Combining naive bayes model and TSP routing algorithm for predicting high quality route sequences

Wenbin Zhu and Ying Fu

School of Business Administration, South China University of Technology

381 Wushan St. Tianhe District, Guangzhou, GD, China

Luo Zhixin

School of Management and Engineering, Nanjing University

Hankou Road 22, Gulou District, Nanjing, Jiangsu Province, China

Abstract

In last mile delivery, experienced drivers often have tacit knowledge, such as which road is hard to navigate. Generally, such knowledge is hard to formulate explicitly and is provided as a set of historical high quality routing sequences. We try to learn the tacit knowledge and predict visiting sequences of unseen future route. Based on the observation that drivers tend to serve customers zone by zone, we decompose the task into a high-level zone sequence prediction and a low-level TSP tour construction. Assuming Markov property, we employ naive bayes model for predicting zone sequences. Historical data are employed to establish transition probability between zones and we seek to identify a zone sequence that maximizes likelihood of been observed. An iterative merging algorithm inspired by clustering algorithm is devised to train the naive bayes model. After predicting zone sequence, we compute the TSP tour for stops in each zone to form visiting sequences of stops. Compared with conventional optimization algorithms, such as heuristic TSP or VNS that attempts to minimize total travel distance, our method produces sequences closer to the actual sequence taken by the drivers. Our method also outperforms sequence to sequence neural network that attempts to directly predict the visiting sequence at stop level.

1. Introduction

Given a set of historical routing sequences with quality labeled low, medium and high by logistic operation experts, our model learns the characteristics of high-quality sequences and predicts visiting order of future unseen high-quality sequences. The performance of our model is measured by how well the predicted visiting sequence matches the actual one by weighted editing distance between the two sequences. It is quite different from conventional routing optimization problem, where the quality of solution is measured by precisely defined objective functions such as total travel time. We believe the key to achieve a good match is to figure out the criteria used by drivers and logistic experts explicitly or implicitly to measure the quality of route sequence. Since the criteria is not explicitly described, we must learn from historical data.

Our first attempt is to utilize the successful sequence to sequence models in many NLP tasks to directly train a neural network based on historical routing sequence. This model performs poorly with an average score of 0.1124 over 2,718 historical high-quality routes, worse than 0.0864 by a heuristic TSP algorithm that merely minimize total distance. We suspect the poor performance is mainly due to the lack of data. Since historical sequence consists of 33 to 238 stops, much longer than the number of words in typical sentences, our model is dealing with prediction tasks larger in scale than typical NLP models. We suspect that more data are required to handle the lager prediction tasks. However, we only have 6,112 historical sequences, whereas typical NLP models are trained with huge corpora that consist of millions of sentences.

Data suggest that drivers tend to visit customers zone by zone, that is, customers in same zone tend to appear consecutively in route sequence. Merging consecutive stops with same zone in historical routes results in a visiting sequence of zones. The average length of zone sequence is about 22, much lesser than the average number of stops in routes, 148. Further analysis shows that zones seldom repeat in a zone sequence. Therefore, we decompose the prediction task into two sub-problems. We first predict the visiting sequence at zone level, then try to predict the visiting order of stops within a zone. For the former task we employ a naive bayes model that tries to maximize the likelihood of zone sequence based on conditional probability estimated from historical data. For the later task, we try to compute the TSP tour of stops in a zone to minimize the total travel time. It turns out that both tasks can be solved by computing a TSP tour with suitably defined distance matrix.

2. Literature Review

The traveling salesman problem (TSP) is one of the most widely studied combinatorial optimization problems, which consists of determining a set of routes for one or several salesmen who visit a given number of cities exactly once and return to the initial positionCheikhrouhou & Khoufi [2021]. Owing to its NP-hardness, TSP is regarded as one of the most challenging problems in Operational Research Pandiri & Singh [2018]. To address the traditional TSP and its variants, three types of approaches, namely exact algorithms, heuristics and meta-heuristics, and other approaches, have been investigated in the literature.

Exact algorithms aim to achieve the optimal solution, and hence are computationally expensive and more suitable for small-sized TSPs. The related existing literature is quite limited. Bellman [1962] proposed a dynamic programming algorithm for TSP. Dell'Amico et al. [2021] presented a branch-and-bound algorithm for the flying sidekick TSP. In this algorithm, a fast lower bound, dominance rules, and three heuristic methods were integrated. Yuan et al. [2020] proposed a branch-and-cut algorithm for the generalized TSP with time windows. To reduce the computation times, an initial upper bound was provided by a simple and fast heuristic. To deal with a new TSP with hotel selection, Barbosa & Uchoa [2020] developed a sophisticated branch-cut-and-price algorithm. Computational results showed that many medium-sized instances, having up to 75 clients and 20 hotels, could be solved to optimality.

Due to great difficulties in solving the TSP and its variants, recent research efforts have been focused

on developing efficient heuristics and meta-heuristics. The heuristics are simple strategies that help produce good solutions. One of the commonly used heuristics for TSP is 2-opt. The meta-heuristics, which is capable of iteratively improving the candidate solutions, have also been introduced to find approximate solutions within reasonable computational cost. The existing meta-heuristics for TSP are classified into two categories: the single solution-based meta-heuristics (SSBM) and the population-based meta-heuristics (PBM). The SSBM maintains only one solution at each cycle and have the power to intensify the search in local regions. The commonly used SSBMs are simulated annealing (SA) Küçükoğlu et al. [2019], tabu search (TS) Lin et al. [2016], variable neighborhood search (VNS) Wang et al. [2019], and large neighborhood search (LNS) Smith & Imeson [2017], etc. Different from SSBMs, PBMs are either based on evolutionary strategies or swarm intelligence. These approaches use the properties of the population to guide the solution to the optimality, therefore are capable of performing a good exploration of search space. The most used PBMs for TSP in the literature include genetic algorithm (GA) Groba et al. [2015], particle swarm optimization (PSO) Mahi et al. [2015], and ant colony optimization (ACO)Wang & Han [2021], etc.

Different techniques, such as probability, game theory, and fuzzy logic, have also been applied for TSP. Kulkarni & Tai [2010] introduced a method using probability collectives, in which the vehicles were represented as autonomous agents and vehicles route as a strategy. Khoufi et al. [2016] proposed a multi-objective optimization problem to determine the robots tours responsible for collecting data from sensor nodes and delivering the data to the depot. A game theory approach was developed to optimize the maximum tour time, the number of robots, and balance the robots' tours. To address a multi-objective TSP, a fuzzy logic-based approach was developed by Trigui et al. [2017]. This approach combined both metrics into a single fuzzy metric and reduced the problem to a single-objective optimization problem.

An extensive literature review indicates that most of research works neglect the travelers' experience when solving the TSP and its variants. Since machine learning is capable of providing useful predictive strategies based on expert knowledge, a promising way is to integrate meta-heuristics with machine learning methods for problem-solving, which may provide better solutions in reasonable computation time.

Sequence-to-sequence nueral networks have achieved good performance in many natural language processing tasks. Vinyals et al. [2015] extend this framework to *pointer network* that uses attentions to output a permutation of input sequences. It is capable to produce high quality route for small scale problems, they reported results for TSP with up to 50 stops. Bello et al. [2016] employ reinforcement learning to train the pointer network without supervised solution. Their approach can find high quality TSP tour with up to 100 stops. Deudon et al. [2018] combine reinforcement learning with a 2OPT local search to solve TSP and Kool et al. [2019] combine reinforcement leanring with transformer architecture to produce TSP tour. In terms of minimizing total distance, machine learning based methods cannot compete with hand-crafted state-of-art heusristics yet, neverthless, they have the potential to pickup domain specific knowledge from training data. The three reinformcement learning based approaches do not require known TSP tour as supervised sample, but they all need a function that can accurately evluate the quality of a route. In our research challenge,

there is no explicity qualitative measure for the quality of route.

3. Methodology

Given a set of historical route sequences with *route_score* set to low, medium and high by logistic experts. Our task is to predict the visiting order of stops in unseen high quality route sequences. Since drivers tend to visit stops zone by zone, stops within same zone tend to appear consecutively in historical route sequences. We decompose the prediction task into two simpler tasks: 1. predicting the visiting order of zones; 2. computing the visiting order of stops within zones.

3.1. Predicting zone sequence

3.1.1. Naive bayes model

To predict the visiting order of zones, we first merge all stops with same *zone_id* in a historical route sequence to obtain *a zone sequence* where stops without *zone_id* are discarded. Since zones seldomly repeat in historical zone sequences (Table 1), we simplify the prediction task by assuming no zone will repeat. We employ a naive bayes model to predict zone sequences.

Table 1: Number of routes by repeated zones. Among 6112 historical routes, 3835 has no repeated zone accounting for 62.75%

repeated zones	route count	percentage
0	3835	62.75%
1	988	16.16%
2	673	11.01%
3	288	4.71%
≥ 4	328	5.37%

Based on historical zone sequences, we estimate the conditional probability $P(Z_{i+1} = b|Z_i = a)$ of visiting a next zone b given current zone a. We assume Markov property holds, that is, the probability of visiting a next zone only depends on current zone but not any other zone visited before current:

$$P(Z_k|Z_1, Z_2, \cdots, Z_{k-1}) = P(Z_k|Z_{k-1}) \tag{1}$$

Repeatedly apply Markov property, the probability of observing a zone sequence $Z_1, Z_2, \cdots, Z_k$ can be decomposed into

$$\begin{aligned} P(Z_1, Z_2, \cdots, Z_k) &= P(Z_k|Z_1, Z_2, \cdots, Z_{k-1})P(Z_1, Z_2, \cdots, Z_{k-1}) \\ &= P(Z_k|Z_{k-1})P(Z_1, Z_2, \cdots, Z_{k-1}) \\ &= P(Z_k|Z_{k-1})P(Z_{k-1}|Z_{k-2})P(Z_1, Z_2, \cdots, Z_{k-2}) \\ &= \cdots \\ &= (\prod_{i=1}^{k-1} P(Z_{i+1}|Z_i))P(Z_1) \end{aligned} \tag{2}$$

We would like to find a zone sequence with maximum probability. Since a sequence always starts and ends with station, $P(Z_1)$ is constant, maximizing (2) is equivalent to the following likelihood:

$$\prod_{i=1}^{k-1} P(Z_{i+1}|Z_i) \tag{3}$$

Finding a zone sequence that maximizes the likelihood as defined by (3) is our naive bayes model for predicting zone sequence.

3.1.2. Estimating transition probability

We estimate the conditional probability of $P(Z_{k+1} = j|Z_k = i)$ through emperical distribution based on high quality routes. More precisely, given a training set $\mathcal{X}$ consists of some historical zone sequences (we randomly chosen 80% of the high quality route sequences to form a training set),

$$P(Z_{k+1} = j|Z_k = i) \approx \frac{\#\text{ of zone sequences where zone } j \text{ immediately follows } i}{\#\text{ of zone sequences includes zone } i} \tag{4}$$

There are 17 stations in historical data, and the same *zone_id* may appear in two stations and refer to two different zones, as data suggest they are more than one thousand kilometers apart. Therefore the transition probability as defined in equation (4) is computed station by station, for each station the transition probability between any pair of zones are computed. The matrix $P = \{p_{ij}\}$ where $p_{ij} = P(Z_{k+1} = j|Z_k = i)$ is often called transition matrix in the context of Markov chain decision.

There are 8,962 unique *zone_id*, but only 2,718 high quality routes. Each route visits 22 zones on average. The transition matrices estimated through (4) are very sparse, merely 0.04% to 0.30% entries are nonzero. This is a sign of lack of sufficient data to estimate a reliable transition matrix. Therefore, we also tried an alternative method to estimate the transition matrix as follows.

- For each zone pair i, j appear in a zone sequence, we define the rank of zone j to be r if j is the r-th closest from i.
- We define the conditional probability $P(r|Z = i)$ to be the probability of visiting the r-th closest zone from i immediately after zone i. These conditional probabilities are estimated using a training set consists of historical high quality routes.
- Then the transition probability from zone i to zone j is defined as $P(Z_{k+1} = j|Z_k = i) = P(r|Z_k = i)$, where j is r-th closest zone from i.

We call this method *estimation by rank*.

A third method to estimate the un-normalized transition probability is through distance between two zones. Let $d_{i,j}$ be the travel time from zone i to j. We define $p_{i,j} = e^{-d_{i,j}}$, that is, we subjectively set the likelihood of visiting j after i to be higher if j is closer to i. The main rational behind this rather ad-hoc method is that, we observed that most of the zones visited next are the top 3 closest zone.

3.1.3. Solving naive bayes model

Given transition matrix among zones, our Bayes model (3) can be solved by minimizing its negative logarithm:

$$-\log \prod_{i=1}^{k-1} P(Z_{i+1}|Z_i) = \sum -\log P(Z_{i+1}|Z_i) \tag{5}$$

There are two possible methods to solve this minimization problem. First, we can define a distance matrix $D = \{d_{ij}\}$ with entries representing the negative logarithm of conditional probability of visiting zone j after zone i:

$$d_{ij} = -\log P(Z_{k+1} = j|Z_k = i) \tag{6}$$

A zone sequence that minimizes (5) is a sequence that starts and ends with station and minimizes total distance, this can be solved by any standard TSP algorithm taking D as distance matrix.

Our second method is inspired by an iterative clustering algorithms, by iteratively merging two shorter zone sequences into a longer zone sequence as follows:

- step 1, initialization: create one trivial zone segment for each zone
- step 2, iterative merging: pick an ordered pair of zone segments A and B that are most likely to be visited in order, and merge the pair into a single zone segment.
  - step 2a, the likelihood of visiting zone segment B after zone segment A is defined to be the transition probability between the last zone in segment A to the first zone in segment B. Note that two segment A and B cannot be merged if A starts with station and B ends with station and there are other segments left. To avoid this special case we define their likelihood to be negative infinity.
  - step 2b, to merge an ordered pair of zone segment A and B into a single segment and the resulting segment is most likely as defined by (3). There are two modes: first, we try to **append** B after A. Second, we exhaustively divided both A and B into two sub-segments and recombine them. For example if A is divided into $A1$ and $A2$, and B into $B1$ and $B2$. We combine the four segments into a single segment in the order of $A1, B1, A2, B2$.
- step 3, repeat step 2 until there is only one zone segment left.

3.2. Computing TSP tour within zones

Let $Z_1, Z_2, \cdots, Z_k$ be a zone sequence, where Z_1 and Z_k includes only the station. We construct a stop sequence by computing TSP tour for stops within each zone and connecting them as follows:

- Z_1 only consists of the station, its entry stop e_1 is the station.
- $s =$ a trivial stop sequence consists of only the station

- For each zone $Z_i, i = 2, \cdots k$
  - step a: we try to identify a stop e_i in Z_i that is closest to previous zone Z_{i-1}. We call e_i the entry stop of Z_i. For a stop s in zone Z_i its distance to previous zone is defined as the shortest distance to any non-entry stop in previous zone.
  - step b: compute a TSP tour that starts with the entry stop e_{i-1} of previous zone Z_{i-1}, visiting all other stops in Z_{i-1} exactly once and ends at e_i the entry stop of zone Z_i.
  - step c: append the TSP tour in step b to end of s. Note that the first stop in new TSP tour is the last stop in the partial route sequence s.

To compute a TSP tour, we first invoke the greedy heuristic implemented by the python package `https://pypi.org/project/tsp-solver2/`. We try to improve the TSP tour by sliding a window along the sequence as follows:

- Initialization: let $s_1, s_2, \cdots, s_n$ be the initial stop sequences and k be the size of sliding window
- For $i = 1$ to $n - k$, take a sub-tour starts at the i-th stop in s that consists of k consecutive stops. Fix the start and end stop of the sub-tour and rearrange the visiting order of middle stops in the sub-tour. When the length of sub-tour is short, say $k \leq 10$ the optimal rearrangement can be computed by dynamic programming algorithm by Bellman [1962].

4. Results and Conclusions

We conduct computational experiments to decide the best configuration of our approach. Since there is a learning component in our approach, we follow the hold-out strategy commonly used by machine learning community to train and validate our model. We randomly sample 80% of high-quality historical route sequences to form the training set, and use the remaining 20% as validation set. The process is repeated five times. Each time we use training set to train our model and then compute two performance measurement of our model. Firstly, our model predicts the sequence for each route in validation set and the prediction is compared with actual sequence using weighted editing distance as described on the website of this research challenge. The average score over validation set measures how well our model generalizes to unseen data. Secondly, we also randomly select the same number of routes as validation set from the training set and report the average performance of our model on the selected training routes. The second measure is an indication whether our model have learned enough from the training set. The small weighted editing distance indicates a good match. Perfect match will result in a score of 0 and poor match will result in a large score close to 1 or even larger than 1.

4.1. How estimating transition probability affect prediction accuracy of zone sequence

In the first experiment, we try to compare different method of estimating transition matrix as described in section 3.1.2 and their performance on predicting zone sequences. The actual zone sequence is computed

by merging stops with same *zone_id* in historical route sequences. Occasionally, some zone appear more than once. In this case, we will only keep the occurrence resultant from merging the most number of stops.

We use TSP as baseline. For every zone, the centroid of stops in the zone is taken as its location. The travel distance between two zone are approximated by the geographical distance (see `https://en.wikipedia.org/wiki/Geographical_distance`) between their centroids. We invoke the heuristic tsp solver in python package tsp-solver2 to obtain a zone sequence that minimizes the total travel distances. The result of our baseline algorithm is reported in first row in Table 2. Column "validation score" is the average score on validation set and "in sample score" is the average score on training set. We can see both score are rather poor for the baseline algorithm. The next two rows report the zone sequence predict by our iterative

Table 2: Average performance on predicting zone sequence.

algo	validation score	in sample score
tsp	0.3497	0.3466
merge(rank)	**0.0957**	**0.0942**
merge(empirical)	**0.1564**	**0.0163**

merging method described in section 3.1.3. They differ in how transition probability between two zones are estimated. For the row **merge(rank)**, estimation is done by rank as described in section 3.1.2 is employed; for the row **merge(empirical)**, estimation is by empirical probability as described by equation (4). As we can see, estimation by rank performs better on validation set with an average score of 0.0957, in contrast to 0.1564 by empirical.

Both merge(rank) and merge(empirical) are substantially better than our baseline TSP algorithm. Therefore, we decided to use our iterative merging algorithm to solve our naive bayes model to predict zone sequences.

4.2. Comparing different algorithms on zone sequence prediction

We described two algorithms for solving our naive bayes model in section 3.1.3. Regardless how we estimate the transition probability, our iterative merging algorithm performs better than TSP algorithm, see Table 3. For example, we transition probability is estimated empirically by equation (4), iterative merging algorithm achieves an average score of 0.1564 on validation set (3rd row in the table), much better than 0.2534 produced by TSP.

Therefore, we decided to use iterative merging algorithm to solve our naive bayes model for predicting the zone sequence.

4.3. Performance on predicting route sequence

Our overall method for predicting route sequence is described as follows. Firstly, we estimate the transition probability between two zones using historical data. Secondly, based on the estimated transition matrix,

Table 3: Two algorithms for solving naive bayes model and their performance on predicting zone sequences

estimating probability	algo	validation score	in sample score
by_rank	merge	**0.0957**	**0.0942**
	tsp	0.1640	0.1628
emperical	merge	**0.1564**	**0.0163**
	solve_tsp	0.2534	0.1767

we employ iterative merging algorithm to predict the zone sequence and then convert zone sequence into stop sequences by computing TSP for stops in each zone as described in section 3.2. The size of sliding window $k = 10$ is used, so that the dynamic programming formulation of a sub-tour can be computed within 1 seconds on a moderate laptop.

We compare the effect of two methods of estimating transition probability, their results are summarized under heading "merge(rank)" and "merge(empirical)" respectively in Table 4. Training and validation is repeated five times, the average score on validation and training set for each repetition are reported. The last two rows report the average and the stand deviation across the five repetition.

Table 4: Comparison on predicting route sequence.

repetition	merge(rank)		merge(empirical)	
	validation	in sample	validation	in sample
1	0.0959	0.0542	0.0757	0.0178
2	0.0850	0.0594	0.0729	0.0185
3	0.0843	0.0677	0.0775	0.0172
4	0.0781	0.0704	0.0714	0.0178
5	0.0772	0.0720	0.0723	0.0173
avg	0.0841	0.0647	**0.0739**	0.0177
std	0.0067	0.0068	**0.0023**	0.0005

Although merge(rank) achieve better zone level prediction accuracy than merge(empirical) as shown in Table 2, its performance on predicting stop sequences are worse with an average score of 0.0841 on validation set, in contrast to 0.0739 by merge(empirical). The performance of merge(rank) is also less table across repetition as indicated by a larger standard deviation of 0.0067, in contrast to 0.0023 by merge(emperical).

In our submission, we estimate transition probability by empirical probability and we utilize all historical high quality routes available.

Figure 1 shows the histogram of the score of 545 route sequences in the validation set produced by our submitted model in repitition 2. More than half of scores are less than 0.063, whereas only 30 scores exceed

0.171.

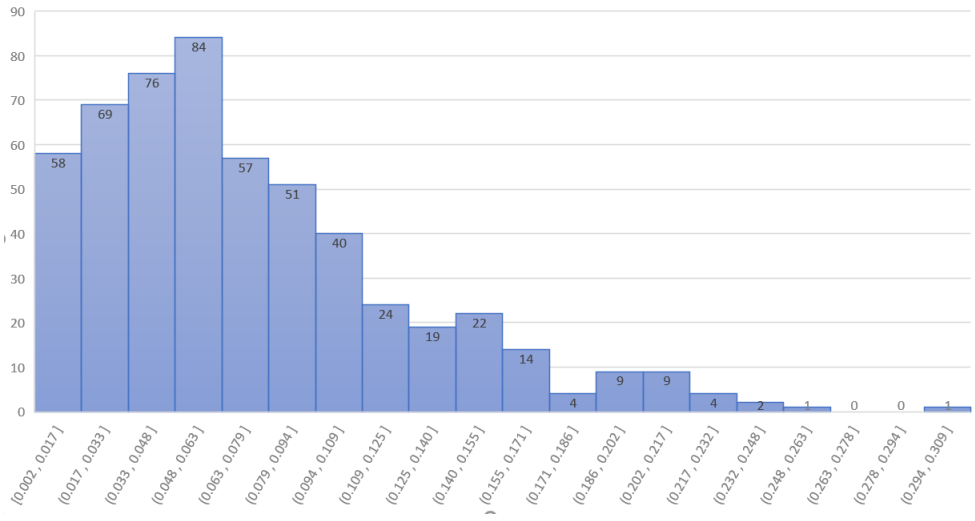

Figure 1: Distribution of route sequence by score, repetition 2

4.4. Other failed attempts

We implemented a few alternative methods.

- TSP: we ignored the delivery time window of all packages and try to compute a TSP tour that minimizes the total travel time. The average score of this method is 0.0846 on all hiqh quality historical routes.
- VNS: for each drop off stop, if any package has a delivery time window, we compute a time window for the stop. The start time for a stop is the latest start time of all packages and the end time for stop is the earliest end time of all packages. If the driver arrives at a stop before start time, it incurs a penalty proportional to the earliness; if the driver arrives after end time, it incurs a penalty proportional to the lateness. We devise a variable neighborhood search heuristic that employ well-known operators such as swap, relocate to minimize the total travel time and penalty. Our VNS is slightly better than TSP with an average score about 0.08. When we further attempt to reduce earliness and lateness, it produces route sequence with worse scores.
- ML: a pure machine learning based method. Given current stop i, we try to predict $f(i, j)$, the number of stops visisted before we visited a stop j. To enrich input features, we also includes the neariest $k = 5$ unvisisted stops for j and i, respectively. That is to say, input of function f will includes features of 12 stops. For each stop, we include features, such as lat, lng, package count, total service time, etc. We employ a two layer fully connected network with ReLU activation function to model f. After the model is trained, to predict a new route, we start with i being the station, then invoke f with every unvisited stop j to find the stop with smallest $f(i, j)$ and use it as next stop. The average score of route sequences produced by this model is 0.1125, which is worse than TSP score.

References

Barbosa, L. H., & Uchoa, E. (2020). A branch-cut-and-price algorithm for the traveling salesperson problem with hotel selection. *Computers & Operations Research*, *123*, 104986.

Bellman, R. (1962). Dynamic programming treatment of the travelling salesman problem. *J. ACM*, *9*, 61–63. URL: https://doi.org/10.1145/321105.321111. doi:10.1145/321105.321111.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *CoRR*, *abs/1611.09940*. URL: http://arxiv.org/abs/1611.09940. arXiv:1611.09940.

Cheikhrouhou, O., & Khoufi, I. (2021). A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy. *Computer Science Review*, *40*, 100369.

Dell'Amico, M., Montemanni, R., & Novellani, S. (2021). Algorithms based on branch and bound for the flying sidekick traveling salesman problem. *Omega*, *104*, 102493.

Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., & Rousseau, L.-M. (2018). Learning heuristics for the tsp by policy gradient. In W.-J. van Hoeve (Ed.), *Integration of Constraint Programming, Artificial Intelligence, and Operations Research* (pp. 170–181). Cham: Springer International Publishing.

Groba, C., Sartal, A., & Vázquez, X. H. (2015). Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: An application to fish aggregating devices. *Computers & Operations Research*, *56*, 22–32.

Khoufi, I., Minet, P., Koulali, M.-A., & Kobbane, A. (2016). Path planning of mobile sinks in charge of data gathering: A coalitional game theory approach. In *2016 IEEE 35th international performance computing and communications conference (IPCCC)* (pp. 1–8). IEEE.

Kool, W., van Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems! In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=ByxBFsRqYm.

Küçükoğlu, İ., Dewil, R., & Cattrysse, D. (2019). Hybrid simulated annealing and tabu search method for the electric travelling salesman problem with time windows and mixed charging rates. *Expert Systems with Applications*, *134*, 279–303.

Kulkarni, A. J., & Tai, K. (2010). Probability collectives: a multi-agent approach for solving combinatorial optimization problems. *Applied Soft Computing*, *10*, 759–771.

Lin, Y., Bian, Z., & Liu, X. (2016). Developing a dynamic neighborhood structure for an adaptive hybrid simulated annealing–tabu search algorithm to solve the symmetrical traveling salesman problem. *Applied Soft Computing*, *49*, 937–952.

Mahi, M., Baykan, Ö. K., & Kodaz, H. (2015). A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. *Applied Soft Computing*, *30*, 484–490.

Pandiri, V., & Singh, A. (2018). A hyper-heuristic based artificial bee colony algorithm for k-interconnected multi-depot multi-traveling salesman problem. *Information Sciences*, *463*, 261–281.

Smith, S. L., & Imeson, F. (2017). Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, *87*, 1–19.

Trigui, S., Cheikhrouhou, O., Koubaa, A., Baroudi, U., & Youssef, H. (2017). Fl-mtsp: a fuzzy logic approach to solve the multi-objective multiple traveling salesman problem for multi-robot systems. *Soft Computing*, *21*, 7351–7362.

Vinyals, O., Fortunato, M., & Jaitly, N. (2015). Pointer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc. volume 28. URL: `https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf`.

Wang, X., Golden, B., & Wasil, E. (2019). A steiner zone variable neighborhood search heuristic for the close-enough traveling salesman problem. *Computers & Operations Research*, *101*, 200–219.

Wang, Y., & Han, Z. (2021). Ant colony optimization for traveling salesman problem based on parameters optimization. *Applied Soft Computing*, *107*, 107439.

Yuan, Y., Cattaruzza, D., Ogier, M., & Semet, F. (2020). A branch-and-cut algorithm for the generalized traveling salesman problem with time windows. *European Journal of Operational Research*, *286*, 849–866.