Machine learning algorithm application in trip planning

Grantas Gadliauskas, Andrius Kriščiūnas

Kaunas University of Technology, Faculty of Informatics,
K. Donelaičio St. 73, 44249 Kaunas, Lithuania
grantas.gadliauskas@ktu.edu, andrius.krisciunas@ktu.lt

Abstract. This article explores how machine learning can be applied in efficiently solving a variation of the Travelling Salesman Problem (TSP) in the context of air travel tourism. Large number of cities create too many trip route combinations to be efficiently evaluated in real time. The method proposed uses a feed-forward neural network to narrow down the number of trip route combinations, while a more traditional algorithm based on dynamic programming is then able to select the best trip offers. It was shown that the method could be applied in practice to achieve almost real-time generation of best possible trip offers while evaluating a large amount of real-world flight data.

Keywords: travelling salesman problem, flight search, combinatorial optimization, neural network.

1 Introduction

Consider a tourist who wants to visit several different cities in a specific date range in a round trip from his home city. The tourist might also have preferences to which cities one wants to visit or avoid. A list of *N* best possible trip offers then should be provided to the user, based on the real-world flight data. The quality of the trip is determined by its price, but additional metrics could be added.

Since flight data updates very often and the number of possible date ranges is immensely huge it is not practical to pre-calculate all the offers. On the other hand, finding the best offers in real-time is inefficient due to the need to compute the best scored combination of flights for a large amount of possible trip routes.

In the combinatorial optimization domain, the more simplified version of this problem is well known as the Travelling Salesman Problem (TSP) [1]. More recent works on the topic also include machine learning approaches such as one by Chaitanya K. *et al.* [2] which makes use of neural networks to

DOI: https://doi.org/10.15388/LMITT.2022.3

perform TSP efficiently with hundreds of nodes. For our problem, however, the number of nodes (possible trip flights) will never be more than a few hundred, but the more important issue is the number of trip routes growing exponentially because of the number of different cities.

This article proposes a heuristic solution that allows to efficiently find the best trip offers using a feedforward neural network combined with a tree search algorithm based on dynamic programming (hereinafter DP). The feedforward neural network (hereinafter FNN) model can narrow down the total number of possible trip route combinations to a smaller amount of potential best trip candidates, while the algorithm based on dynamic programming is then able to select the *N* best trips. 3 scenarios with different constraints on the trip offer are tested.

2 Method

Each trip contains a set of cities *T*, $|T| = N$, with a particular order. A trip starts at the start city t_1 and the last city visited is denoted as t_n. Every trip is a round trip and ends at the start city t_1. The ordered set *T* is referred to as a *trajectory.*

The trajectories used in our experiment are made up from 100 selected European cities. City selections are based on OpenStreetMap Place Importance Score (OSM PIS). Each city has a set of airports assigned to it, which is used to associate flight data with the city. The cities used are marked in Figure 1.

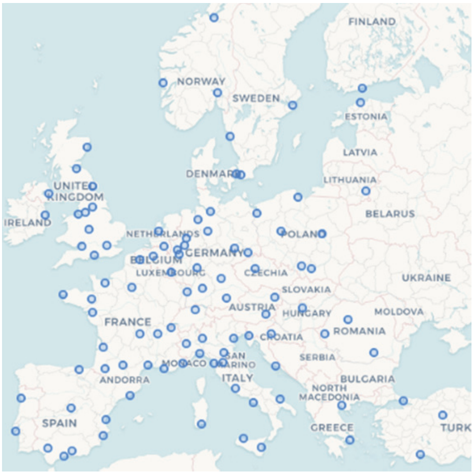

Figure 1. Selected cities

Each of the trajectories consist of 5 cities. Trajectories are generated based on the real-world flight routes which seldom change. Since there could be a total of 100 · 99 · 98 · 97 · 96 trajectories, it would be impractical to consider all the possible combinations. Trajectory amount can be reduced by selecting only the most attractive trajectories – ones with the best trajectory round score and combined OSM PIS. Round score is calculated by dividing the total trajectory distance by the minimal possible distance connecting all the cities. Since only 5 cities make the trajectory, calculating the round score is trivial. 210000 trajectories were generated for our experiment. An example of a trajectory with its adjacent cities connected by a blue line is presented in Figure 2.

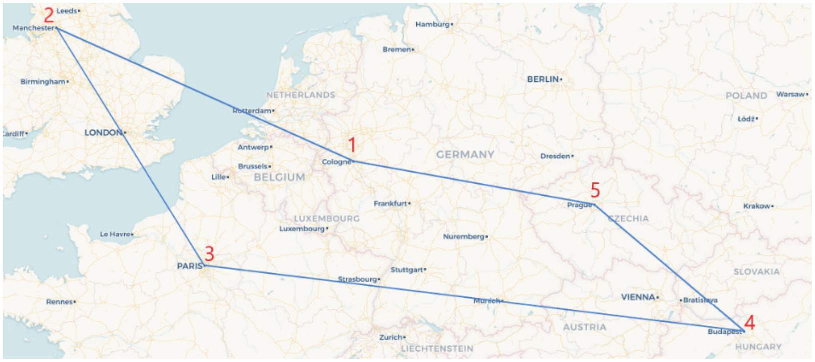

Figure 2. Trajectory graph example

The score of the trip equals to the sum of its flight ticket prices. The lower the score, the better the trip is considered.

Our method to find the best *N* trips given *M* possible trajectories and flight data is as follows:

1. Pass the flight prices of *M* trajectories of the given date range to the FNN model. Each day can have at most a single flight for a given route between two cities.
2. Pass some amount of best predicted trajectories and their flight prices to the DP algorithm.
3. Use the best *N* trajectories returned by the DP algorithm and the flight data to build the best *N* trip offers.

To determine if our method is viable in practice, we evaluate speed and accuracy metrics. Speed is measured as the combined computing time of FNN prediction and DP algorithm. Accuracy is determined by comparing the final output of *N* best trajectories to expected *N* best trajectories and diving the sum of matching pairs by *N*. Accuracy is influenced by the number of total trajectories passed to the FNN model and the number of best model predictions passed to the DP algorithm.

3 trip generation scenarios are explored in this research. They differ by the constraints applied to what can be considered a viable trip.

- Scenario 1: No additional constraints.
- Scenario 2: Each city has a minimum, maximum and preferred number of days to spend in that city. Trip generation must respect the minimum and maximum constraints and apply a penalty if the time spent in a city does not match its preferred number. The penalty subtracts a value of 5 from the trip score per absolute day offset.
- Scenario 3: Same as scenario 2, also, a total trip length constraint is added. The trip length must be in one of the three intervals:
  - o 10 – 13 days
  - o 14 – 20 days
  - o 21 – 24 days

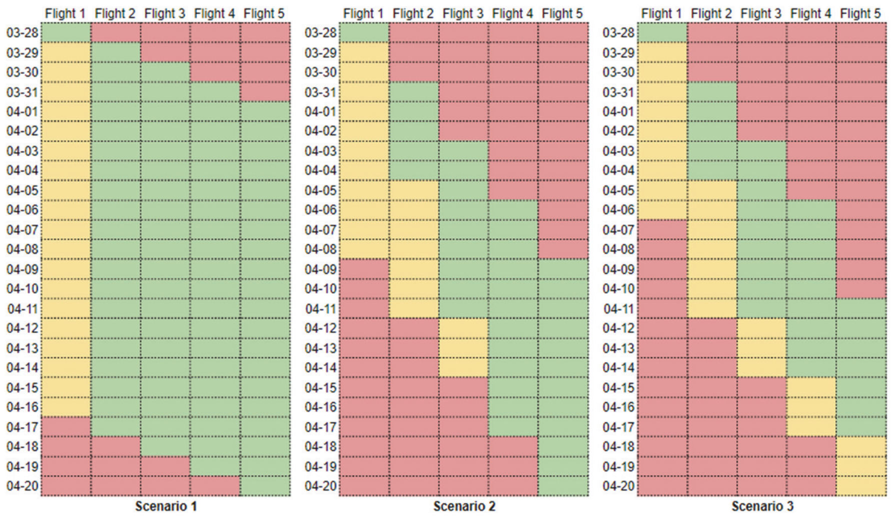

Figure 3. Testing scenarios

The scenarios are visualized in Figure 3. Testing scenarios, where the X axis represents the flights, and the Y axis represents the days. The red cells mark the days in which it is impossible to take the flight to match the given constraints. The green cells represent the possible days to take flights if trip is starting from the earliest day (03-28). Yellow cells represent other possible days if trip were to start from another day. With each scenario, the amount of possible flight combinations is reduced. In the Figure 3. Testing scenarios example, the min. and max. number of days to stay in every city are 3 and 7 respectively and the total trip length should span from 14 to 20 days. In our experiment, for all scenarios we constrain the maximum trip length to 24 days.

3 Experiment

The computing environment used for this research has the following parameters:

- RAM memory: 32 GB
- CPU: AMD Ryzen 7 3700X 8-Core Processor 3.60 GHz (1 CPU core used)
- OS: Windows 10 64-bit architecture

Python 3.9 programming language is used to program the software. PyTorch machine learning framework is used to create the neural network model.

Common neural network training hyperparameters for each scenario are as follows:

- Batch size: 128
- Optimizer: Adam [3]
- Loss function: Mean absolute error (MAE)
- Learning rate: Reducing learning rate on plateau by a factor of 0.5 on 3 consecutive epochs without improvement.
- Epochs: Until does not improve for 10 epochs or until 60.

The departure dates for all the flight data in the experiment span between 2021-03-28 and 2021-05-15. Flight data from 2021-03-28 to 2021-04-20 is used for training, while data from 2021-04-21 to 2021-05-15 is used for validation. For scenario 1, flight prices of 24 days for each of the 5 trajectory cities are passed to the model as an input, for a total input length of 120. If the flight data for a particular day is missing, it is passed to the model as a value of -1. For scenario 2, the number of min., max. and preferred days for each city is added to the input, which increases the input length to 135.

For scenario 3, numbers for min. and max. trip length are added, for an input length of 137. The target trip scores for the model training were built using the DP algorithm. If not a single trip can be built for a trajectory under certain constraints, the target is set to a value of 1000. The model outputs a single value – a trip score. 210000 inputs were used to train model for scenarios 1 and 2, while scenario 3 trained with three times number of inputs (630000) due to three distinct intervals used for total trip length.

Neural network validation accuracy during training for each scenario is presented in figures Figure 4 , Figure 5 and Figure 6. The used notation to describe the model architectures in the figure legends is as follows: *I x H ∗ N x O*, where *I* is the number of inputs for the input layer, *H* is the number of inputs for each hidden layer, *N* is the number of hidden layers and *O* is the number of outputs (1 output describing the trip score). The best model architecture is highlighted with a yellow marker. In general, to obtain the optimal validation accuracy, the models had to become more complex as the trip constraints increased. Model for scenario 2 tends to overfit the most and the model state after 4th epoch is used for its metric check. Techniques such as dropout [4] and dataset scaling were tested but failed to improve the model accuracy.

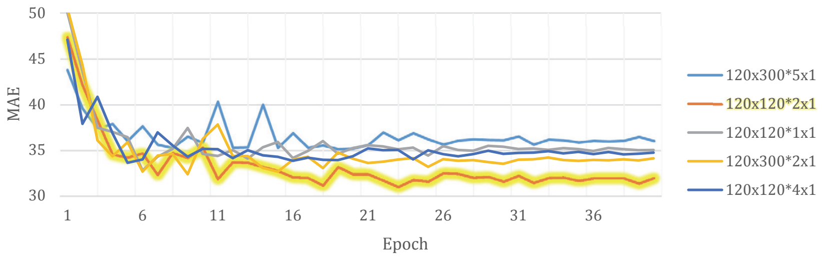

Figure 4. Scenario 1 validation accuracy

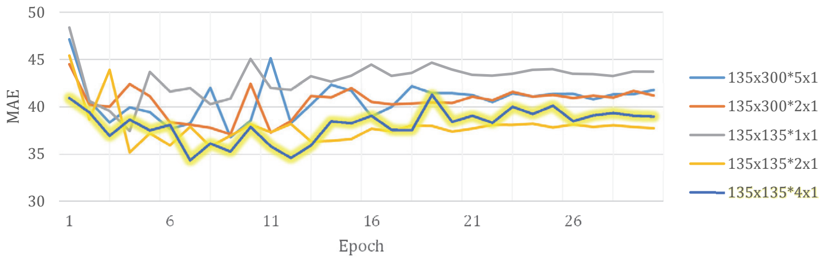

Figure 5. Scenario 2 model validation accuracy

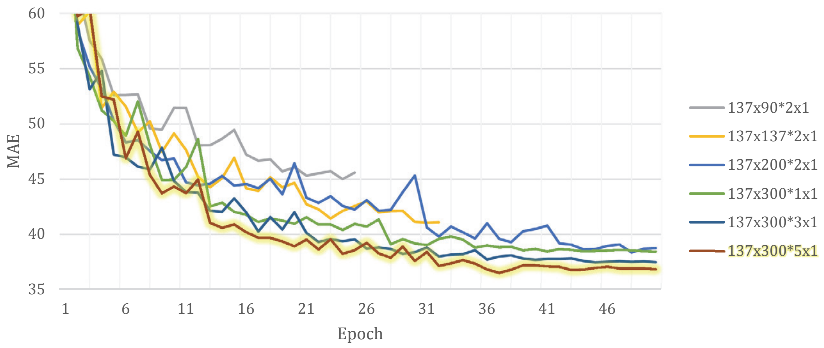

Figure 6. Scenario 3 model validation accuracy

The dynamic programming algorithm is used to find the actual best possible flight combination for a given trajectory. It works like a brute force tree search algorithm, but with optimizations. Instead of checking each possible flight combination, it stops traversing the flights if the flight for a given day was already traversed and had a better total flight price. It also only iterates through the dates which match the constraints of the min. and max. days to stay in a certain flight city and ignores days which do not match the total trip length constraint (such days are marked red in Figure 3).

4 Results

The final accuracy and speed results are presented in tables Table 1 and Table 2. Accuracy match results were averaged over 50 test runs. In the accuracy result table, cells marked in red, yellow, and green represent respectively the worst, the second best and the best scenario for the testing parameters of the rightmost 3 columns. Columns "*Total predictions made*", "*Top N predictions to search in*" and "*Required top N matches*" denote respectively how many inputs were passed to the FNN model, how many of the best results then were passed to the DP algorithm, and how many final trip offers do we want to output. The cell values in bold mark the values for which at least 80% of the required top *N* matches do match, which is considered a good result.

It is possible to infer from the accuracy results that the more constrained the trip generation scenario is, the more accurate the final matches tend to be. Since the mean absolute error of the FNN model validation accuracy was lower for the more constrained models, this might not seem reasonable.

Table 1. Accuracy results

Actual matches			Required top N matches	Top N predictions to search in	Total predictions made
1 Scen.	2 Scen.	3 Scen.			
5,48	6,72	**8,72**	10	50	2000
10,86	12,20	**16,22**	20	50	2000
23,04	23,84	31,78	50	50	2000
6,76	**8,26**	**9,50**	10	100	2000
13,92	15,38	**18,42**	20	100	2000
32,42	34,18	**41,74**	50	100	2000
8,96	**9,82**	**9,82**	10	250	2000
18,16	**19,26**	**19,46**	20	250	2000
43,62	**44,76**	**47,58**	50	250	2000
9,84	**9,98**	**9,84**	10	500	2000
19,70	**19,94**	**19,56**	20	500	2000
48,58	**48,90**	**48,82**	50	500	2000
3,60	3,92	6,08	10	50	10000
6,26	6,50	11,00	20	50	10000
11,32	12,32	21,28	50	50	10000
5,16	6,32	**8,12**	10	100	10000
9,22	11,06	15,34	20	100	10000
19,56	22,62	32,82	50	100	10000
7,00	**8,02**	**9,32**	10	250	10000
13,14	14,92	**18,40**	20	250	10000
29,28	34,88	**43,92**	50	250	10000
8,12	**9,24**	**9,76**	10	500	10000
15,46	**17,44**	**19,44**	20	500	10000
35,74	**41,90**	**48,04**	50	500	10000
3,48	1,24	3,92	10	50	50000
4,46	2,06	6,74	20	50	50000
5,80	4,24	12,12	50	50	50000
4,86	2,24	5,50	10	100	50000
6,70	3,44	9,94	20	100	50000
10,14	7,80	19,50	50	100	50000
6,60	6,12	**8,34**	10	250	50000
10,04	10,98	15,10	20	250	50000
17,10	21,58	31,74	50	250	50000
7,98	**8,44**	**9,44**	10	500	50000
12,70	15,54	**17,50**	20	500	50000
24,18	33,50	**40,68**	50	500	50000

Table 2. Speed results

Trajec-tories	Scenario 1		Scenario 2		Scenario 3	
	FNN time	DP time	FNN time	DP time	FNN time	DP time
50	0:00:00:001	0:00:00:022	0:00:00:001	0:00:00:017	0:00:00:002	0:00:00:013
100	0:00:00:001	0:00:00:045	0:00:00:001	0:00:00:035	0:00:00:004	0:00:00:023
250	0:00:00:001	0:00:00:109	0:00:00:003	0:00:00:093	0:00:00:009	0:00:00:065
500	0:00:00:002	0:00:00:227	0:00:00:004	0:00:00:183	0:00:00:018	0:00:00:123
2000	0:00:00:008	0:00:00:881	0:00:00:016	0:00:00:653	0:00:00:070	0:00:00:470
10000	0:00:00:041	0:00:04:351	0:00:00:080	0:00:03:141	0:00:00:338	0:00:02:305
50000	0:00:00:191	0:00:21:987	0:00:00:423	0:00:15:972	0:00:01.706	0:00:11:449

However, it may be explained by the greater value of the standard deviation of more constrained scenario model target array (trip scores) compared to less constrained scenario targets. The final matches tend to be less accurate the more trajectory inputs are passed to the FNN model and the fewer top predictions are ran through the DP algorithm.

The speed results show that the performance of the FNN is extremely quick running faster than half of a second for 50000 trajectory inputs in 1st and 2nd scenario and in 1.7 seconds in 3rd scenario, which uses a more complex neural network architecture. The DP algorithm time decreases as the amount of trip constraints increases.

5 Conclusions

In this article it was investigated if combining the speed of feedforward neural networks and the accuracy of traditional search algorithms can be used to quickly generate attractive trip offers using real world flight data. The results show that for cities which contain as much as 50000 trajectories, it is possible to generate as much as 50 trip offers in which at least 80% of them match the best possible offers in under 2 seconds under the constraints of this experiment. This shows that the method can be applied in practice, and it will be strongly considered to be integrated into a newly developing trip planning software system.

References

[1] R. M. Karp, „Reducibility among combinatorial problems,“ *Complexity of computer computations,* pp. 85-103, 1972.

[2] C. K. Joshi, Q. Cappart, L.-M. Rousseau, T. Laurent, „Learning the Travelling Salesperson Problem Requires Rethinking Generalization,“ 2020.

[3] D. P. Kingma, J. Ba, „Adam: A Method for Stochastic Optimization,“ 2014.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, „Dropout: a simple way to prevent neural networks from overfitting,“ *The journal of machine learning research,* 2014.