Estimating Path Lengths in Traveling Salesman Images

Maxime Drouhin
Department of Business Informatics
Hochschule Karlsruhe
Karlsruhe, Germany
mdrouhin2+hka@gmail.com

Randolf Nkimo
Department of Business Informatics
Hochschule Karlsruhe
Karlsruhe, Germany
nkra1011@h-ka.de

Tobias Frey
Faculty of Electrical Engineering and Information Technology
Hochschule Karlsruhe
Karlsruhe, Germany
frto1020@h-ka.de

***Abstract*—- Objective: Estimate path lengths from TSP images. [1] - Methods: Machine learning with feature engineering and an algorithmic approach. - Findings: Algorithmic method yielded better results in terms of accuracy.**

***Index Terms*—Traveling Salesman Problem, Path Length Estimation, Machine Learning, Algorithmic Approach, Image Processing**

I. Introduction

- Introduce the Traveling Salesman Problem (TSP).
- Explain the importance and applications of TSP.
- State the specific challenge: Estimating path length from images.
- Brief overview of the data provided in the competition.

II. Problem Statement

- Goal: Estimating total path length from images.
- Dataset: Images with cities and paths labeled.
- Challenges: Handling image processing and distance measurement.
- Variability: Different resolutions, image dimensions, and path complexities.

III. Machine Learning Approach

A. Methodology

1) Feature Extraction: The feature extraction process employed several algorithms to derive meaningful attributes from the raw data.

- *Edge Detection*
  We use the Canny edge detection algorithm to identify edges in an image. The process begins with Gaussian smoothing to reduce noise. Next, intensity gradients are calculated, and non-maximum gradient values are suppressed to thin the edges. Double thresholding is then applied to differentiate between strong and weak edges. By detecting the contours using the Canny algorithm, we can calculate the total length of these contours, which serves as our primary feature. This method is especially effective for images with a few distinct or well-defined paths.
  This approach enhances the feature extraction process by providing a quantitative measure of the edge lengths in the image, as illustrated in Figure 1.

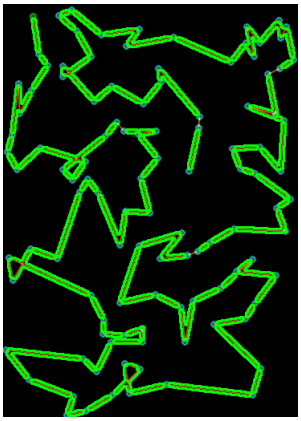

Fig. 1. Edge detection (in green)

- *Count the number of blue pixels*
  In this approach, we count the number of blue pixels in each image, which correspond to the movement lines depicted in the images. By taking the total number of blue pixels and dividing it by the width of the image, we can approximate the total length of these lines. This process involves several steps:
  1) *Thresholding:* The image is thresholded to retain only the blue pixels, isolating the lines of movement.
  2) *Width Analysis:* We analyze representative images to approximate the width of the lines.
  3) *Length Approximation:* By dividing the number of

blue pixels by the estimated width of the lines, we obtain an initial feature representing the total length of movement.

This method leverages color-based segmentation and geometric analysis to extract a feature that quantifies the extent of movement within the images, which we employ as a secondary feature.

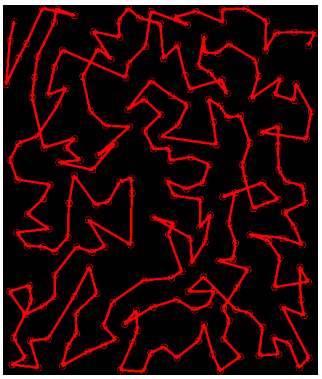

Fig. 2. Blue pixels count (in red)

2) Model Implementation: The study employs three machine learning models: Random Forest, Gradient Boosting Machine (GBM), and Neural Network (MLPRegressor).

1) For the *Random Forest* model, the parameters used are: `max_depth=10`, `min_samples_split=10`, `n_estimators=200`.
2) For the *Gradient Boosting Machine (GBM)*, the specified parameters are: `n_estimators=100`, `learning_rate=0.1`, `max_depth=3`.
3) For the *Neural Network (MLPRegressor)*, the key parameters include: `hidden_layer_sizes=(100,)`, `activation='relu'`, `solver='adam'`, and `max_iter=500`.

B. Results

1) *Performance Metrics:* RMSE, R^2 values for different models.

Model	RMSE	R^2
Random Forest	22872.88	0.15
Gradient Boosting Machine (GBM)	22997.06	0.14
Neural Network (MLPRegressor)	26600.62	-0.14

TABLE I
PERFORMANCE METRICS FOR DIFFERENT MODELS

The table presents the performance metrics (RMSE and R^2) for three models: Random Forest (RF), Gradient Boosting Machine (GBM), and Neural Network (MLPRegressor). The RF has an RMSE of 22872.88 and an R^2 of 0.15, the GBM has an RMSE of 22997.06 and an R^2 of 0.14, and the MLP has an RMSE of 26600.62 and an R^2 of -0.14. The RF and GBM show similar performance, with a slight advantage for the RF. The MLP, with a higher RMSE and a negative R^2, is significantly less performant.

2) *Error Analysis:* Distribution of errors, outliers.

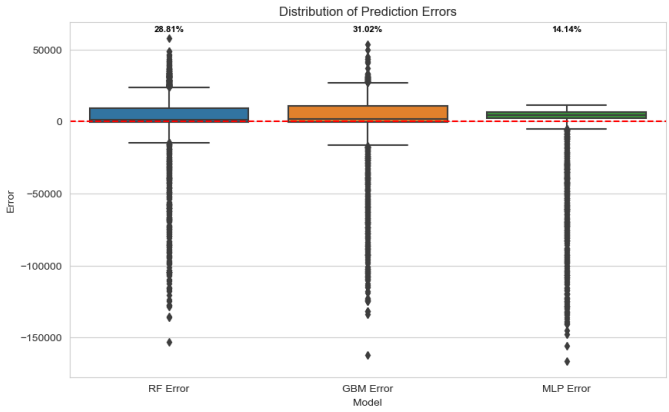

Fig. 3. Error Analysis: Distribution of errors, outliers

The graph shows the distribution of prediction errors for three models: Random Forest (RF), Gradient Boosting Machine (GBM), and Multi-Layer Perceptron (MLP). The outlier rates are 26.50 % for RF, 30.06% for GBM, and 14.08% for MLP. The MLP demonstrates the lowest variability in errors with a median close to zero and the lowest outlier rate. However, despite having the lowest outlier rate, the MLP's errors are more significant on average, leading to a higher RMSE. This means that the MLP tends to make more consistent errors, but when it errs, the errors are larger, which negatively impacts its RMSE. In contrast, GBM has the highest variability and the highest outlier rate, indicating less stable performance. RF is intermediate, with moderate variability and outlier rate, and an RMSE lower than the MLP.

Therefore, although the MLP has a lower outlier rate, it is less performant overall due to its higher RMSE. The RF is the best model, balancing error variability and overall performance effectively.

Unfortunately, these machine learning models do not yield satisfactory results notably on images containing complex trajectories. This limitation prompted us to explore algorithmic approaches to address the issue.

IV. Algorithmic Approach

A. Methodology

Since these images are all very similar in structure to each other and contain easily distinguishable shapes and colors, an algorithmic approach came, to mind when first inspecting the problem. We decided to give it a try. After further consideration the general approach for the algorithmic solution was separated into five parts:

- Find the position of all points
- Get the position of the starting point
- Score the connection between all paths
- Find the optimal path to connect all points
- Calculate the length along the path

For the first part the open-CV library is used. The image is first filtered for the correct color range using the `inRange` function. Then the circle detection is made by `findContours`. The centroid of each contour is calculated with the moments function. The starting position is calculated by getting the mean position of all green pixels in the image. For the scoring of the connection multiple approaches where tried. In the end a simple and fast approach of counting the number of black pixels in a straight line between the two points and dividing by the length of the line was used. This results in a metric where a connection is scored between zero and one with lower scores representing a better connection. In order to evaluate the functionality of the above mentioned functions additional data was generated, where the positions and order of points was known. With this it could be checked if the functions returned the center of points and if connected points got low connection-scores. With this generated data it was also planned to train machine learning models to take over finding the points and/or scoring the connections. However this was not realised due to time constraints. So the generated data was not used afterwards. Continuing with the algorithmic approach, the next step is to find the optimal thru the points. To do this multiple functions were evaluated. The first and simplest approach is always follow the lowest remaining path. This approach is commonly known as *'greedy'*. It will hereinafter be referred to as *'Simple'*. Another approach is the use of the `heuristics.solve_tsp_simulated_annealing` function of the `python_tsp` library [3]. The time for the function to solve the path is limited to 60 seconds. This approach will hereinafter be referred to as *'TSP'*. Lastly multiple custom approaches were tried. Since the problem is O(n!) a brute force approach is not viable. The shortest path also isn't on a length on a 2D plane but instead on arbitrary scores between the points. As such usual heuristics are not effective. The custom algorithm works by making an assumption about the worst path score possible in the correct path. It then tries to create a path using only the remaining paths. With this assumption it can terminate paths early if they leave a point with no valid connections or more than two points with one connection. This speed the search up enormously. Once any valid path is found it will be returned. If it is impossible to make a path or the search takes to long the assumption about the worst path score is corrected. If, even after multiple tries, the algorithm doesn't succeed it is terminated for good since the problem then returns to be O(n!). In this case the algorithm returns that no path is detected. This was the case for 40% of the images. The custom approach will hereinafter be referred to as *'Valid'*. A variation of the algorithm was also tested. In the variation the minimum connection-score of a point is subtracted from all other connection-scores of the point. This was done in an attempt to reduce the unevenness of the connection-scores between badly and accurately detected point centers. This variation will hereinafter be referred to as *'Valid_zero'*. All of the above mentioned algorithms (if successful) return an order for the detected points. To get the length of the path the points are ordered accordingly and the euclidean distances along the path are summed up.

RMSE	simple	tsp	valid	valid zero	AI
replaced	6768.0	4707.0	4364.6	5234.3	6427.8
removed	6992.7	4378.3	3663.5	3640.9	4936.6

TABLE II
RMSE OF ERROR

RMSE	simple	tsp	valid	valid zero	AI
replaced	0.2361	0.2725	0.1997	0.2026	0.3922
removed	0.1734	0.2383	0.0406	0.0396	0.4806

TABLE III
RMSE OF NORMALIZED ERROR

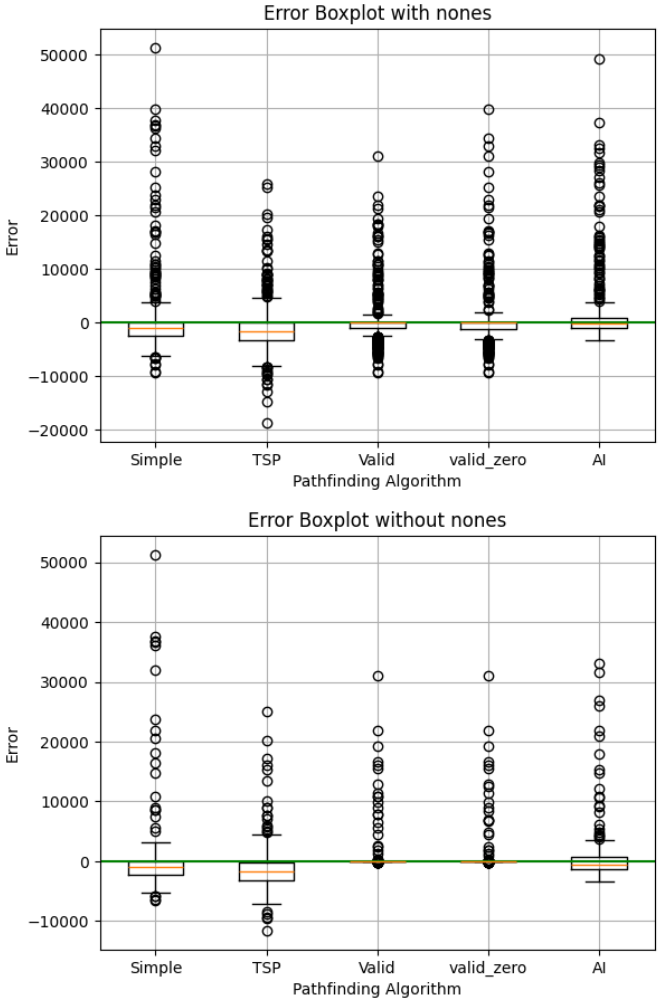

Fig. 4. Error on the algorithmic approach

B. Results

In order to evaluate the performance a baseline is useful. The kaggle challenge [1] doesn't have a subjected solutions. However there is code provided by the Author [2]. After

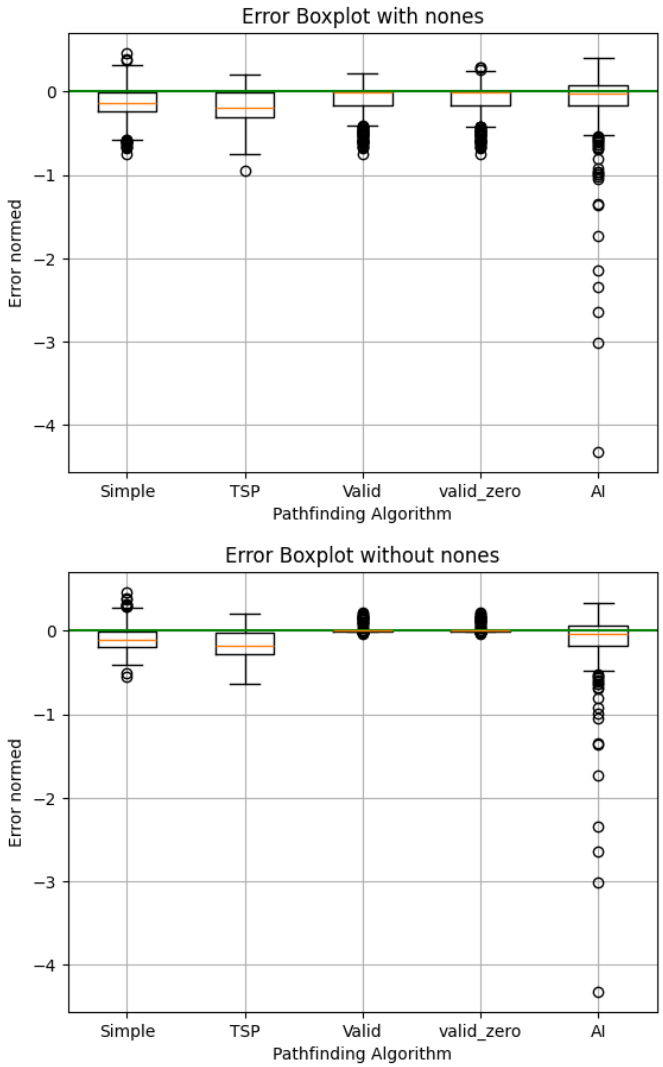

Fig. 5. Normalized error on the algorithmic approach

running the code the resulting model is used for comparison and will hereinafter be referred to as *'AI'*. All referenced approaches were used on the same images. Due to the long time to compute the algorithmic approaches 511 images were evaluated. The *'valid'* approach did only produce a path for 298 images. The *'valid_zero'* approach only for 261 images. Images where the approach resulted in no valid path where either removed or the resulting path was replaced by the path produced by the 'simple' approach. In Fig. 4 the absolute error over the different approaches is plotted. In table II the root mean square error is shown. Both are shown with replaced nones and with removed nones. To better show the error relative to the true path length, both the plot (5) and table (III are also shown normalized. To normalize the results the formula $\frac{true\_length - detected\_length}{true\_length}$ was used.

The fastest approach is the *'AI'* with an mean time of 0.2 seconds. The *'simple'* approach is the next fastest, with an mean time of 3.5 seconds. The other approaches have times

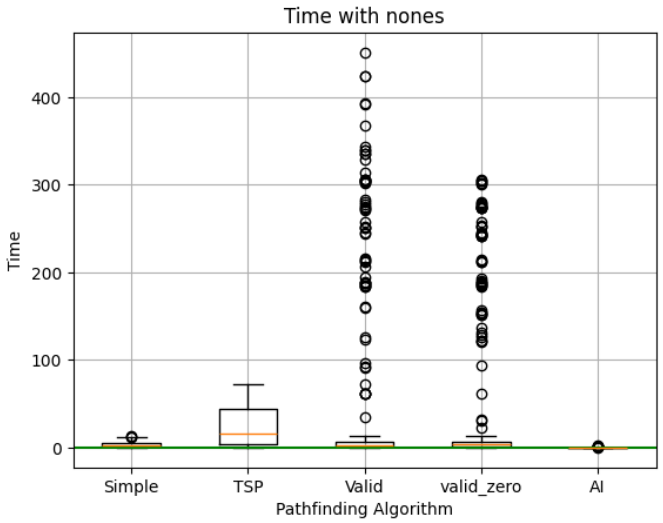

Fig. 6. Times for each approach

around 30 seconds. However the *'valid'* approaches have a much lower median since both have many outliers time-wise. The exact time distribution is shown in figure 6.

Both *'valid'* approaches are the most accurate. Especially for smaller path lengths. The *'AI'* approach seems to overestimate the path lengths for smaller paths quite often. It is also important to note that the *'AI'*' was trained on the same data the evaluation took place in. On real data the *'AI'* will likely perform worse. However especially on longer paths the *'AI'* provides similar results to the 'simple' algorithmic approach. Time wise the algorithmic approach is many times faster than any algorithmic approach.

C. Improvements

As shown in figure 7 the detected center point of the circle doesn't match the exact center. However it seems the connection points also don't quite match with circle centers. Both the circle detection and the connection-scoring were evaluated on generated data. What is shown here is original data. It is possible the generated data isn't as good of an approximation as first thought. To truly evaluate the accuracy of the circle detection and line scoring, time consuming manual labeling of the original data must be done. With better circle and line-scoring the algorithmic approach will likely perform even better. The exact timing of the algorithmic approach could also be improved. There wasn't enough time to accurately fine tune the timeouts of the *'valid'* and *'tsp'* approach in order to get the ideal ration of detected paths and time expended. The path-finding algorithm can always be improved. Currently it is still many times faster to find mistakes later in the path than mistakes earlier on. It is also likely the algorithm runtime could be improved with multithreading.

V. Conclusion

Both the Machine Learning and the algorithmic approaches have their use cases. While the Machine Learning approach is

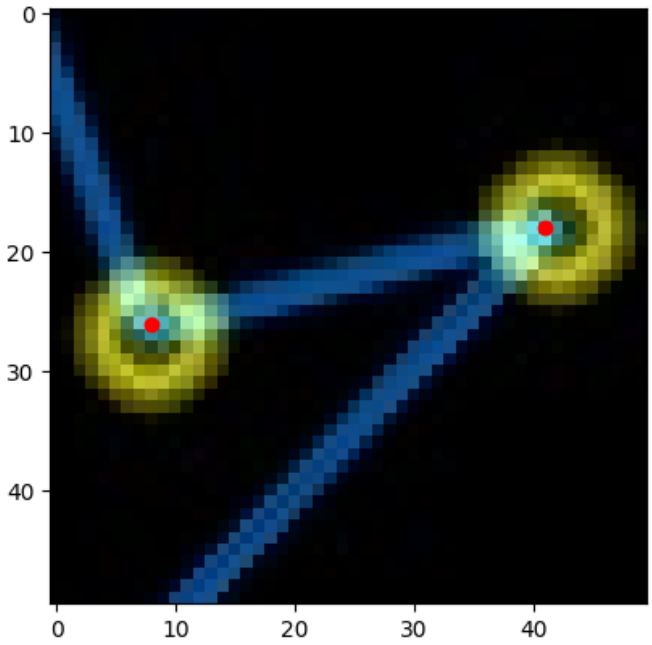

Fig. 7. Example of he point detection

much faster at calculating a path, it is also less accurate. An advantage of the algorithmic approach is the ease with which it is possible to retrace the solution provided by the approach. The choice of the approach depends on the application it will be used for.

While working on the problem, the differences between ML and algorithmic methods were very pronounced. The extreme advantage of the algorithmic approach in providing the correct path and, consequently, the correct path length became less pronounced with the increasing complexity of the provided image. At some point, the algorithmic approach starts to take longer to solve, then returns incorrect paths, or even provides no path at all. Meanwhile, the ML approach continues to provide approximate solutions in a predictable time, independent of the image complexity.

REFERENCES

[1] https://www.kaggle.com/datasets/jeffheaton/traveling-salesman-computer-vision
[2] https://www.kaggle.com/code/jeffheaton/starter-tsp-cv
[3] https://pypi.org/project/python_tsp/