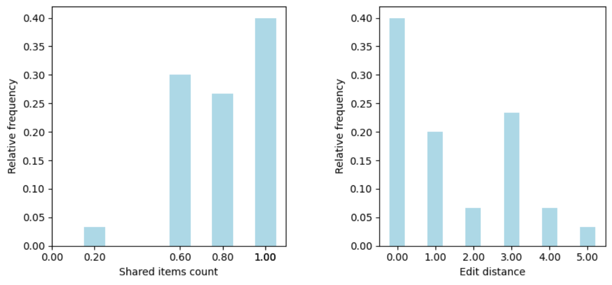

Figure 4: Performance measures for Task 2 on Dataset 1, with n=20

For **Dataset 2**, with $n = 20$ standard routes selected for each driver in the filtering step, the average shared count of SRs is 0.48 and the average edit distance is 3.55. This indicates that the more advanced algorithm is not performing as good as in the previous case, including approximately half of the optimal routes in the suggested top 5 lists. For no driver, both algorithms propose the same top 5 list. Moreover, out of the 100 standard routes suggested in the top 5 lists, 85 are from the original standard routes, while the remaining 15 are from the updated ones. The first route in the optimal top 5 lists was correctly included as the first one in suggested the top 5 lists for 12 drivers out of 20.

However, by increasing n to 50, the average shared count increases to 0.67 and the average edit distance of decreases to 2.5. Therefore, on average, between 3 and 4 of the optimal routes are included in the top 5 list of each driver (without taking the order into account). Moreover, 10% of the drivers get the optimal top 5 list and 14 drivers out of 20 are recommended the correct first standard route.

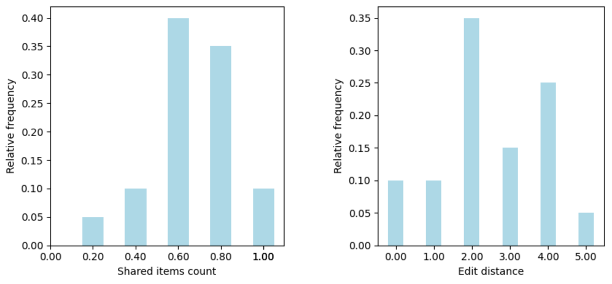

Figure 5: Performance measures for Task 2 on Dataset 2, with n=50

For **Dataset 3**, the algorithm's performance experiences a significant decline, reflected in an average shared count of 0.08 (with a shared count of 0 for 14 drivers out of 20). The edit distance is 5 for all drivers, except for one, where it is 3. Moreover, the algorithm correctly identified the first route for only 1 out of 20 drivers.

In contrast to the previous cases, all included routes are from the recommended ones. This behavior is not surprising given that, for dataset 3, the recommended routes resulted in a substantial increase in average similarity with the corresponding actual routes.

Despite increasing the value of n to 50, no significant improvement is observed in this scenario.

Similar to dataset 3, **Dataset 4** exhibits poor algorithm performance when $n = 20$. This is evident in a shared count of 0.06, with a shared count of 0 for 16 out of 20 drivers. The edit distance is consistently 5 for most drivers, except for three cases where it is 4.

Additionally, the algorithm correctly identifies the first route for only 3 out of 20 drivers. Also in this scenario, all included routes are from the recommended ones.

However, increasing the value of n to 50 results in a significant improvement in performance. The average shared count increases to 0.37, even though the average edit distance decreases slightly to 4.2. Furthermore, the algorithm correctly identifies the first route for 10 out of 20 drivers, and for all drivers, at least one route from the optimal top 5 is included in the suggested top 5.

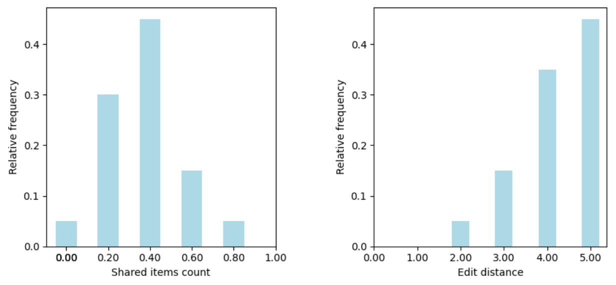

Figure 6: Performance measures for Task 2 on Dataset 4, with n=50

Overall, the algorithm exhibits strong performance in cases where the average similarity of the actual routes to their corresponding standard routes is relatively high, indicating that drivers adhere closely to the scheduled routes. This is evident in datasets 1 and 2, where there is less risk that the estimated Jaccard similarity significantly diverges from the actual similarity. However, when this assumption no longer holds, relying solely on the estimated Jaccard similarity between actual routes and standard routes becomes insufficient.

5.4 Evaluation of Task 3

For the evaluation of this final task, we will adopt a similar approach to the one used for task 1. Our aim is to compare each driver's ideal route to all the actual routes implemented by the driver and compute the average similarity. We will employ the same similarity measure utilized thus far, treating the ideal route as if it were a standard route. The results are shown in Table 1.

The results indicate a performance trend in the algorithm similar to that observed in Task 1, that increases as the dataset number increases (i.e., as we enforce more repetitive behavior in the implementation of actual routes). Moreover, an improvement in performance can be observed when placing greater weight on the Jaccard similarity between cities, hinting at the influence of merchandise updating on low similarity scores.

Dataset	Avg. sim. w1 = 0.3	Std. dev.	Avg. sim. w1 = 0.5	Std. dev.	Avg. sim. w1 = 0.7	Std. dev.
1	1.6%	3.8%	2.3%	5%	3%	6.3%
2	4.6%	3.6%	7.5%	5.8%	10.5%	8%
3	12.4%	5.1%	20.4%	8.5%	28.6%	12%
4	14.3%	5.9%	23.6%	9.6%	33%	13.4%

Table 1: Driver's average similarities to their ideal routes

However, the achieved accuracy levels in datasets 3 and 4 fall short of those attained in Task 1. One possible explanation is the assignment of routes to drivers in the synthetic dataset generation process. Specifically, when creating an actual route, a random driver is assigned to that route with equal probability. Consequently, each driver is likely to have implemented all, or nearly all, standard routes approximately the same number of times. This uniform distribution across drivers makes it challenging for the algorithm to capture prevalent patterns in the implementation of actual routes even in datasets 3 and 4, where patterns are introduced. Since each driver has implemented numerous standard routes, the algorithm struggles to discern predominant features that occur most frequently.

A scenario where this algorithm is likely to perform much better is if each driver is in charge of performing only a few (one or two) standard routes. In such a scenario, the algorithm could effectively capture a driver's most frequent trips, which do not differ too much from each other as they perform the same standard routes repeatedly. This would allow the algorithm to generate an ideal route by prioritizing the driver's frequent trips. However, this hypothesis will not be tested in this project, as it necessitates the creation of a dataset using a markedly different methodology.

6 CONCLUSION

In conclusion, this project addresses the challenges in transportation faced by logistics companies, particularly in managing deviations from planned routes by truck drivers. The significance of efficient route planning for resource management and customer satisfaction is highlighted. The proposed system aims to align assigned routes with driver preferences and minimize discrepancies between standard and actual routes.

While the system excels in certain conditions, it also highlights the importance of dataset characteristics in determining algorithm efficacy. Future work should explore variations in dataset generation and algorithm fine-tuning to enhance overall performance.

REFERENCES

[1] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of massive data sets.* Cambridge university press.