Informatics

Search Space Reduction Through Machine Learning for the Electric Autonomous Dial-A-Ride Problem

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software and Information Engineering

by

Caspian Coleman
Registration Number 12038410

to the Faculty of Informatics

at the TU Wien

Advisor: Günther Raidl
Assistance: Maria Bresich

Vienna, September 28, 2024

Caspian Coleman

Günther Raidl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Caspian Coleman

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt.

Wien, 28. September 2024

Caspian Coleman

Acknowledgements

Standing on the shoulders of giants. Therefore I would like to express my deepest gratitude towards my supervisors that supported me troughout the development process of the bachelor thesis: Günther Raidl and Maria Bresich as they provided both the theoretical basis and code implementation of the underlying E-ADARP. I would also like to express my deepest gratitude towards Dr. Pascal Welke who supervised me during the development of a machine learning project that became part of the foundation of this bachelor thesis. For the development of the machine learning models I want to also thank the Machine Learning Research Unit of the TU Wien.

Abstract

Make traffic great again. That is the central goal of the electric autonomous dial-a-ride problem (E-ADARP). The principal idea is to use self-driving cars that are fueled by electricity, which then can just be ordered by a customer to a certain location to pick them up and drop them off at another specified location. During the trip additional customers can also be picked up. enabling ride sharing, or, if the introduced detour is too great, they can be served after the previous customer is dropped off. However, the complexity of the E-ADARP leads to a steep performance decrease with increasing problem size. Especially the route planning for the vehicles hinders timely discovery of E-ADARP solutions. This thesis therefore explores the usage of machine learning (ML) to reduce the computational complexity of a given E-ADAR-Problem by removing parts of the search space before actually solving the problem and consequently accelerating the solving process. Specifically, this thesis identifies features representing high quality areas of the search space, creates datasets based on these features, trains for different variants of a Support Vector Model and tests both their individual predictive power as well as how they contribute to a speed-up of a given algorithm for the E-ADARP. The results show great run time reductions of the solving process while barely reducing solution quality or even improving it.

Contents

CHAPTER 1

Introduction

Electric autonomous vehicles (EAV) are a powerful and prospective answer for current challenges in traffic and public transport, especially as a combined fleet in urban areas. The resulting routing problem for a fleet of this particular structure is called the electric autonomous dial-a-ride problem (E-ADARP), which plans a route for each EAV, picking up and dropping off customers at their requested locations.
So, why would such a fleet of EAVs be helpful in the first place? First off, a network of these EAVs keeps travel individual while reducing both the high cost that comes with owning a car and the time wasted with the car just being parked somewhere. E.g., a typical day for a working-class individual could be driving to work in the morning, working for eight hours, driving home, doing arbitrary things back at home, and going to sleep. As is visible, the car is not used most of the time. A fleet of self-driving vehicles tackles this by being available for the next customer right after having finished dealing with the first. Next, this approach further increases the efficiency by being able to carry more than one passenger at a time, even with different drop-off locations. This improvement leads to a reduction of congestion. Lastly, because of the increase in efficiency and the usage of electric vehicles, the environmental impact is reduced.

However, to realistically be able to plan and set routes for a greater number of EAVs, the computational complexity of the E-ADARP demands an efficient solving process. Therefore, various heuristics are being used, including deterministic annealing [SDP23] and large neighborhood search (LNS) [BRL24]. Yet, to further enhance the performance for big problem sizes, the search space can be pruned, as proposed in [AGS19]. And here, machine learning (ML) can help with improving these complex pruning decisions. This thesis, therefore, explores an ML-based pruning approach through identifying structures in existing solutions for given problem instances of the E-ADARP, and using this knowledge of the structure of solutions to remove parts of the problem instances that are likely not contained in close-to-optimal solutions beforehand. This starts off with an extensive analysis of existing data, followed up by a development of four variants of Support

Vector Machines, including the definition of both features and labels, based on a weak supervision approach. Next, these models will be put to the test on current standard problem instances to assess the individual performance of each model, as well as their contribution to the speed-up of the solving process of the given instances. The results show that a considerable speed-up is possible while losing little solution quality or even improving it.

Before delving deeper into the pruning process, first off, the definition of the underlying electric autonomous dial-a-ride problem is given, which is directly adopted from the works of [BKG19] and [BRL24].

1.1 Problem Definition

The E-ADARP is defined on a directed graph $G = (V, A)$, with vertex set V that represents all locations part of the instance, and arc set $A = (i, j) : i, j \in V, i \neq j$. Graph G is complete in its original definition; however, the preprocessing steps, including the ML pruning explained later, change this. With that, there exists a fleet of n_K vehicles, denoted by $K = 1, \ldots, n_K$, which provide service for n customer requests within a planning horizon of T^{plan} time units. The set of locations, $V = N \cup O \cup F \cup S$, is composed of the following:

- $N = P \cup D$ is the set of all customer related locations, with:
  - $P = \{1, \ldots, n\}$, representing the pickup locations and
  - $D = \{n + 1, \ldots, 2n\}$, representing the drop-off locations.
  - Each combined pair $(i, i + n)$ represents the i-th request, with pickup location $i \in P$ and drop-off location $i + n \in D$. Additionally, there exists a maximum user-ride time u_i.
- O is the set of all origin depots. Each vehicle $k \in K$ has an assigned origin depot location $o_k \in O$, where it starts service at the beginning of T.
- F is the set of all destination depots. Each vehicle $k \in K$ has an assigned destination depot location $f_k \in F$, where it arrives at the end of service to finish.
- S is the set of all available charging stations. In addition to that, each $s \in S$ has a charging rate α_s, specifying the amount of energy charged per time unit.

For the temporal dimension, each $i \in V$ has an earliest possible service time w_i^{start} and a latest possible service time w_i^{end}. Further, pickup and drop-off locations (so the locations that make up requests) have a service duration d_i, which represents customers entering or leaving the vehicle and other interactions. Next, each vehicle has a load, representing the amount of customers it is carrying at the moment, and a maximum load capacity C_k. This load increases by l_i at $i \in P$, and decreases by l_{i-n} at $i \in D$. Each vehicle has a maximum battery capacity of Q and an initial battery level $B_{k,1}$ at the start of the planning horizon. Lastly, each arc $(i, j) \in A$ is associated with a travel time $t_{i,j}$ and a battery consumption $\beta_{i,j}$. An example of such a graph can be seen in Figure 1.1.

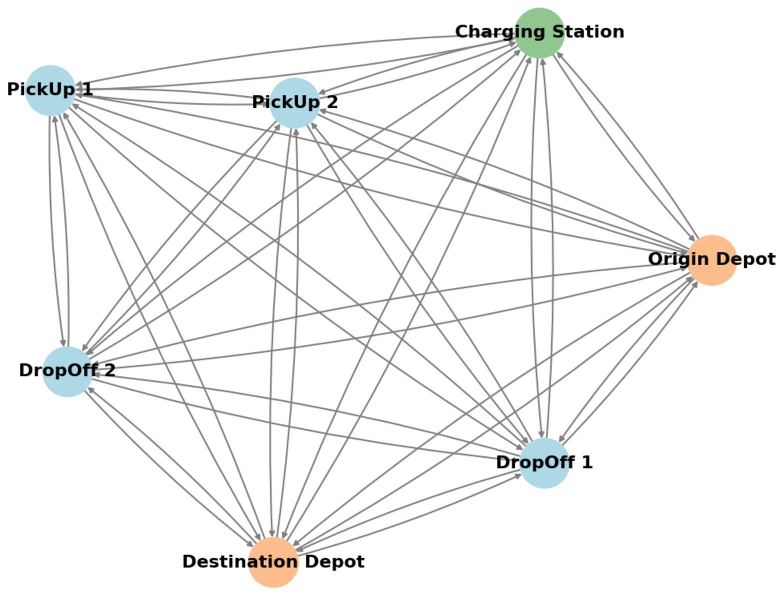


Figure 1.1: Example of a graph

A route of a vehicle is then a combination of possible locations $i \in V$ starting at an origin depot and ending at a destination depot. Therefore, an E-ADARP solution would be a composition of these n_k vehicle routes with also fixing a service start time t_i^{serv} at each location/node i of the route. For a solution to be feasible, it then also has to serve every request exactly once: For each request $i = 1, \dots, n$ there exists one route that visits pickup location i and drop-off location $i + n$ in this order. Additionally, the maximum user ride time and maximum battery capacity can not be exceed, the given time windows not violated and for each vehicle the battery may not run out during the route. Because the E-ADARP also takes the electricity consumption into account the usage of charging stations is required. As a consequence of the constraints and insertion of charging stations, additional algorithms become necessary to both evaluate the routes, to ensure all constraints are met, and place the charging stations at efficient positions in the route. Examples of these algorithms can be seen in [BRL24].

1.2 Objective

E-ADARP then aims at determining a feasible solution which minimizes following objective function:

$$\min \ w^{\text{routing}} \sum_{k \in K} \sum_{(i,j) \in A} t_{i,j} x_{k,i,j} + w^{\text{excess}} \sum_{i \in P} t_i^{\text{excess}}. \tag{1.1}$$

This therefore is a weighted combination of the total travel time over all routes and the total user excess ride time for all customers. Then w^{routing} and w^{excess} are the weighting factors between these two addends. Next, $x_{k,i,j}$ is a binary decision variable representing if vehicle k travels from location i directly to j. To calculate the user excess ride time t_i^{excess} for request $i \in P$ following formula is used:

$$t_i^{\text{excess}} = t_{i+n}^{\text{serv}} - t_i^{\text{serv}} - d_i - t_{i,i+n}.$$

That is the difference between the actual ride time of the customers of request i and the travel time $t_{i,i+n}$ it would take travelling directly from pickup to drop-off-location of the customer.

1.3 Research Questions

In the context of the now defined E-ADARP the research question is: To what extent can machine learning speed-up route planning of the E-ADARP through pruning the search space before searching for the best solution using a state-of-the art solving heuristic?
For specification, a reduction/pruning of the search space here means removing some of the possible options during the search for the best solution. While possibilities for the reduction are plenty (including focusing on whole routes, combinations of requests, insertions of single requests etc.) the main focus will be on the simplest approach by trying to remove singular arcs $(i, j) \in A$ before the solving process. While it is also possible to dynamically remove arcs during the solving process of a heuristic (e.g., an LNS), this will not be addressed here. Other options, including whole routes and single requests, will also be analyzed, yet not further considered during the pruning process.
As basis for the analysis an algorithm and its implementation developed by Bresich et al. ([BRL24]) will be used. In the cited paper they presented two variants of an LNS to solve the E-ADARP, of which the better performing was chosen for further study. This thesis will continue in first understanding the data and structure of both instances and solutions of the instances to set a necessary basis for the ML models.

CHAPTER 2

Data Analysis

The overarching narrative of data analysis here is to detect structures of already existing solutions, and use this knowledge to first ensure that there exist features that would enable an ML model to classify data points, and secondly, use these features to train the ML model. The process starts with solution generation, which uses the mentioned LNS by Bresich et al. These solutions are calculated via an heuristic but are of high quality and for many instances even the best found until now [BRL24]. Next, these solutions, which are a possible subset of its instances, are compared with the corresponding complete instances visually and statistically on different levels of depth. For example, a solution S consists of a number of routes $R = \{r_1, r_2, ..., r_m\}$, each containing a number of locations n_i with $1 \leq i \leq m$ which are connected by arcs $(i, j) \in A'$. Set A' represents the arcs that are part of the solution S. As the instance G is originally a complete graph, $A' \subset A$. Now, each arc has features, including arc length or relative location, that can be compared. The goal is then to detect those features of an arc that have a reduced range of values for arcs $(i, j) \in A'$ compared to all arcs $(i, j) \in A$. That is, arcs part of the solution have a certain structure that not all other arcs share. This can then be used by an ML model to classify between arcs of "high quality"-so arcs being part of a solution-and arcs of "low quality". This approach can be expanded for any part of the instance/solution, including complete routes and single requests. As a sidenote, the following analysis will mainly focus on instances of medium size as for the biggest instances the solutions were not of high enough quality. Additionally, the smallest instances were not of interest as the focus of the pruning is a speed-up of instances that benefit from that.

2.1 Arcs

As already defined, an instance is represented by a directed complete graph $G = (V, A)$ with every arc $\{(i, j) : i, j \in V, i \neq j\}$ holding following features that will be analyzed. The interest is here to classify between "good" and "bad" arcs. As the instances are originally complete, a linear rise of customers leads to a quadratic rise in arcs, which heavily impacts run time. If one would be able to decide beforehand if an arc would possibly be part of a solution, one could remove many arcs before the execution of the solving algorithm which tackles this problem.

2.2 Distance

Each arc $\{(i, j) : i, j \in V, i \neq j\}$ is defined by the two locations i and j which it connects. Now again, every location i has an assigned geographical coordinate vector $\mathbf{i} = (i_1, i_2)$. As a consequence each arc can be seen as a displacement vector in a two dimensional space connecting two position vectors representing the locations. On this mathematical basis the distance between each location to each other location can be quantified and, depending on which measurement of distance is used, assigned as feature to the corresponding arc. Further, this is an arc specific but not a location specific feature as-at least before the pruning process-every location is connected with every other location and therefore has $|V| - 1$ distances to $|V| - 1$ other locations. Next, the analyzed measurements for distance with definition and the respective results.

- **Euclidean distance:** As the classical measurement for distance, the Euclidean distance here for each $i, j \in V$ with $\mathbf{i} = (i_1, i_2)$ and $\mathbf{j} = (j_1, j_2)$ as the position vectors is defined as
$$d(\mathbf{i}, \mathbf{j}) = \sqrt{(j_1 - i_1)^2 + (j_2 - i_2)^2}$$
To now assess its usefulness as a feature, the Euclidean distances of the arcs, which were part of a solution of a given instance, were plotted against all Euclidean lengths of all arcs of the instance which can be seen, as an example, in Figure 2.1. To avoid overpopulating the thesis with plots, the remaining plots of other instances will not be depicted. Moreover, if a feature varies in structure across different instances, this will, of course. be addressed. The same approach also holds true for the remaining part of this thesis.

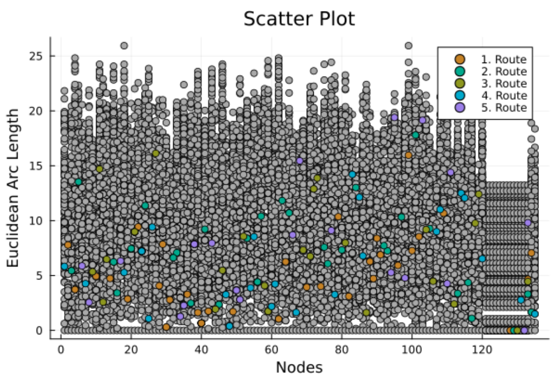


Figure 2.1: Example plot Euclidean distance

As visible in the example plot, yet also holding true for all instances, arcs part of the solution tend to be shorter in terms of Euclidean distance with exceptions however being relatively long. Therefore, the distribution of Euclidean distances for arcs part of the solution is not equal to the one of all arcs, which qualify this feature (while not being perfect) for the ML model. In addition to that, this plot structure repeats itself in similar fashion for all measurements of distances to follow.

- **Similarity based on $\cos\alpha$:** As the next measurement of distance α here is the angle between the two displacement vectors defined through $\mathbf{i}$ and $\mathbf{j}$. Therefore $\mathbf{i}$ and $\mathbf{j}$ are here not interpreted as position vectors but displacement vectors. Note, that, even though $\cos\alpha$ is not a true distance measurement, it is used here as such by interpreting higher similarity as being closer together. Further, this measurement, in contrast to the Euclidean distance, is independent of the lengths of both $\mathbf{i}$ and $\mathbf{j}$. Following, the similarity based on $\cos\alpha$ is here defined as:

$$d_{\cos\alpha}(\mathbf{i},\mathbf{j}) = \frac{\mathbf{i}^T\mathbf{j}}{\sqrt{(\mathbf{i}^T\mathbf{i})(\mathbf{j}^T\mathbf{j})}} = \frac{\mathbf{i}^T\mathbf{j}}{\|\mathbf{i}\| \cdot \|\mathbf{j}\|}$$

The results can be exemplarily seen in Figure 2.2. In terms of interpretation, a cosinus value of 1 is given, when the displacement vector of both locations have exactly the same angle, which means they are minimized in proximity to one another, and -1 when the angle is 180° and the two locations are as far apart as possible. Note for the distance, the smaller of the two possible angles is always used. Because of that, for an angle greater than 180°, the explementary angle would be smaller than 180° and is used instead. While it can be seen in the plot that most locations part of the solution are close to a value of 1 and therefore are relatively close to each other, a shorter distance between locations in the solution seems to be less prevalent compared to other distance measurements. As a consequence, the similarity based on $\cos\alpha$ performs worst of the distance metrics.

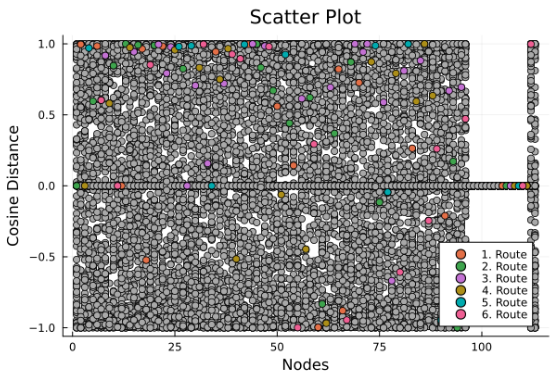


Figure 2.2: Example plot similarity based on $\cos\alpha$

- **Manhattan distance**: The Manhattan distance is a simpler and more robust alternative to the Euclidean distance and defined here as:

  $$d_M(\mathbf{i}, \mathbf{j}) = |i_1 - j_1| + |i_2 - j_2|$$

  Similarly, the results also resemble those obtained with Euclidean distance, as can be seen in the example plot 2.3, and make this metric usable as a feature. Just as in the plots for Euclidean distance, here the distances of the arcs part of the solution are plotted against all arcs part of the instance.

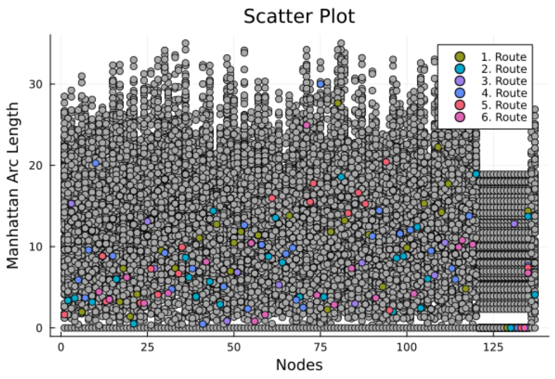


Figure 2.3: Example plot Manhattan distance

- **α-Nearness**: Another analyzed distance measurement used in the context of graphs, the α-Nearness, is defined here as follows: Consider a weighted, undirected graph $G_1 = (V, E_1)$, where V is the set of vertices and E_1 is the set of undirected arcs between each vertex in V. Each arc $e_{i,j} \in E_1$ has a weight $w(e_{i,j})$ equal to the $d(\mathbf{i}, \mathbf{j})$ between locations $i, j \in V$ with assigned position vector $\mathbf{i}$ and $\mathbf{j}$. Note the earlier definition of $d(\mathbf{i}, \mathbf{j})$ under Euclidean distance. Let T be a minimum spanning tree of G_1 and $T_{(i,j)}$ be a minimum spanning tree that includes the arc $(i, j) \in E_1$.

  Then the $\alpha-$Nearness is the difference between the sum of weights in the minimum spanning tree T and the sum of weights of the minimum spanning tree including an arc $T_{(i,j)}$ [HK70]:

  $$d_\alpha(\mathbf{i}, \mathbf{j}) = \sum_{e \in T} w(e) - \sum_{e \in T_{(i,j)}} w(e).$$

  Now, this distance measurement can be applied to each arc in the instance graph. As analysis, if α-Nearness is sensible as a feature, the α-Nearness of arcs part of the solution was plotted against all arcs in the instance, which can be seen in Figure 2.4. The feature performs similarly to the Euclidean and Manhattan distances, which again makes it usable for the training of the ML model.

  Based on the analysis until now, Euclidean distance, Manhattan distance and α-Nearness perform similar and best. These features, however, are dependent on

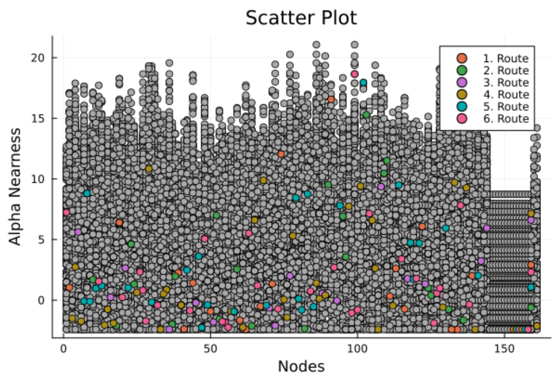


Figure 2.4: Example plot α-Nearness

the instance as another bigger instance in a greater geographical dimension would, e.g., also lead to a greater overall Euclidean distances, Manhattan distances and α-Nearness. This could lead to varying performance for different instances. As a consequence, the Euclidean distance relative to the variance of Euclidean distances and Euclidean distance relative to the total distance of arcs were also tested. The definitions for both metrics are given as:

– **Euclidean distance relative to variance** s^2

$$d_{s^2}(\mathbf{i},\mathbf{j}) = \frac{d(\mathbf{i},\mathbf{j})}{s^2}$$

with

$$s^2 = \frac{1}{n-1} \sum_{(i,j)\in A} (d(\mathbf{i},\mathbf{j}) - \overline{dist})^2$$

and

$$\overline{dist} = \frac{\sum_{(i,j)\in A} d(\mathbf{i},\mathbf{j})}{|V|}$$

– **Euclidean distance relative to total distance of arcs**:

$$d_{tot}(\mathbf{i},\mathbf{j}) = \frac{d(\mathbf{i},\mathbf{j})}{\sum_{(i,j)\in A} d(\mathbf{i},\mathbf{j})}$$

This is mentioned only for completeness, as the visualization, while executed, looks equivalent to the one of the standard Euclidean distance, just with smaller values on the y-axis, and all instances used in the thesis are of equal geographical size. Still, accounting for different variance or total distance over all routes might improve the performance of the feature. This relativization was not done for the remaining distance metrics because the original metrics performed comparably anyways, and the given time frame did not allow further analysis in this direction.

2.3 Spatial Position

In this next section, the thesis will consider a consequence of each arc having a distance value: The neighborhood between locations. Thus, as the graph is originally complete, a location has an arc to each other location, each with a given distance. Neighborhood here then is just a sorting of the locations based on these distances. Therefore, each location has one nearest neighbor, a second nearest neighbor and so forth. Based on that, two possible features were analyzed, starting with:

- **Nearness based on neighborhood**: For a given arc (i, j) the nearness based on neighborhood is here defined as:

$$\text{near}_{i,j} = |\{k \in V \setminus \{i, j\} | dist(\mathbf{i}, \mathbf{k}) < dist(\mathbf{i}, \mathbf{j})\}|$$

  This nearness value $\text{near}_{i,j}$ is then assigned to arc $\{(i, j) : i, j \in V, i \neq j\}$. In simple terms this resembles, starting from location i, how many locations are closer to i than j.

- **k-nearest neighbors**: Another similar variant to the nearness based on neighborhood is to see if, for an arc (i, j), j is part of the k-nearest neighbors to i. To be exact, this checks if $j \in N_k(i)$ with

$$N_k(i) = \{j \in V,\ k \leq |\{l \in V,\ dist(\mathbf{i}, \mathbf{l}) \leq dist(\mathbf{i}, \mathbf{j})\}|\}$$

  These two features were analyzed combined via plotting, an example of which can be seen in Figure 2.5. This plot is to be interpreted like follows: On the x-axis, the

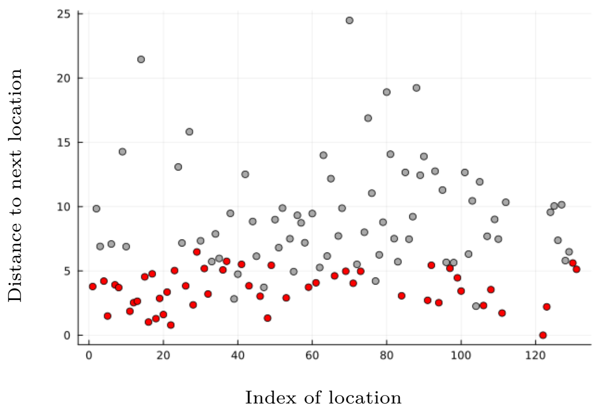


Figure 2.5: Example plot 20-nearest neighbors

indexes of the locations and on the y-axis the distance from each location to its subsequent location in the solution is plotted. Consequently, each point represents an arc (i, j) with the index of i on the x-axis and the distance from i to j on the y-axis. Further, each grey point is an arc (i, j) where $j \notin N_k(i)$ and each red point an arc (i, j) where $j \in N_k(i)$. In this example $k = 20$, however also $k = 3$, $k = 5$, $k = 10$, $k = 40$, $k = 60$, $k = 80$, $k = 100$ were tested. First, to qualify the feature of k-nearest neighbors as feasible, the number of red points should be clearly greater than k as this would indicate that neighboring locations tend to be more inside the neighborhood than outside. Yet, even up to $k = 100$, arcs remain with the second location being outside the 100 nearest neighbors of the first, and while the number of red points is greater than k, it is overall visible that the feature, independent of k, does not perform particularly well. The same conclusion can then also be taken for nearness based on neighborhood. That is because, here, even for big k values, there were still many points outside the k-nearest neighbors range, indicating that the feature will not perform well when differentiating between arcs part of the solution and arcs which are not.

2.4 Temporal Position

In the last section of analyzed features concerning singular arcs, another dimension next to the dimension of place is considered: time. First off, here the time it takes a vehicle to get from location A to location B is directly tied with the distance between the two locations. Therefore, this will not be further considered as a feature. Next, as mentioned, every location i has an earliest possible service start time w_i^{start} and a latest possible service start time w_i^{end} assigned to it. The analysis of how these values help in the classification process is done via following features. The overall idea is that closer time windows are favorable.

- **Difference of earliest service start times of locations**: For an arc (i, j) with $i, j \in V$ this is
$$d_{(i,j)}^{start} = w_i^{\text{start}} - w_j^{\text{start}}.$$
Then again to analyze this feature all $d_{(i,j)}^{start}$, with (i, j) being part of a route of a solution, are plotted against all $d_{(k,l)}^{start}$ with $k, l \in V, k \neq l$. An example can be seen in Figure 2.6. Each plotted point represents $d_{(i,j)}^{start}$ of (i, j) with grey points representing the remaining arcs while colorful points are part of the solution (each different color represents a different route). Also note the black points which represent arcs part of the solution that connect to a charging station, origin depot, or destination depot. This is done because the time windows of these locations span over the whole planning horizon T, which sometimes leads to great differences but is an edge case that cannot be generalized. Therefore, those values should be seen independently from the remaining $d_{(i,j)}^{start}$. Based on this interpretation, it is clearly visible that the difference in earliest service start times of arcs part of the solution follows a certain pattern. This aligns with the idea mentioned earlier that time windows closer together (here the earliest service start times) are favorable and shows that this feature works nicely.

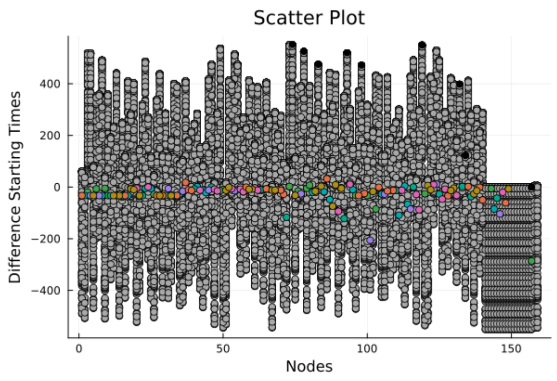


Figure 2.6: Example plot Difference earliest service start times

In fact, every feature concerning time windows analyzed here performed similarly well continuing with:

- **Difference of the latest service start times of locations**: For an arc (i, j) with $i, j \in V$ this is
$$d^{end}_{(i,j)} = w^{\text{end}}_i - w^{\text{end}}_j$$
and looking at the example plot Figure 2.7, again with all instances having an alike plot, a similar result can be seen. Additionally, the interpretation and analysis of the plot also stay the same.

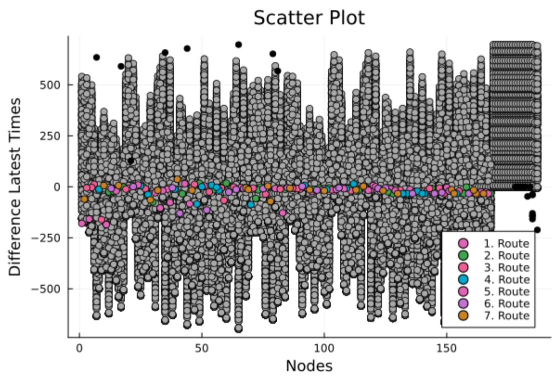


Figure 2.7: Example plot Difference latest service start times

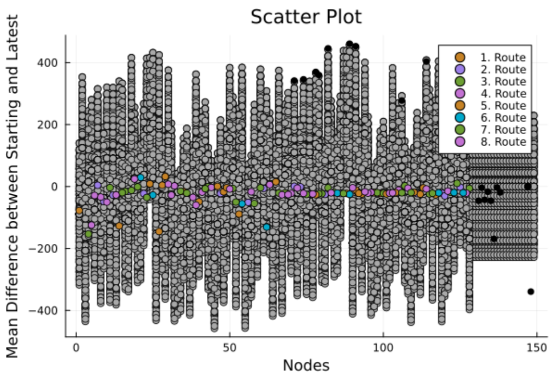
Lastly, the remaining two features analyzed in the time window context where first

- **The mean of both differences**: For an arc (i, j) with $i, j \in V$ this is
$$d^{mean}_{(i,j)} = \frac{d^{start}_{(i,j)} + d^{end}_{(i,j)}}{2}$$
and second,

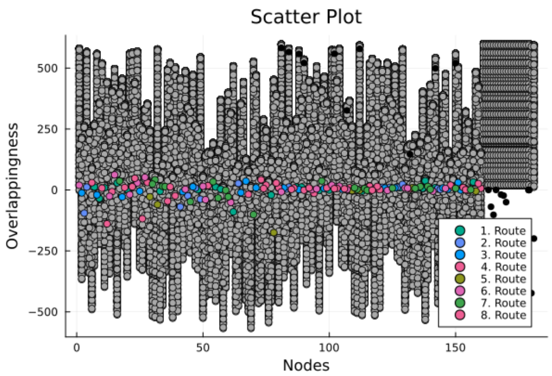
- **Overlappingness** which is here defined as the difference between the latest service time of the first location and the earliest service time of the second location. For an arc (i, j) with $i, j \in V$ this is

$$d^{ov}_{(i,j)} = w^{\text{end}}_i - w^{\text{start}}_j.$$

Now again, both the analysis of $d^{mean}_{(i,j)}$ and $d^{ov}_{(i,j)}$ was equal to the one of $d^{start}_{(i,j)}$ with similar results. In addition to that, two example plots can be seen in Figure 2.8a and Figure 2.8b.



(a) Example plot $d^{mean}_{(i,j)}$



(b) Example plot $d^{ov}_{(i,j)}$

This concludes the data analysis concerning arcs, yet one is not to leave empty-handed. Especially the features about time windows and distance performed well in the initial analysis and in actuality, most features addressed here were also tested during the training of the ML models.

2.5 Single Requests

Following the analysis of arcs, this section will have a closer look at single requests. A request consists of two locations $i \in P$ and $n + i \in D$ and the customer is picked up at i to be dropped off again at $n + i$ within the maximum user ride time. While the study of arcs had a removal of arcs before the execution of the solving algorithm in mind, the idea here is to limit the possible insertion positions of the connected locations. Here, it should be added that each $i \in P$ and $j \in D$ has to be part of a route and also, each location is fixed geographically and in terms of time windows. As a consequence, possible insertion positions are not actually different positions but just different relative to which route the location and at which place in the route the location is added to.

- Specifically, it first was considered if some insertion positions of location $i + n$ can be ruled out beforehand, given an already route assigned location $i \in P$.

- As a second option, it was also analyzed if some insertion positions of location i can be ruled out beforehand, given a route R it should be assigned to.

The *first feature was ruled out* after the first inspection because on average there are only 0.5354 locations between a request pair of locations. With a variance of 0.5285 and a maximum number of locations between pickup and drop-off location pair of 3, there are not enough possible insertion positions left to reduce the number further. *The second feature was ruled out* as well: While initial analysis showed that predicting the correct insertion position of a location i based on w_i^{start}, w_i^{end} and $\mathbf{i} = (i_1, i_2)$ works well, the heuristic process takes too long to deliver a speed-up compared to the original approach. To be exact, for the analysis, each location of a complete route of a solution was removed, heuristically inserted based on the parameters above and lastly the difference between the original and predicted insertion position was calculated. Based on that, the predicted and original positions were on average 0.9788 in the route apart and therefore similar. The heuristic took the vector $\mathbf{i}_{heuristic} = (w_i^{\text{start}}, w_i^{\text{end}}, i_1, i_2)$ of the location i, which was to be inserted, and found location j with vector $\mathbf{j}_{heuristic} = (w_j^{\text{start}}, w_j^{\text{end}}, j_1, j_2)$ closest to $\mathbf{i}_{heuristic}$. The predicted insertion position of i then is the previous position of j. The basis for that heuristic is found in the results of the arc analysis, as there, it was already perceptible that neighboring locations have similar time windows and are rather close together. The sequential nature of the routes in terms of time windows can be also seen in Figure 2.9.

As this simple heuristic already is too slow, this feature was also not further considered for the development of ML models.

2.6 Routes

Finally, in the last section of the data analysis routes will be the point of interest. The ideas of this approach are based on the work of F. Arnold and K. Sörensen, trying to distinguish between good and bad route structures mathematically [AS19]. Further, the goal here is to find structures in the routes of a solution to either be able to assign single requests to a reduced number of routes, or even a single route. This is in contrast to the current approach, which tries all possible routes, and is equivalent to both knowing, how routes “typically” look like, and ensuring, that only routes of this “typical” structure are created by only allowing requests to be added to fitting routes. However, for the current instances, it was not yet possible to find a usable feature or even usable structure in the data which renders this approach unproductive:

- **Geographical location**: First off, the geographical location was assessed with the idea that different routes are distributed in different areas of the instance. However, this is not the case as can be seen in Figure 2.10. The plot represents the geographical grid of the instance with colorful points being part of the route and the black points part of any other route. Consequently, the plot shows the geographical

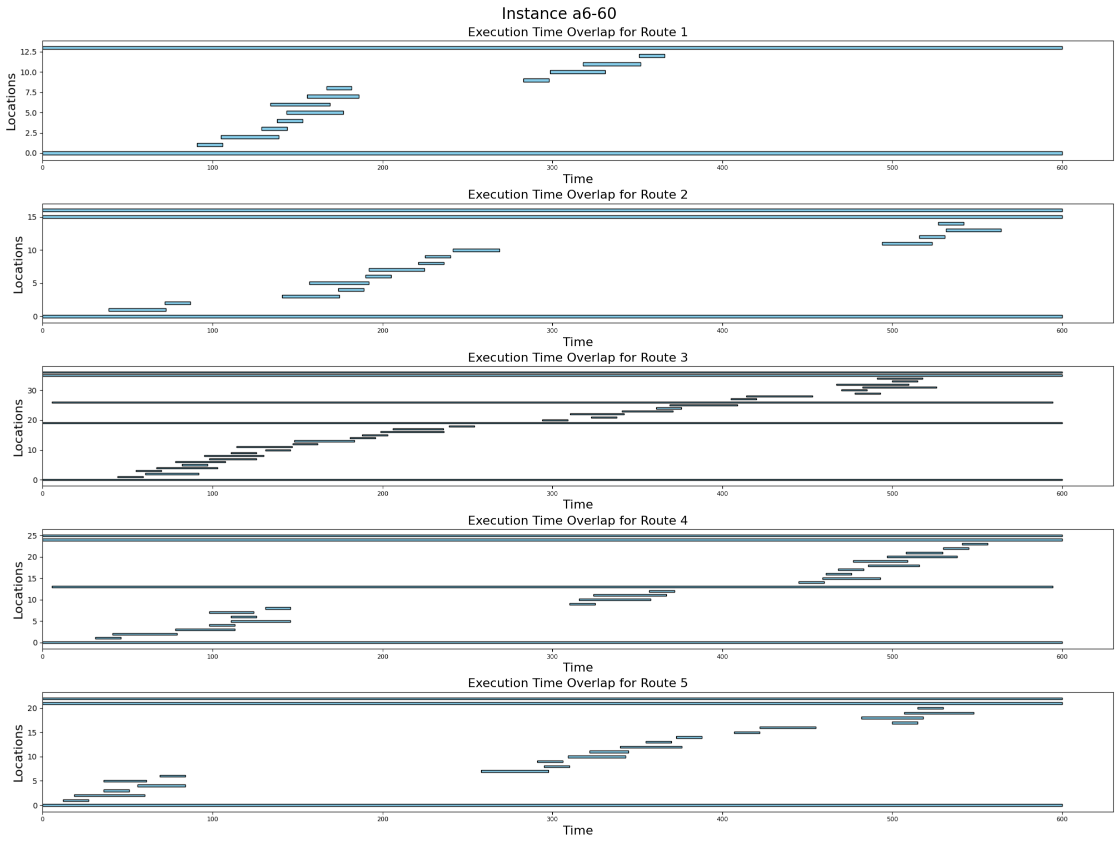


Figure 2.9: Example plot sequential time windows

position of every location of the given instance. Given this interpretation, it is visible, that all routes are mostly distributed through the whole instance which makes this feature alone useless.

- **Time windows**: Next, time windows alone show also little potential. This is because most routes span over the whole time horizon T^{plan} and additionally no other structure that differs between routes was found. How time windows are distributed throughout routes can again be seen in Figure 2.9.

- **Principal Component Analysis**: After the analysis of time windows and geographical locations alone did not bring any interesting results, a combination of these features was considered. This was executed via a Principal Component Analysis (PCA) with four dimensions: w_i^{start}, w_i^{end} and i_1, i_2 for $\mathbf{i} = (i_1, i_2)$. Again, also this combination showed no patterns that would help in the training of an ML model as can be seen in Figure 2.11. The colorful points once more depict locations part of the given route of the solution and the black points are all remaining locations. The four dimensions are reduced to two by just showing the first two components of the PCA.

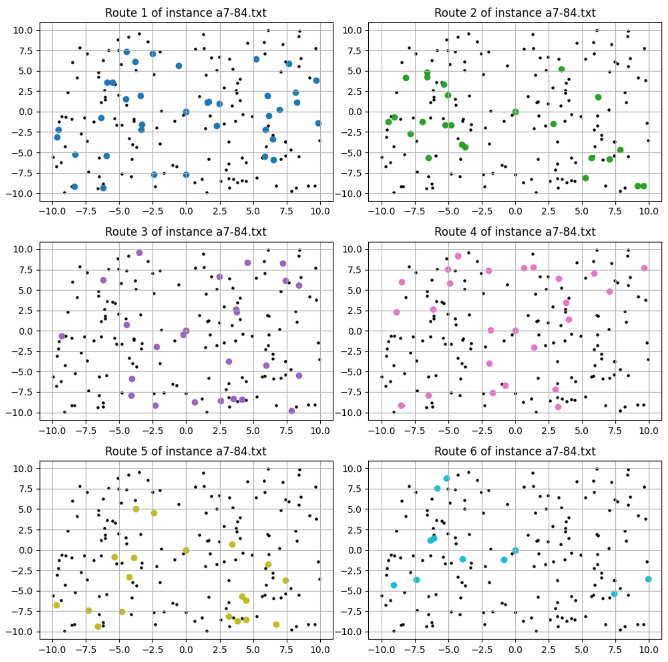


Figure 2.10: Example plot geographical location

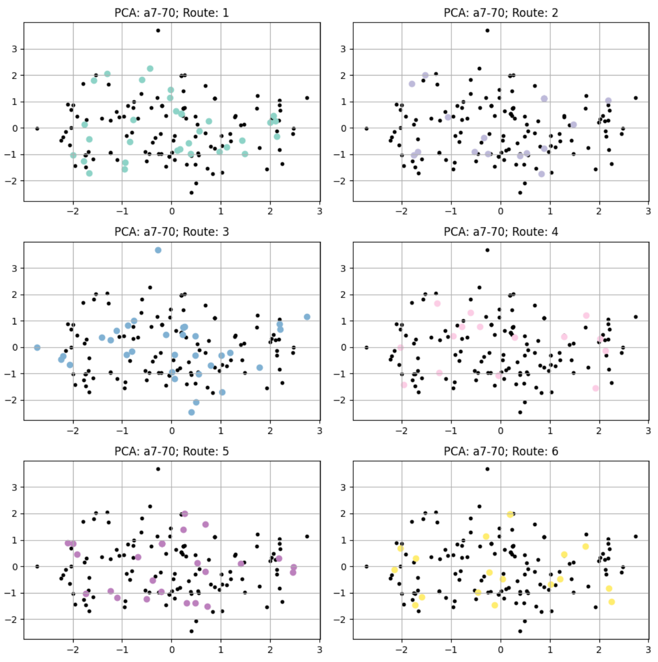


Figure 2.11: Example plot Principal Component Analysis

- **Solutions as a whole**: Finally, for the opportunity to find some other patterns that not yet have been discovered, all routes of the solutions of all instances were visualized. This can be seen in Figure 2.12. For interpretation, each subplot resembles the geographical grid of the same instance and shows a different route of the same solution. Next,
  - each ● blue point is a pickup location,
  - each ● orange point a drop-off location
  - and each ● green point a charging station, origin depot or destination depot.

  In addition to that, each arrow connects two neighboring locations in the route. Therefore, following the arrows, starting with the origin depot, will lead along the route exactly in the same order as the EAV would drive. Lastly, the grey lines connect corresponding pickup- and drop-off locations from the same request. While the plots again confirm the global geographical distribution of routes, they also partly explain why. Looking closer at the structure, this global distribution is mostly caused by the corresponding pickup- and drop-off locations of the same request. These two locations are often in different areas of the instance, so they force the routes to take on this structure. In accordance with that, many of the longest arcs part of the solution are direct connections between the two locations of one request. Even though this opens up new ideas concerning differentiation between these forced longer arcs and the remaining ones, this was not yet analyzed further.

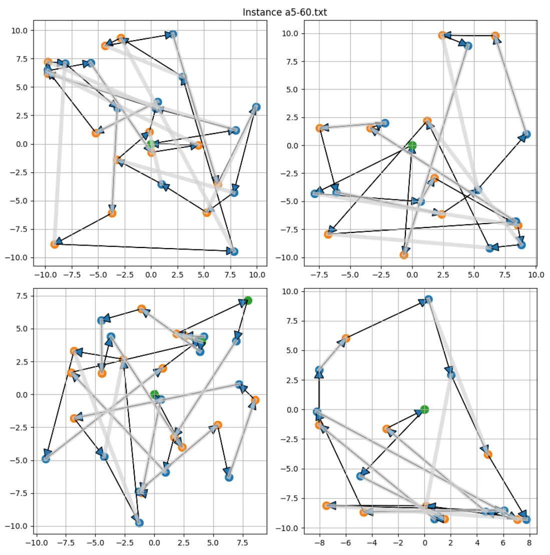


Figure 2.12: Example plot complete route

In conclusion: As of now, no features or patterns were found that could be used in the later process of training ML models.

This wraps up the current standing of the data analysis. In summary, features of arcs especially concerning distance and time windows looked promising while features of routes and single requests showed little potential. *Thus, only the removal of arcs will be further considered and worked with.*

CHAPTER 3

Machine Learning Models

After better understanding the underlying data and extracting possible features, now the basic idea is to take an instance $G = (V, A)$, calculate a feature vector for every arc $(i, j) \in A$ in the graph, remove the worst arcs based on the predictions of the machine learning model, and use this newly created instance for the later solving process. Through that, arcs that would not have been in the solution anyway are never considered and during every iteration of a solving algorithm, several possible, yet not interesting, options for insertion are left out.

Crucially, here it is only considered to strictly classify between including the arc in the further search or not and the task therefore is binary classification.

3.1 Model Choice

First off, for the model choice, four variants of Support Vector Models were analyzed: a non-linear Support Vector Machine (SVM), a Support Vector Regression (SVR), a linear SVM and a linear SVM with a Nystroem transformer.

3.1.1 Motivation

The selection of Support Vector Models is justified by their strong performance in previous, similar work (see, e.g., [FAC23], [AS19]) and their simplicity and speed, which align with the goal of accelerating the process. Since the task involves binary classification, any variant of SVM naturally presents itself as a sensible option.
The specific cases of a linear SVM and a linear SVM with a Nystroem transformer were added later, as the original, non-linear model required too much time for its predictions. While the linear SVM qualifies itself by being the fastest SVM variant, the addition of the Nystroem transformer aims at improving the prediction quality relative to the linear SVM without becoming as slow as the non-linear SVM. In contrast, an SVR cannot be used

directly for classification. Nonetheless, this model was included for two reasons: first, the easier application of the heuristically calculated data, and second, the anticipation that it might be possible to assign a “quality” value to each arc instead of directly classifying it. The first reason will be discussed further in Section 3.3, while the second could form the basis for a probability-oriented pruning strategy, where arcs of lower quality are used less frequently instead of being removed entirely. However, this probability-based approach will not be explored further in this thesis. Consequently, the continuous results of the SVR have to be discretized.

3.1.2 Model Description

With the choice of each model justified, each will now be described in more depth.

- **non-linear SVM**: The non-linear SVM resembles a classic SVM model without additional restrictions. This especially includes unrestricted decisions regarding the kernel and stands in contrast to the linear SVM option. Specifically, the radial basis function (RBF)-, polynomial- and sigmoid kernel were considered as they are the most commonly used. Additionally, for the polynomial kernel, only a degree of 3 was tested[CV95].
- **SVR**: To enable the SVR for classification, a threshold value was used, which classifies all arcs with a predicted value lower than the threshold into the negative class, and those with a higher predicted value into the positive class. For the kernel function, only an RBF kernel was used, as it is the most popular option [DBK+97].
- **linear SVM**: As the simplest SVM variant, the linear SVM assumes that the data is already linearly separable without mapping it into a higher dimensional space. Thus, the linear SVM has no need for a kernel function. [BGV92].
- **linear SVM with Nystroem transformer**: Lastly, as already mentioned, the combined model of a linear SVM and a Nystroem transformer aims to combine the speed of the linear SVM with the improved predictive capabilities of the non-linear SVM (see [WS00] and [YLM+12] for more on the Nystroem transformer). This is achieved using the Nystroem transformer, which projects data points into a higher-dimensional feature space, similar to the kernel function in a non-linear SVM. However, instead of computing the kernel matrix using all data points, it approximates the matrix using a subset of them. After the transformation, the maximum-margin hyperplane separating the two classes of data points can be found. This last step is executed via the linear SVM and happens equally for the non-linear SVM. As a consequence, the combined model can be used as an approximated replacement of the non-linear SVM. Again, only the RBF kernel was examined as the kernel function to be approximated.

Continuing with these selected models, every ML training needs datasets, the creation of which will be examined next. A dataset consists of a number of arcs, assigning each arc both a feature vector, which the ML model uses to predict the class of the arc, and a label, which the ML model uses to evaluate its prediction.

3.2 Features

This section delves into the combination of features used for the training of all earlier-mentioned model options. This starts with the results of the Data Analysis 2 from which following features are taken:

$d(\mathbf{i},\mathbf{j})$	$d_\alpha(\mathbf{i},\mathbf{j})$
$d^{start}_{(i,j)}$	$d^{end}_{(i,j)}$
$d^{mean}_{(i,j)}$	$d^{ov}_{(i,j)}$
$N_5(i)$	$\text{near}_{i,j}$

with $i, j \in V$ and $\mathbf{i} = (i_1, i_2)$ and $\mathbf{j} = (j_1, j_2)$ being the corresponding location vectors. Note that due to the similar nature of $d(\mathbf{i},\mathbf{j})$ and $d_M(\mathbf{i},\mathbf{j})$, $d_M(\mathbf{i},\mathbf{j})$ was not further considered for simplicity reasons and instead $N_5(i)$ and $\text{near}_{i,j}$ were tested in the training as well.

Additionally, for the sake of further exploration, the subsequent features were also implemented.

- **Distance to the closest origin depot**: This is defined for arc (i, j) as

  $$d^{\text{min}}_{\text{origin}}(i,j) = \min_{o \in O} d(\mathbf{mid_{(i,j)}}, \mathbf{o})$$

  with $\mathbf{o} = (o_1, o_2)$ being the location vector of $o \in O$ and $\mathbf{mid_{(i,j)}} = \frac{\mathbf{i+j}}{2}$.

- **Distance to the furthest away origin depot**: This is defined for arc (i, j) as

  $$d^{\text{max}}_{\text{origin}}(i,j) = \max_{o \in O} d(\mathbf{mid_{(i,j)}}, \mathbf{o}).$$

- **Distance to the center of origin depots**: This is defined for arc (i, j) as

  $$d^{\text{center}}_{\text{origin}}(i,j) = d(\mathbf{mid_{(i,j)}}, \mathbf{c_{origin}})$$

  with $\mathbf{c_{origin}} = \left(\frac{\sum_{o \in O} o_1}{n}, \frac{\sum_{o \in O} o_2}{n}\right)$.

- **Distance to the closest destination depot**: This is defined for arc (i, j) as

  $$d^{\text{min}}_{\text{dest}}(i,j) = \min_{d \in D} d(\mathbf{mid_{(i,j)}}, \mathbf{d})$$

  with $\mathbf{d} = (d_1, d_2)$ being the location vector of $d \in D$.

- **Distance to the furthest away destination depot**: This is defined for arc (i, j) as
$$d_{\text{dest}}^{\max}(i, j) = \max_{d \in D} d(\mathbf{mid_{(i,j)}}, \mathbf{d}).$$
- **Distance to the center of destination depots**: This is defined for arc (i, j) as
$$d_{\text{dest}}^{\text{center}}(i, j) = d(\mathbf{mid_{(i,j)}}, \mathbf{c_{dest}})$$
with $\mathbf{c_{dest}} = \left(\frac{\sum_{d \in D} d_1}{n}, \frac{\sum_{d \in D} d_2}{n} \right)$.
- $\mathbf{i} \in \mathbf{S}$: As charging stations play a vital part in planning, the idea was to see if including the information that the arc is connected to a charging station could affect the prediction. Consequently, the subsequent three features check if any location of the arc (i, j) is a charging station.
- $\mathbf{j} \in \mathbf{S}$
- $\mathbf{i}, \mathbf{j} \in \mathbf{S}$

This completes all features used during the training of all ML models. In addition to that, the feature vectors stay the same for all datasets and are calculated for all arcs of the instances noted in 4.1. To complete the datasets the labeling remains.

3.3 Labels

As of now, only the instances, but no solutions, were used to create the datasets. This changes with the labeling process and leads to a central problem. First off, in theory each arc that would be part of any optimal solution S^{opt} should be labeled in the positive class, and else in the negative class, which represents the distinction between arcs of "good" quality and "bad" quality. To be exact, each data point would be assigned a value of 1 or 0 as a label. Note, that there could be more than one optimal solution and the set of arcs in the positive class could be greater than the number of arcs in one optimal solution. However, as already addressed, the complexity of the E-ADARP makes calculating any S^{opt} in a reasonable time for more than five cars infeasible. Consequentially, the solutions to the instances are based on heuristics (e.g. the LNS-heuristic of [BRL24] which is used here) and not optimal. To deal with these "unclean" labels, a weak supervision approach is used: Instead of calculating one solution for a given instance, 100 solutions are calculated. Then, for the SVMs, three possibilities were tested: The data point of an arc (i, j) is assigned a label of 1 if this arc appears at least in *one* of the 100 solutions, *two* of the 100 solutions or *three* of the 100 solutions and else 0. Thus, three distinct datasets were created based on these thresholds, which were then used by the linear SVM, non-linear SVM, and linear SVM with Nystroem transformer. For the SVR, the labels were created by first calculating a weighting ρ_k for each of the 100 solutions of one instance like follows:

$$\rho_k = 1 - \frac{e^{\hat{f}^k_{obj}}}{\sum_{k=1}^{100} e^{\hat{f}^k_{obj}}}$$

with $\hat{f}^k_{obj} = \frac{f^k_{obj}}{min_{l\in\{1,..,100\}} f^l_{obj}}$ and f^k_{obj} being the objective function value (so the solution quality) for the kth-solution of the given instance. Then the label for the data point of arc $(i, j) \in A$ is

$$\gamma_{i,j} = \sum_{k=1}^{100} \rho^k x^k_{i,j}$$

with $x^k_{i,j}$ being the decision variable if the arc (i, j) is part of the solution k. Here see also [FAC23]. This calculation of labels is better at addressing the varying and imperfect quality of solutions showcasing the advantage of the SVR as it works with continuous labels. With this fourth dataset, which is only used by the SVR, finalized and the models defined, it is now time to evaluate their performance.

CHAPTER 4

Experimental Analysis

The machine learning models were implemented and trained in Python 3.11 with scikit-learn and cross-validated with GridSearchCV. The four datasets were created over all arcs of all instances defined in [RCL07] which were then enhanced with E-ADARP-specific features as detailed in [BKG19] resulting in a total of 10 instances which can be seen in Table 4.1.

Table 4.1: Instance from [RCL07]

Instances	**Number of Cars**	**Number of Requests**	**Number of Arcs**
a5-60	5	60	18,496
a6-48	6	48	12,996
a6-60	6	60	19,044
a6-72	6	72	26,244
a7-56	7	56	17,424
a7-70	7	70	25,600
a7-84	7	84	35,344
a8-64	8	64	22,500
a8-80	8	80	33,124
a8-96	8	96	45,796

This leads to 256,568 combined arcs/data points for training and testing over all instances. As most arcs of an instance are not part of a solution all datasets were subsampled after label assignment and the number of data points with positive labels was leveled with the number of data points with negative labels. Following subsampling, each dataset was divided in a uniform random manner, with 80% allocated for training and 20% reserved for testing. Next, the 100 solutions of each instance were calculated on a 13th Gen Intel Core i9-13900H 2.60 GHz with 16.0 GB of RAM based on the LNS of [BRL24] with a maximum of 30000 iterations.

4.1 Parameters

The GridSearch of the SVR tested the regularization parameter C with the values 0.01, 0.1, 1, 10, and 100, the Epsilon insensitive ϵ with the values 0.01, 0.1, 0.5, 1, for the kernel function only the RBF kernel and γ with $\frac{1}{n_{features}}$, $\frac{1}{n_{features}*var(\mathbf{X})}$, 0.001, 0.01, 0.1 and 1 with $n_{features}$ being the number of features and $var(\mathbf{X})$ being the variance of the feature matrix $\mathbf{X}$. As a scoring metric for evaluation, the negative mean squared error is used. For the non-linear SVM the RBF-, polynomial-, and sigmoid kernel functions are tested with possible C values 0.1, 1, 10, and 100. Moreover, for he polynomial- and sgmoid kernel the bias term r was set to 0.0. For γ $\frac{1}{n_{features}}$ and $\frac{1}{n_{features}*var(\mathbf{X})}$ is considered. Further, the linear SVM uses the same C values as the non-linear SVM. Lastly, the linear SVM with Nystroem transformer approximates the RBF kernel with possible γ values of 0.5, 0.75, and 1.0 and a possible number of landmarks of 10, 20, and 30. The linear SVM with Nystroem transformer again uses the same possible C values.

4.2 Comparison of Original and Pruned Variant

To analyze if the pruning of the instance leads to a speed-up, the solutions of the original LNS approach are compared with the solutions of the approach that beforehand prunes the instances with the ML models. In contrast to the testing and evaluation of the ML models themselves, as mentioned earlier, which was conducted only on the instances of [RCL07], the examination of potential speed-up was performed on both the instances in [RCL07] and the larger instances from [Lim23], shown in Table 4.2.

Table 4.2: Instances from [Lim23]

Instances	**Number of Cars**	**Number of Requests**	**Number of Arcs**
a180-3600	180	3600	57,229,225
a200-4000	200	4000	70,644,025
a220-4400	220	4400	85,470,025
a240-4800	240	4800	101,707,225
a260-5200	260	5200	119,355,625

To better adapt the models for different types of instances either the weightings of the positive and negative class for the loss function (for the non-linear SVM and linear SVM with Nystroem transformer) or the threshold value (for the linear SVM and SVR) are adapted and, for each instance, varying options were compared. The usage of these thresholds enables the linear SVM and SVR to only be trained once for each of its possible datasets (see 3.3) and then just be adapted for each instance by changing the threshold. On the contrary, the other models, adapted by weightings, have to be retrained for each weightings combination, which would lead to multiple similar but not identical models. However, looking ahead, the same weightings were used for all instances which were best solved by a SVM that was tuned via weightings and therefore only one model of each variant was needed for the final results. This comparison was executed on single cores of 2.4 GHz Intel Xeon E5-2640 v4 processors with a memory limit of 30 GB. For each weighting combination/threshold value of each model for each instance, 20 solutions were computed while for the original approach for each instance, 50 solutions were computed. In addition to that, the solving process of the smaller instances was set to stop after 30000 iterations while the solving process of the bigger instances stopped after 900 seconds.

With the experimental environment set, the next step is to sum up the results.

CHAPTER 5

Results

This chapter Results is split into two parts. The first outlines the performance of the tested ML models by themselves and the second reviews if and how much speed-up the usage of these ML models delivers.

5.1 Machine Learning Models

While an interesting feature α-Nearness was simply to computationally expensive for further testing and had to be left out.

5.1.1 Non-linear SVM

The best SVM model was based on an RBF kernel with a C value of 100, γ of 0.0001, and dataset based on an arc appearing at least once over all 100 solutions. The best combination of features was $d(\mathbf{i},\mathbf{j})$, $d^{start}_{(i,j)}$, $d^{end}_{(i,j)}$, $d^{mean}_{(i,j)}$, $d^{ov}_{(i,j)}$, $N_5(i)$ and $\text{near}_{i,j}$. With these settings, the model performed quite well with an accuracy of 0.892, and additionally the confusion matrix and a classification report can be seen in Table 5.1 and Table 5.2. The quality of the non-linear SVM is also visible in its ROC curve in Figure 5.1. Note, that class “1” is the positive class and represents arcs that were part of the solution while “0” is the negative class and represents arcs not part of the solution.

Table 5.1: Confusion Matrix non-linear SVM

	Predicted 0	Predicted 1
Actual 0	696	136
Actual 1	42	775

Table 5.2: Classification Report non-linear SVM

Class	Precision	Recall	F1-score	Support
0.0	0.94	0.84	0.89	832
1.0	0.85	0.95	0.90	817
Accuracy			0.89	
Macro avg	0.90	0.89	0.89	1649
Weighted avg	0.90	0.89	0.89	1649

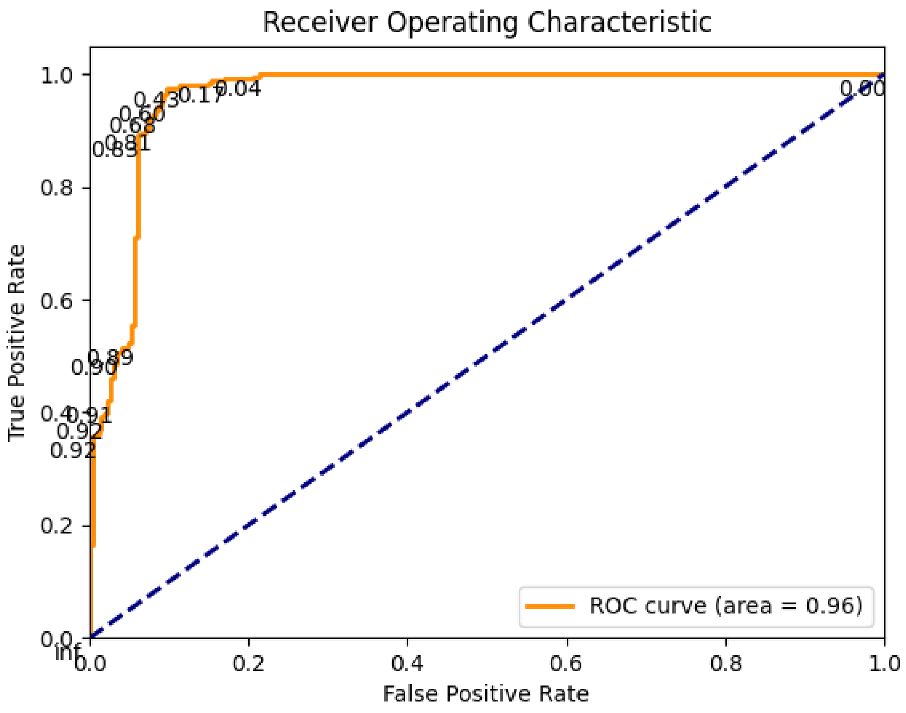


Figure 5.1: ROC curve of the non-linear SVM

5.1.2 SVR

The best SVR model had a coefficient of determination r^2 of 0.220, a mean squared error of 496.939 with a C value of 100, ϵ of 1 and γ of 1 and therefore did not really perform up to expectation. The best combination of features was $d(\mathbf{i},\mathbf{j})$, $d^{start}_{(i,j)}$, $d^{end}_{(i,j)}$, $d^{mean}_{(i,j)}$, $d^{ov}_{(i,j)}$ and near$_{i,j}$. The performance can also be seen in Figure 5.2 with heteroscedasticity and patterns indicating the low quality of the model.

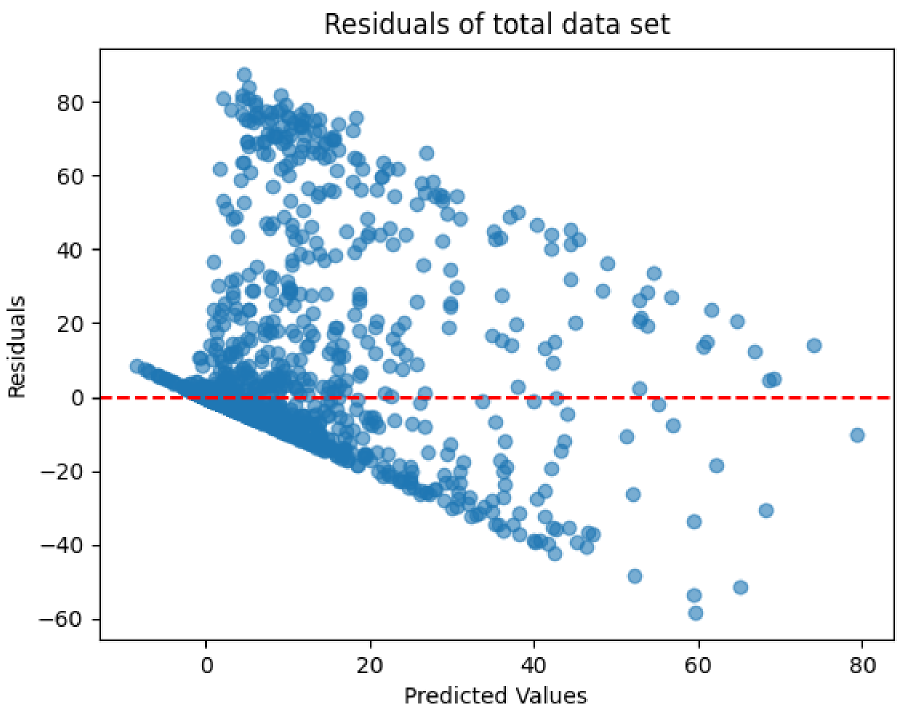


Figure 5.2: Residuals plot of the SVR

5.1.3 Linear SVM

The best linear SVM model used a C value of 1 and the dataset based on an arc appearing at least once over all 100 solutions, achieving an accuracy of 0.704 with the feature combination $d(\mathbf{i},\mathbf{j})$, $d^{start}_{(i,j)}$, $d^{end}_{(i,j)}$, $d^{mean}_{(i,j)}$, $d^{ov}_{(i,j)}$ and near$_{i,j}$. Equal to the non-linear SVM, the confusion matrix and a classification report can be seen in Table 5.3 and Table 5.4, and the ROC curve in Figure 5.3. However, the accuracy barely decreases to 0.681 if only $d(\mathbf{i},\mathbf{j})$ and $d^{mean}_{(i,j)}$ are used. This is crucial for the big instances as here even a linear SVM would take too long for prediction using a big feature vector.

Table 5.3: Confusion Matrix linear SVM

	Predicted 0	Predicted 1
Actual 0	560	272
Actual 1	216	601

Table 5.4: Classification Report linear SVM

Class	Precision	Recall	F1-score	Support
0.0	0.72	0.67	0.70	832
1.0	0.69	0.74	0.71	817
Accuracy		0.704		
Macro avg	0.71	0.70	0.70	1649
Weighted avg	0.71	0.70	0.70	1649

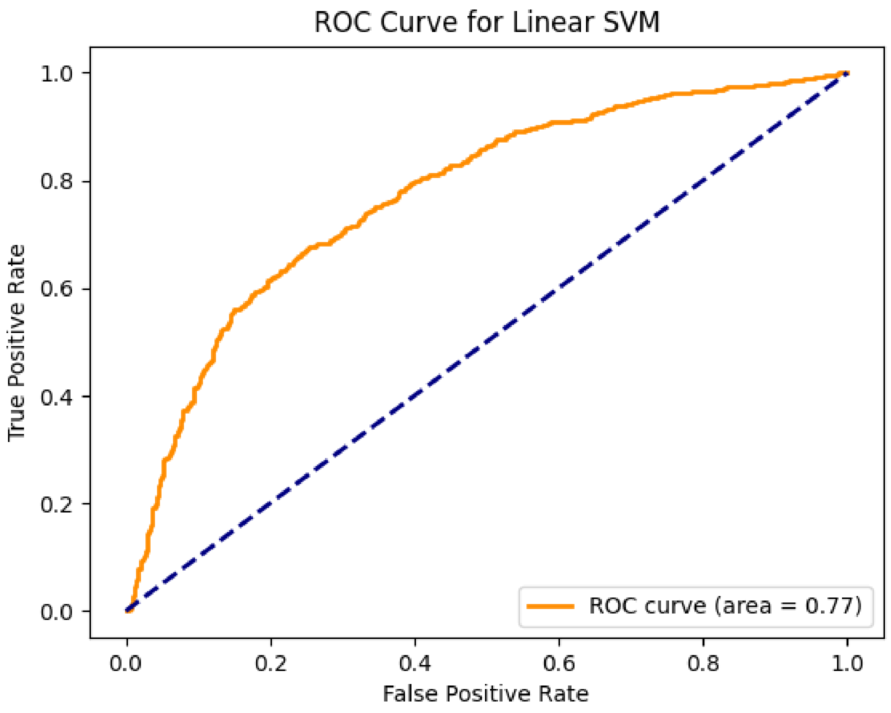


Figure 5.3: ROC curve of the linear SVM

5.1.4 Linear SVM with Nystroem Transformer

For the last model the best parameters were a γ of 0.5 with 30 landmarks and a C value of 100. Combined with the features $d(\mathbf{i},\mathbf{j})$, $d^{start}_{(i,j)}$, $d^{end}_{(i,j)}$, $d^{mean}_{(i,j)}$, $d^{ov}_{(i,j)}$ and near$_{i,j}$ and, once more, the dataset based on an arc appearing at least once over all 100 solutions the accuracy reached 0.885. More metrics can be found in Table 5.5 and Table 5.6 and the corresponding ROC curve can be seen in Figure 5.4. Similar to the linear SVM the performance is comparable when using only $d(\mathbf{i},\mathbf{j})$ and $d^{mean}_{(i,j)}$ with an accuracy of 0.875.

Table 5.5: Confusion Matrix linear SVM w. Nystroem

	Predicted 0	Predicted 1
Actual 0	696	136
Actual 1	54	763

Table 5.6: Classification Report linear SVM w. Nystroem

Class	Precision	Recall	F1-score	Support
0.0	0.93	0.84	0.88	832
1.0	0.85	0.93	0.89	817
Accuracy		0.88		
Macro avg	0.89	0.89	0.88	1649
Weighted avg	0.89	0.88	0.88	1649

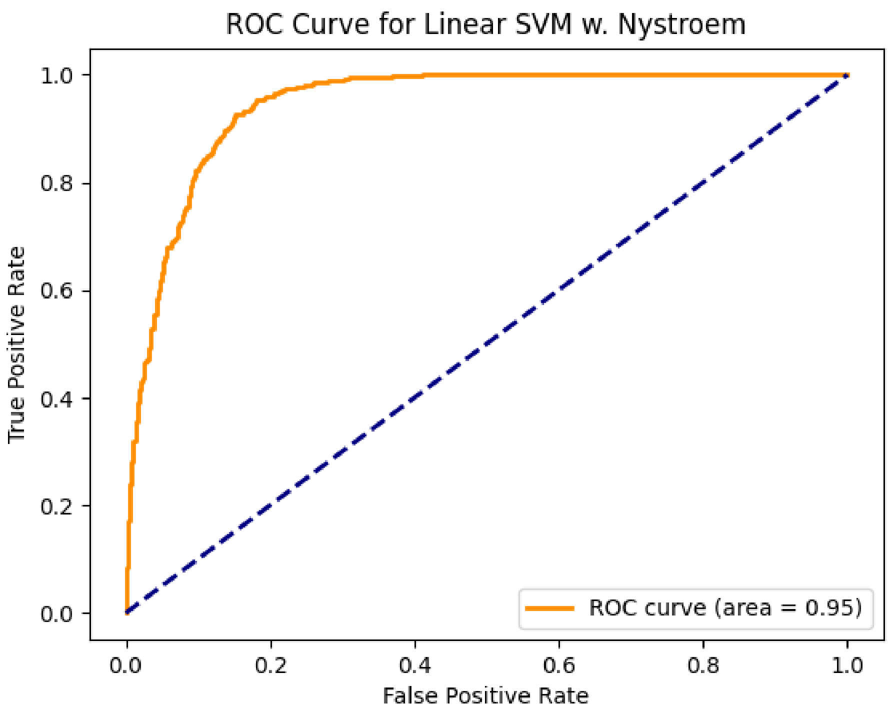


Figure 5.4: ROC curve of the linear SVM w. Nystroem

In conclusion, the non-linear SVM and linear SVM with the Nystroem transformer performed well with similar results while the non-linear SVM was slightly better. The simplification of the linear SVM also reduced the predictive quality while not completely removing it and lastly, the SVR did not show any potential.

5.2 Speed-up

Based on these results, this section analyzes how the run time and solution quality change, when the models are used to prune the instance before the execution of the search algorithm.

First off, the SVR was briefly tested, however only on the smaller instances form [RCL07]. For the threshold values 0.5, 0.7, and 0.8 were tested which is equivalent to removing 50, 30 and 20 percent of arcs. After the pruning process, the LNS was unable to find a solution for four out of the nine tested instances, yet the run time of the still solved instances was, on average, reduced by 23 percent while the objective function value on average increased by about 14.6 percent in comparison to the original approach. Hence, for the instances that stay feasible, the process of pruning does speed-up the search for the best solution considerably, in comparison to the approach without pruning, while losing a percentage-based smaller amount of solution quality.

Continuing with the non-linear SVM, this model was simply too computationally inefficient for the bigger instances, taking over an hour to predict all arcs for each instance in 4.2. Additionally, the non-linear SVM did not bring any improvements in solution quality for the smaller instances compared to the linear SVM with the Nystroem transformer, yet took longer for the prediction process. As a consequence, the non-linear SVM is simply inferior to the linear SVM with the Nystroem transformer in the context of this thesis.

The speed-up results of the linear SVM were also outperformed by the linear SVM with the Nystroem transformer for the smaller instances, however, surprisingly the linear SVM performed best for the bigger instances. The exact numbers can be seen in Table 5.7 which shows the objective values of the original and pruning based approach together with the absolute and percentage-based differences. Moreover, for each instance the best threshold is listed. As the threshold is originally 0, which would represent a point being on the decision hyperplane of the linear SVM, values under 0 here increase positively classified arcs while values over 0 increase negatively classified arcs. As visible, the pruned variant outperforms the original approach through out all big instances in the same time frame showcasing the potential of this new ML-based pruning approach. It should also be mentioned, that these results were achieved using the already addressed simplified version of the linear SVM using only the two features $d(\mathbf{i},\mathbf{j})$ and $d^{mean}_{(i,j)}$.

Table 5.7: Comparison between Original Best Obj and Pruned Best Obj

Statistic	**Original**	**Pruned**	**Difference**	**% Diff.**
a180-3600	**Threshold:**	**-0.0375**		
Mean	34266.974	29534.439	-4732.536	-13.81%

Statistic	**Original**	**Pruned**	**Difference**	**% Diff.**
Median	33910.605	29502.244	-4408.361	-13.00%
Variance	1091591.561	18278.221	-1073313.340	-98.33%
a200-4000	**Threshold:**	**0.0375**		
Mean	38266.942	32612.527	-5654.415	-14.78%
Median	38138.933	32598.412	-5540.521	-14.53%
Variance	730781.581	22741.802	-708039.779	-96.89%
a220-4400	**Threshold:**	**0.075**		
Mean	42196.016	35830.309	-6365.707	-15.09%
Median	42254.584	35808.616	-6445.968	-15.26%
Variance	312921.066	15177.805	-297743.261	-95.15%
a240-4800	**Threshold:**	**0.1**		
Mean	46621.734	39067.042	-7554.692	-16.20%
Median	46733.811	39043.950	-7689.860	-16.45%
Variance	136119.368	37917.011	-98202.357	-72.14%
a260-5200	**Threshold:**	**0.175**		
Mean	50453.999	42340.159	-8113.839	-16.08%
Median	50459.760	42321.612	-8138.148	-16.13%
Variance	49800.225	30185.222	-19615.003	-39.39%

Lastly, the linear SVM with the Nystroem transformer performed best for the smaller instances. Using the pruning based approach the mean objective function values increased by a maximum of four percent while the mean run time improved by at least 35 percent. The exact results can be seen in Table 5.8, Table 5.9 and are also exemplarily visualized in Figure 5.5. The used weighting for all instances was 1 for the positive class and 0.5 for the negative class.

Table 5.8: Comparison between Original Best Obj and Pruned Best Obj Across Instances

Instance	**Statistic**	**Original**	**Pruned**	**Difference**	**% Diff.**
a5-60	Mean	697.882	718.499	20.618	2.95%
	Median	699.068	718.277	19.210	2.75%
	Variance	14.360	30.591	16.232	113.04%
a6-48	Mean	511.412	522.113	10.701	2.09%
	Median	509.890	521.429	11.539	2.26%
	Variance	10.363	3.601	-6.762	-65.25%
a6-60	Mean	695.909	710.444	14.535	2.09%
	Median	695.700	709.986	14.286	2.05%
	Variance	8.463	13.027	4.565	53.94%

Instance	Statistic	Original	Pruned	Difference	% Diff.
a6-72	Mean	785.194	823.949	38.755	4.94%
	Median	784.997	823.402	38.405	4.89%
	Variance	30.040	36.833	6.793	22.61%
a7-56	Mean	625.040	635.738	10.698	1.71%
	Median	624.386	635.036	10.650	1.71%
	Variance	14.794	31.528	16.734	113.12%
a7-70	Mean	769.354	786.104	16.750	2.18%
	Median	769.311	786.517	17.206	2.24%
	Variance	26.811	28.781	1.970	7.35%
a7-84	Mean	901.949	938.440	36.491	4.05%
	Median	899.968	938.735	38.766	4.31%
	Variance	36.676	29.949	-6.727	18.34%
a8-64	Mean	645.669	661.534	15.865	2.46%
	Median	644.395	659.900	15.505	2.41%
	Variance	21.085	62.075	40.990	194.40%
a8-80	Mean	820.359	840.384	20.026	2.44%
	Median	819.674	840.296	20.623	2.52%
	Variance	29.863	27.092	-2.770	-9.28%
a8-96	Mean	1067.734	1099.629	31.895	2.99%
	Median	1067.735	1097.854	30.119	2.82%
	Variance	52.837	63.174	10.337	19.56%

Table 5.9: Comparison between Original Total Time and Pruned Total Time Across Instances

Instance	Statistic	Original	Pruned	Difference	% Diff.
a5-60	Mean	29.800	19.086	-10.714	-35.95%
	Median	29.757	19.327	-10.431	-35.05%
	Variance	1.714	0.931	-0.783	-45.70%
a6-48	Mean	27.454	16.254	-11.201	-40.80%
	Median	27.698	16.202	-11.496	-41.51%
	Variance	1.370	0.744	-0.626	-45.70%
a6-60	Mean	27.238	16.554	-10.683	-39.22%
	Median	27.289	16.442	-10.847	-39.75%
	Variance	1.248	0.737	-0.511	-40.97%

Instance	Statistic	Original	Pruned	Difference	% Diff.
a6-72	Mean	36.508	22.275	-14.234	-38.99%
	Median	36.721	22.497	-14.224	-38.73%
	Variance	2.494	1.364	-1.130	-45.31%
a7-56	Mean	27.590	16.481	-11.109	-40.26%
	Median	27.577	16.418	-11.159	-40.46%
	Variance	1.320	0.481	-0.839	-63.55%
a7-70	Mean	33.022	20.604	-12.418	-37.61%
	Median	33.241	20.551	-12.691	-38.18%
	Variance	2.278	0.862	-1.416	-62.14%
a7-84	Mean	39.660	24.004	-15.656	-39.48%
	Median	40.039	24.025	-16.014	-40.00%
	Variance	2.652	1.084	-1.568	-59.12%
a8-64	Mean	34.922	21.222	-13.700	-39.23%
	Median	34.990	21.394	-13.596	-38.86%
	Variance	2.036	1.662	-0.375	-18.41%
a8-80	Mean	40.479	23.679	-16.799	-41.50%
	Median	40.696	23.812	-16.884	-41.49%
	Variance	2.025	0.858	-1.167	-57.61%
a8-96	Mean	46.010	27.042	-18.968	-41.23%
	Median	46.137	26.662	-19.476	-42.21%
	Variance	3.672	1.128	-2.544	-69.27%

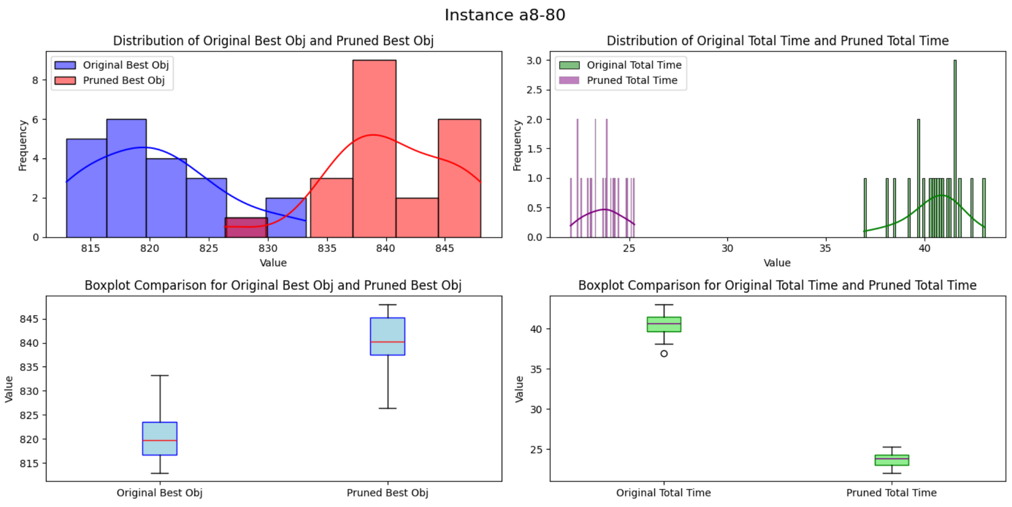


Figure 5.5: Results of smaller instances

CHAPTER 6

Conclusion

In conclusion, removing arcs with an ML model from a given E-ADARP instance before executing a solving algorithm on the instance heavily improves run time and can even improve the solution quality for growing instance size. Starting with the data analysis, which uncovered several potent features, a necessary basis for the development of ML models was set. In the following chapter, four options for the ML model were investigated and both labels and features were defined. Based on this raw build, an experimental environment to test these options was set and in the last chapter, both the performance of the models on their own and their contribution to the speed-up of the solving of the E-ADARP were put forward. Now, based on these results, it can be said that machine learning can speed-up route planning in E-ADARP by pruning the search space before searching for the best solution using LNS (large neighbourhood search).

CHAPTER 7

Future Work

As for the future, this ML-based approach still holds a lot of potential. An interesting idea is the already mentioned exchange of the hard binary classification process towards a probability-based approach, that also sometimes allows edges of low quality to be selected during the construction of a solution. Further, while the structure of routes was shown to be complicated in the data analysis, there still arose some patterns, like most long arcs being forced by far-apart locations of the same request. Diving deeper into the data and making these patterns usable, could enable the training of new ML models, that determine if a request is compatible with a whole route. Especially with growing route length, this could lead to great time savings. Finally, the usage of ML in parts of algorithms does not have to be reduced to the pruning beforehand. Other possibilities like the selection of destroy and repair operators for the LNS can be just as powerful.

List of Figures

List of Tables

Bibliography

[AGS19] Florian Arnold, Michel Gendreau, and Kenneth Sörensen. Efficiently solving very large-scale routing problems. *Computers Operations Research*, 107:32–42, 2019.

[AS19] Florian Arnold and Kenneth Sörensen. What makes a VRP solution good? the generation of problem-specific knowledge for heuristics. *Computers Operations Research*, 106:280–288, 2019.

[BGV92] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery.

[BKG19] Claudia Bongiovanni, Mor Kaspi, and Nikolas Geroliminis. The electric autonomous dial-a-ride problem. *Transportation Research Part B: Methodological*, 122:436–456, 2019.

[BRL24] Maria Bresich, Günther R. Raidl, and Steffen Limmer. Letting a large neighborhood search for an electric dial-a-ride problem fly: On-the-fly charging station insertion. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '24, page 142–150, New York, NY, USA, 2024. Association for Computing Machinery.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[DBK$^+$97] Harris Drucker, Christopher Burges, Linda Kaufman, Alexander Smola, and V. Vapnik. Support vector regression machines. *Adv Neural Inform Process Syst*, 28:779–784, 01 1997.

[FAC23] James Fitzpatrick, Deepak Ajwani, and Paula Carroll. Learning to prune electric vehicle routing problems. In Meinolf Sellmann and Kevin Tierney, editors, *Learning and Intelligent Optimization*, pages 378–392, Cham, 2023. Springer International Publishing.

[HK70] Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

[Lim23] Steffen Limmer. Bilevel large neighborhood search for the electric autonomous dial-a-ride problem. *Transportation Research Interdisciplinary Perspectives*, 21:100876, 2023.

[RCL07] Stefan Ropke, Jean-François Cordeau, and Gilbert Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[SDP23] Yue Su, Nicolas Dupin, and Jakob Puchinger. A deterministic annealing local search for the electric autonomous dial-a-ride problem. *European Journal of Operational Research*, 309(3):1091–1111, 2023.

[WS00] Christopher K. I. Williams and Matthias Seeger. Using the nyström method to speed up kernel machines. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, page 661–667, Cambridge, MA, USA, 2000. MIT Press.

[YLM+12] Tianbao Yang, Yu-Feng Li, Mehrdad Mahdavi, Rong Jin, and Zhi-Hua Zhou. Nyström method vs random fourier features: a theoretical and empirical comparison. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 476–484, Red Hook, NY, USA, 2012. Curran Associates Inc.