Industrial Application of Artificial Intelligence to the Traveling Salesperson Problem

Michael Lehenauer
Salzburg University of Applied Sciences
Salzburg, Austria
mlehenauer.its-m2020@fh-salzburg.ac.at

Stefan Wintersteller
Salzburg University of Applied Sciences
Salzburg, Austria
swintersteller.itsb-m2020@fh-salzburg.ac.at

Martin Uray
Salzburg University of Applied Sciences
Salzburg, Austria
martin.uray@fh-salzburg.ac.at

Stefan Huber
Salzburg University of Applied Sciences
Salzburg, Austria
stefan.huber@fh-salzburg.ac.at

***Abstract*—In this paper we discuss the application of Artificial Intelligence (AI) and Machine Learning (ML) to the exemplary industrial use case of the two-dimensional commissioning problem in a high-bay storage, which essentially can be phrased as an instance of traveling salesperson problem (TSP).**

We investigate the *mlrose* library that provides an TSP optimizer based on various heuristic optimization techniques. Our focus is on two methods, namely Genetic Algorithm (GA) and Hill Climbing (HC), which are provided by *mlrose*. We present modifications for both methods that improve the computed tour lengths, by moderately exploiting the problem structure of TSP. However, the proposed improvements have some generic character and are not limited to TSP only.

***Index Terms*—Artificial Intelligence, Traveling Salesperson Problem, Genetic Algorithm, Hill Climbing, Commissioning, Material Flow**

I. Introduction

A. Motivation

In this paper, we investigate the application of methods of AI to an industrial problem on the example of optimizing commissioning tasks in a high-bay storage. Our goal is not to improve on the state of the art in this task but instead shed light on this problem from an AI engineering point of view. From this point of view, we first have to translate this problem adequately to apply methods of artificial intelligence, then we would seek for established software implementations of these methods and evaluate these on the given task of high-bay storage commissioning.

The commissioning problem or order picking problem is the following: We are given a high-bay storage where goods are stored in slots arranged on a two-dimensional wall. An order comprises a finite set of places on that wall that need to be visited to pick up the goods. We desire to do this as quickly as possible. Assuming a tapping point where the collection device starts and ends its job, we can interpret this as an instance of the TSP: Given a set of n locations $p_0, \dots, p_{n-1}$ in the plane, we ask for the shortest closed tour on $\mathbb{R}^2$ that visits all points $p_0, \dots, p_{n-1}$.

What we essentially ask for is the optimal order at which we visit the locations p_i. Furthermore, the way we measure distances between pairs of locations is relevant. To sum up, we consider $p_0, \dots, p_{n-1}$ in a metric space (X, d) with a metric d, encode a tour as a permutation $\pi\colon \{0, \dots, n-1\} \to \{0, \dots, n-1\}$ and ask for a tour π that minimizes the tour length $\ell(\pi)$ with

$$\ell(\pi) = \sum_{i=0}^{n-1} d(p_{\pi(i)}, p_{\pi((i+1) \bmod n)}). \tag{1}$$

In this paper, we may interchangeably represent a permutation π as the sequence $(\pi(0), \dots, \pi(n-1))$, when it fits better to the formal setting.

The metric d models movement characteristics of the collection device. For instance, if vertical and horizontal motions cannot happen simultaneously at the high-bay storage then the Manhattan metric is a reasonable model. If the speeds in horizontal and vertical directions differ then weights in the metric definition can accommodate this circumstance. For the remainder of this paper, we simply assume that d is the Euclidean metric and (X, d) is, therefore, the Euclidean plane. This leads to a case, where many optimization algorithms can be applied. In this paper, we discuss especially the Genetic and Hill Climbing Algorithm and improvements of these algorithms. As there are already libraries for standard implementations of these algorithms, we do not implement the algorithms from scratch, rather we use a library called *mlrose* as a base and improve the above stated algorithm based on this library.

B. Theoretical background and related work

The TSP is a classical problem in operations research and algorithm theory. In the list of 21 NP-complete problems by Karp [1] is the related (undirected) Hamiltonian cycle problem: Given an (undirected) graph with n vertices, is

Stefan Huber was supported by the European Interreg Österreich-Bayern project AB292 KI-Net and Martin Uray by the European Interreg Österreich-Bayern project AB215 DataKMU.

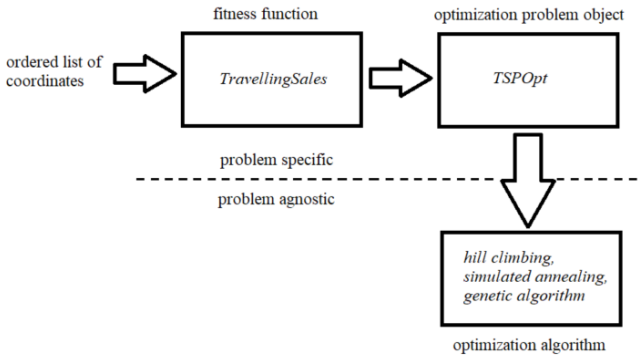


Fig. 1: Overview of solving TSP using *mlrose*, illustrated as a block diagram.

there a Hamiltonian cycle? This problem can be considered to be a special case of (the decision problem of) TSP after choosing the edge weights appropriately and hence (the decision problem of) TSP is NP-complete, too.

The Euclidean plane adds additional structure to the TSP. Although TSP in the Euclidean plane is still NP-complete, the additional structure allows for polynomial-time approximation algorithms: Mitchell [2] and Arora [3] independently presented a polynomial-time approximation scheme polynomial-time approximation scheme (PTAS) for TSP in the Euclidean plane, for which they received the Gödel prize. The algorithm by Christofides is a 1.5-approximation in the metric space [4]. While Christofides algorithm is reasonably simple to implement, it has a time complexity of $O(n^3)$.

Christofides algorithm is closely related to the minimum spanning tree (MST). The Euclidean MST is actually a subgraph of the Delaunay triangulation that can be computed in $O(n \log n)$ time and possesses only $\Theta(n)$ edges. For Euclidean TSP heuristics based on the Delaunay triangulation we refer to [5].

The TSP can be phrased as a problem of AI. Following the notation of Russel and Norvig [6], we deal with a fully observable, single-agent, deterministic, sequential, static, discrete, and known environment. As many NP-complete problems, also TSP has attracted extensive AI research, which essentially spans the full spectrum of the AI landscape, such as logic-based methods, e.g., through Satisfiability Modulo Theories (SMT), machine learning or various, often biologically inspired, optimization techniques. In this paper we will focus on the latter.

There are a few meta-heuristic algorithms out there for approaching the global minimum in a multidimensional space. Many approaches and improvements have already been made to get closer to the global minimum without letting the computation time escalate. Arash Mazidi et al. [7] have combined the Ant colony and the GA to solve a navigation routing problem. For the generation of the initial population, they have used the Ant colony algorithm and for learning and evolving every single node, they used the GA. The results of this paper were not only a better approach to reach the global minimum, moreover, the computation time also decreased slightly in comparison to other algorithms like Particle swarm optimization or Simulated annealing. Sanchez et al. [8] have solved TSP using GA on a GPU. The method they used, was to parallelize the mutation of different nodes on different threads and CUDA blocks. This approach resulted in a significant improvement on the problems computation time. Urrutia et al. [9] have used stack data structure and dynamic programming to solve TSP. Results show that optimal solutions are obtained. Also Reinforcement Learning (RL) and supervised learning are applied for TSP [10]. With RL several variants were proposed, like a hybrid algorithm combining a GA and multi-agent RL [11] as well as Deep RL [12]. With supervised learning, variants, like recurrent neural networks [13], and more recently, also the transformer architecture [14] were proposed to solve the TSP.

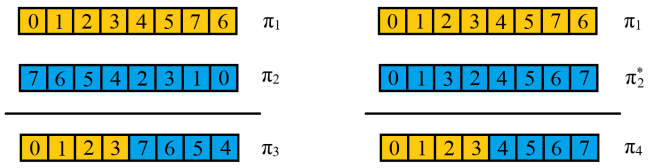
A recent and comprehensive overview on TSP using AI is given by Osaba et al. [15], covering the GA. This work highlights the most notable GA crossover variants.

The HC is comprehensively described in Russel and Norvig, including several modifications to overcome issues, like plateaus or ridges [6]. Additional extensions, like Simulated Annealing, Tabu Search, the Greedy Randomize Adaptive Search Procedure, Variable Neighborhood Search, and the Iterated Local Search support to overcome the local optima problem [16].

To the best of the author's knowledge, the proposed contributions have not been covered in literature so far.

II. Solve TSP using *mlrose*

The python package *mlrose* provides a set of randomized optimization and search algorithms to a range of different optimization problems. This paper deals with the TSP, therefore the necessary steps to solve this problem using *mlrose* will be discussed. Fig. 1 shows an overview of how to solve TSP using *mlrose*. First, a fitness function object has to be defined, which is to be maximized. In the case of the TSP, the fitness function is given by $-\ell$, i.e., maximizing the fitness is equivalent to minimizing the costs expressed by the tour length $\ell(\pi)$ of a tour π. The *TravellingSales* class implemented by *mlrose* provides a fitness function object for that purpose. An object of this class is initialized by input an ordered list of n elements. The *evaluate* method of the *TravellingSales* class receives a state vector as an input parameter and returns the cost of the given state. The state vector is represented by a vector of n integers in the range 0 to $n-1$, specifying the order in which the elements are visited. Once a fitness function object is created, it can be used as an input to an optimization problem object. This object is used to contain the relevant information about the optimization problem to be solved. The *TSPOpt* class implemented by *mlrose* is used. When initializing an object of this class, the *maximize* parameter can be used to determine whether it is a maximization or minimization problem. Finally, an optimization algorithm is selected and used to solve the problem [17]. The optimization algorithms



(a) π_1 and $\pi_2 = \pi_1^*$ recombined to candidate state π_3.

(b) π_1 and π_2^* recombined to candidate state π_4.

Fig. 2: Recombination of exemplary states π_1 and π_2 to candidate states π_3 and π_4.

are problem-agnostic, while the optimization problem object and the fitness function are problem-specific. In the following sections, the randomized optimization algorithms HC and GA are introduced, weak points of the implementation of *mlrose* are pointed out and modifications are carried out to increase the performance of these algorithms.

The precalculated data set *att48* from TSPLIB [18] is used to measure the performance of the standard and the modified algorithms. This data set contains 48 cities in a coordinate system with a minimal tour length of 33523 (unit-less).

III. Genetic Algorithm (GA)

A. General basics

The GA is an optimization and search procedure that is inspired by the maxim "survival of the fittest" in natural evolution. A candidate solution (individual) is encoded by a string over some alphabet (genetic code). Individuals are modified by two genetic operators: (i) random alteration (mutation) of single individuals and (ii) recombination of two parents (crossover) to form offsprings. Given a set of individuals (population), a selection mechanism based on a fitness function together with the two genetic operators produce a sequence of populations (generations). The genetic operators promote exploration of the search space while the selection mechanism attempts to promote the survival of fit individuals over generations. The GA as implemented in *mlrose* terminates after no progress has been made for a certain number of generations or a predefined maximum number of generations.

For GA to work well, it is paramount that a reasonable genetic representation of individuals is used. In particular, the crossover operator needs to have the property that the recombination of two fit parents produces fit offsprings again, otherwise the genetic structure of fit individuals would not survive over generations and GA easily degenerates to a randomized search. Furthermore, the efficiency of GA depends on the initial population, the selection, and the recombination strategy. The mutation rate and the size of the population are hyperparameters.

B. Implementation in mlrose

mlrose uses the state vector representation from section II directly as genetic representation, i.e., an individual is encoded as a permutation sequence π of the integers $0, \ldots, n-1$, which are indices of the n locations to be visited. Recombination of a first parent π_1 and a second parent π_2 works as follows: The sequence π_1 is considered to be split at a random position, the prefix of π_1 is taken and the missing locations in the genetic string are taken from π_2 in the order as they appear in π_2.

Note that TSP has the symmetry property that a solution candidate π and its reverse counterpart π^* can be considered to be the same solutions. Not only is $\ell(\pi) = \ell(\pi^*)$ but in some sense the structure of the solution is the same. The reason behind this is that the pairwise distances between locations in the Euclidean plane (or adequate metric spaces) are invariant with respect to reflection.

However, the recombination strategy does not take this symmetry property into account. This leads to the following problem: Consider the recombination of two parents, π_1 and π_2, that are reasonably similar and fit, however, their direction of traversal is essentially opposite. Then the offspring first traverses the locations like π_1 and then continues with π_2 that possesses the reversed direction, which likely destroys the fit solution structure displayed by π_1 and π_2. That is, two fit parents produce unfit offsprings, as illustrated by Figure 2a.

As an illustrative extreme example, assume π_1 is a globally optimal solution of TSP and $\pi_2 = \pi_1^*$. For sake of argument, assume $\pi_1 = [0, 1, \ldots, 7]$ and $\pi_2 = [7, 6, \ldots, 0]$ as in fig. 2a. Then the offspring π_3 that results from a split in the middle of the genetic string would be $\pi_3 = [0, \ldots, 3, 7, \ldots, 4]$, which is now typically far from globally optimal, i.e., $\ell(\pi_3) \gg \ell(\pi_1)$, see fig. 2a. This recombination would only not hurt if the middle of the fit tour π_1, where the split point of the recombination is located, would happen to be close to the start or end of π_1.

C. Modification

To mitigate the presented issue of the recombination strategy, we would like to have a natural notion of direction of traversal of a tour, so we could figure out whether we would need to reverse the parent π_2 before recombining it with π_1. But since we lack an adequate mathematical notion, we factor out the two possibilities of tour traversals of π_2 in a different way.

When recombining π_1 and π_2, we actually consider two candidate offsprings: offspring π_3 from π_1 and π_2 and offspring π_4 from π_1 and π_2^*. See fig. 2b for an illustration of π_4 for the extreme example from before and fig. 2a for π_3. We then compare the fitness values of the two candidate offsprings, i.e., we compare $\ell(\pi_3)$ and $\ell(\pi_4)$, and keep only the better one as the recombination result. Following our observation from the previous section, we expect that one offspring of two fit parents results from a direction-conforming recombination and the other does not. (Of course, it still can happen that the two parents are bad mates for other reasons, i.e., they can still be structurally insufficiently compatible.)

This way we turn the original recombination operator into a reversal-invariant recombination operator. Note that

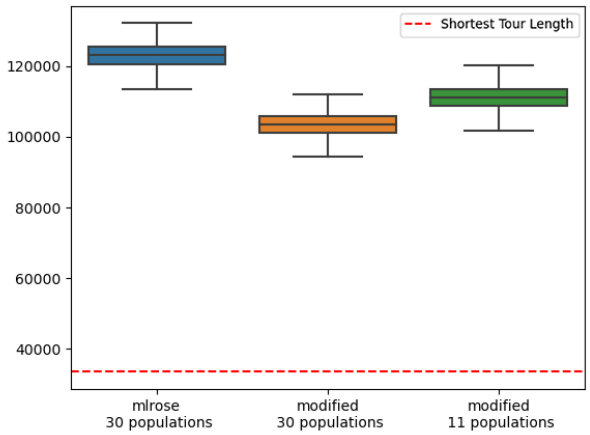


Fig. 3: Box plots on the tour lengths for the experiments with the GA. The optimal solution is depicted by the red dashed line.

our proposed recombination operator is beneficial not only for TSP, but generally for all problems with this reversal symmetry of the genetic encoding of individuals.

In literature other recombination methods, based on crossover [19], [20] and mutation [21], can be found. These recombination operators work differently, but can all be applied additionally within the proposed method, instead of the implemented recombination operator.

D. Results

An experiment was carried out over 1000 attempts to measure the performance of the modified GA in comparison with the implementation by *mlrose*. Both algorithms use a shared parameter configuration of a population size of 200 and a maximum number of 30 generations, if not stated otherwise. While we would set a small positive mutation rate in practice, the higher the mutation rate is, the closer both implementations converge to a random search. Hence, for the sake of comparison, we set the mutation rate to 0.0 for our experiment. All results visually look like Gaussian distributed (not shown in the paper) and hence the notation of mean±standard deviation is used in the following discussion.

The tour lengths of the modified and the original implementation are shown in fig. 3. The red dashed horizontal line marks the optimal solution at a tour length of 33523. The results show, that the modified GA (103278 ± 3370) is closer to the optimum than the original algorithm by *mlrose* (122714 ± 4003).

The gain in the performance by the modified algorithm is at the expense of a higher computation time. The initial algorithm by *mlrose* (5281 ± 350) is faster than the modified GA (14505 ± 666 ms). The modification of the GA results in a mean computation slowdown by a factor of about 2.75, compared to the original implementation by *mlrose*.

For a fair comparison of the computation times between these two implementations, a configuration was empirically determined, where both approaches perform comparably on the computation time. This evaluation uses the CPU clock time, based on a Intel Core i7-7700K (4.20 GHz). For this experiment, the population is set to 11, resulting in a computation time of 5513 ± 403 ms, which is close to the result by the initial implementation by *mlrose*. The tour lengths for this configuration are shown in green in fig. 3 (110888 ± 3689), and still performs significantly better than the original version with a larger population size of 30.

IV. Hill Climbing (HC)

A. General basics

HC is a simple and most widely known optimization technique, cf. [6]. When phrased as a maximization (minimization, resp.) problem of a function f over some domain, Hill Climbing moves stepwise upward (downward, resp.) along the steepest ascent (descent, resp.) until it reaches a local maximum (minimum, resp.) of f. As mentioned in section II, for TSP we minimize ℓ, which equivalently means maximizing the fitness function $f = -\ell$. The domain is given by the transposition graph $G = (V, E)$ over the vertex set V of permutations π, where E contains an edge (π, π') between vertices π and π' iff we can turn π into π' via a single transposition, i.e., the tour π' results from the tour π by swapping two locations only. Figure 4 illustrates the transposition graph G.

Given this formalization, we can speak of a neighborhood of π as the set of permutations adjacent to π within G. Note that each permutation π has $\binom{n}{2}$ neighbors in G. Furthermore, ℓ can be considered to be a scalar field over V with respect to which we can apply HC along the edges of G. The paths traced out by HC on the way to a local optimum within a transposition graph have been studied by Hernando et al. [22]. They show, for instance, that the distance to the local optimum is often not monotonically decreasing for various optimization problems. (However, TSP was not part of their study.)

In more detail, HC starts with a random tour π and calculates the cost $\ell(\pi)$ to be minimized. Then it considers all neighbors of π within G and their costs. If the neighbor π' with minimum costs has a lower cost than $\ell(\pi)$ then it moves to π' and repeats. Otherwise, π constitutes a local minimum and HC either terminates or restarts with a new random tour π, as implemented in *mlrose*. After a given maximum number of restarts, HC returns the best permutation found in all runs.

B. Implementation in mlrose

While vanilla HC is a simple optimization technique, various improvements are known in the literature to overcome different shortcomings, see [6] for an overview.

First of all, HC gets stuck in local optima. While this is no issue for convex (minimization) problems, the magnitude of this issue increases with the number of local optima and their respective sizes of the associated basins, i.e., the set of points from which HC ends up in a given local optimum, see also [22]. For TSP this is an issue and the *mlrose* implementation

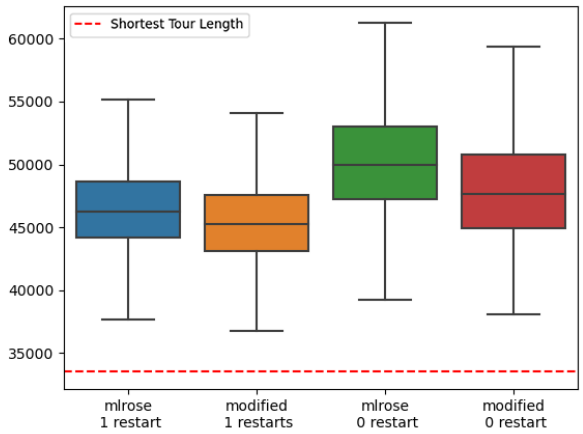


Fig. 5: Box plots on the tour lengths for the experiments with the HC. The optimal solutions is depicted by the red dashed line.

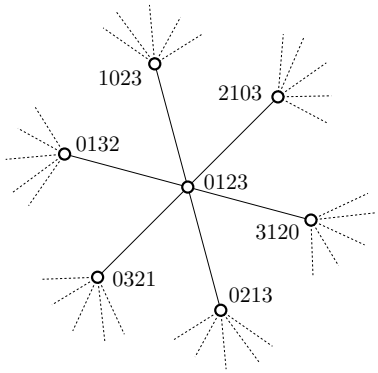


Fig. 4: Part of the transposition graph G for $n = 4$ showing the neighborhood of the permutation 0123.

provides a restart mechanism to mitigate this. A loosely related issue is the presence of ridges where local optima are sort of arranged at a string.

A second well-known issue for HC is the existence of plateaus, i.e., subregions of the domain where the fitness function is constant such that HC is no uphill direction. A mitigation to this issue is to allow a certain number of sideways moves [6] to cross the plateau for the lucky case that an uphill direction does indeed exist at the boundary of the plateau. No such mechanism is implemented in *mlrose*. On the other hand, for TSP it is very unlikely that the best neighbor π' of a permutation π would have the same fitness (tour length) although π and π' differ by a transposition. That is, it is unlikely that the tour length stays the same after swapping two locations.

C. Modification

From the discussion in the previous section, we take away that local optima are the prominent issue for HC on TSP in *mlrose*. In topography, we have the notion of the prominence of a mountain. Vaguely speaking, it tells how far we need to go down from a local maximum to a shoulder from which we can pursue an ascending path that exceeds this local maximum. A more precise notion is given by the persistence of the local maximum in the super-level set filtration in the field of persistent homology, see Huber [23].

Here a natural measure for the prominence of a local maximum is the number of steps in the transposition graph. A prominence of k means that we have to admit k downward steps from a local maximum until we can pursue an ascending path that allows us to escape the local maximum.

In the *mlrose* implementation, HC is already stuck at local maxima with a prominence of only 1, and it would resort to a restart. Our modification simply allows for a single downward step from local maxima to overcome local maxima of prominence 1. If the following step would lead us back to the old local maximum, we terminate this run and apply a restart as the original version. More generally, we simply keep a data structure of previously visited permutations to disallow cycles in the paths traced by HC.

This modification leads to another advantage: In the course of restarts, a series of Hill Climbing searches from randomly generated starting points is performed. After a restart, if the algorithm reaches a state that was visited in a previous trial, the Hill Climbing implemented by *mlrose* will take the same path again and will reach the very same local minimum again. The modified algorithm terminates after an already visited state is reached, which constitutes an early out optimization.

D. Results

An experiment over 1000 attempts was performed to measure the performance of the modified HC in comparison with the HC implementation by *mlrose*. The tour lengths of the modified algorithm and the HC implemented by *mlrose* using 0 and 1 restarts are shown in fig. 5. The results again look visually rather Gaussian distributed (not shown in the paper), such that we use the notation of mean$\pm$standard deviation in the following. The red dashed horizontal line again marks the optimal tour length of 33523.

With 1 restart, the modified algorithm performs best (45420 ± 3340), while the implementation by *mlrose* has a slightly higher overall tour length (46438 ± 3427), which is expected since the modified version effectively extends the exploration.

For further comparison, the restarts parameter of both algorithms (standard HC implementation and modified) are limited to 0 to test the performance against the default configuration of *mlrose*. Here the improvement of the modified version (47944 ± 4348) (red) over the original version (50263 ± 4498) (green) becomes more significant.

The modification influences on the computing time as the modified algorithm is slightly slower (36588 ± 4243 ms) than the HC implemented by *mlrose* (34128 ± 3956 ms). Similar, the results for the experiments with 0 restarts. The modified algorithm has a higher computing time (18273 ± 2980 ms)

than the HC implementation by *mlrose* (16701 ± 2551 ms). Increasing the number of restarts from 0 to 1 gives a slow-down of a factor of 1.9, for the modified and the original implementations likewise.

V. Conclusion and final remarks

This paper was motivated by the industrial application of AI to the industrial problem of optimizing commissioning tasks in a high-bay storage, which translates to the TSP.

We chose *mlrose* for a AI library that already provides optimization routines for TSP and had a closer look at two optimization techniques, namely GA and HC. Moderately exploiting the problem structure of TSP, we propose two improvements, one for the GA and one for the HC, respectively, which improved the mean TSP results by 15% for GA and 3% for HC. The modifications we propose, however, have some generic character and are not only applicable to TSP.

For the GA, a significant improvement on the computed tour length can be shown based on our reversal-invariant crossover operator. This gain comes to the cost of the computational time. However, we show that even if we compensate for additional computational time by reducing the population size, our modification still outperforms the original version of *mlrose*.

For the HC, the goal of the experiment was to show, that a problem-specific treatment is necessary for TSP. The implementation provided by *mlrose* is a problem-agnostic vanilla implementation, that does not offer any problem-specific optimizations. By altering the algorithm towards the properties of the TSP, a clear improvement can be observed.

Finally, we would like to remark that to some extent our paper could be seen as a showcase that AI libraries should only carefully be applied as plug-and-play solutions to industrial problems and the specific problem structure of the industrial problem at hand likely provides means to improve the performance of the generic implementations. While the democratization through meta-learning facilities like AutoML relieve an application engineer from the tedious search for ML methods and their hyperparameters for a given problem at hand, we believe that in general, they do not make an understanding of the underlying methods obsolete.

References

[1] R. Karp, "Reducibility among combinatorial problems," *Complexity of Computer Computations*, vol. 40, pp. 85–103, Jan. 1972.

[2] J. S. B. Mitchell, "Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-mst problem," in *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '96. USA: Society for Industrial and Applied Mathematics, 1996, pp. 402—408.

[3] S. Arora, "Polynomial time approximation schemes for euclidean tsp and other geometric problems," in *Proceedings of 37th Conference on Foundations of Computer Science*, 1996, pp. 2–11.

[4] N. Christofides, "Worst-case analysis of a new heuristic for the travelling salesman problem," Graduate School of Industrial Administration, Carnegie Mellon University, Technical Report 388, 1976.

[5] A. Letchford and N. Pearson, "Good triangulations yield good tours," *Computers & Operations Research*, vol. 35, pp. 638–647, 2008.

[6] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2010.

[7] A. Mazidi, M. Fakhrahmad, and M. Sadreddini, "Meta-heuristic approach to cvrp problem: Local search optimization based on ga and ant colony," *Journal of Advances in Computer Research*, vol. 7, no. 1, pp. 1–22, 2016.

[8] L. Sánchez, "Parallel genetic algorithms on a gpu to solve the travelling salesman problem," *Revista en Ingeniería y Tecnología*, vol. 8, no. 2, 2015.

[9] S. Urrutia, A. Milanés, and A. Løkketangen, "Parallel genetic algorithms on a gpu to solve the travelling salesman problem," *Revista en Ingeniería y Tecnología*, vol. 8, no. 2, 2015.

[10] L. M. Gambardella and M. Dorigo, "Ant-Q: A Reinforcement Learning approach to the traveling salesman problem," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 252–260.

[11] M. M. Alipour, S. N. Razavi, M. R. Feizi Derakhshi, and M. A. Balafar, "A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem," *Neural Comput & Applic*, vol. 30, no. 9, pp. 2935–2951, Nov. 2018.

[12] S. Miki, D. Yamamoto, and H. Ebara, "Applying Deep Learning and Reinforcement Learning to Traveling Salesman Problem," in *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. Southend, United Kingdom: IEEE, Aug. 2018, pp. 65–70.

[13] M. S. Tarkov, "Solving the traveling salesman problem using a recurrent neural network," *Numer. Analys. Appl.*, vol. 8, no. 3, pp. 275–283, Jul. 2015.

[14] X. Bresson and T. Laurent, "The Transformer Network for the Traveling Salesman Problem," *arXiv:2103.03012 [cs]*, Mar. 2021.

[15] E. Osaba, X.-S. Yang, and J. Del Ser, "Traveling salesman problem: a perspective review of recent research and new results with bio-inspired metaheuristics," in *Nature-Inspired Computation and Swarm Intelligence*. Elsevier, 2020, pp. 135–164.

[16] M. A. Al-Betar, "β-Hill climbing: an exploratory local search," *Neural Comput & Applic*, vol. 28, no. S1, pp. 153–168, Dec. 2017.

[17] G. Hayes, "mlrose: Machine Learning, Randomized Optimization and SEarch package for Python," https://github.com/gkhayes/mlrose, 2019.

[18] G. Reinelt, "TSPLIB–a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[19] A. Roy, A. Manna, and S. Maity, "A novel memetic genetic algorithm for solving traveling salesman problem based on multi-parent crossover technique," *Decis. Mak. Appl. Manag. Eng.*, vol. 2, no. 2, Oct. 2019.

[20] A. Hussain, Y. S. Muhammad, M. Nauman Sajid, I. Hussain, A. Mohamd Shoukry, and S. Gani, "Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator," *Computational Intelligence and Neuroscience*, vol. 2017, pp. 1–7, 2017.

[21] O. Abdoun, J. Abouchabaka, and C. Tajani, "Analyzing the Performance of Mutation Operators to Solve the Travelling Salesman Problem," *arXiv:1203.3099 [cs]*, Mar. 2012.

[22] L. Hernando, A. Mendiburu, and J. A. Lozano, "Hill-Climbing Algorithm: Let's Go for a Walk Before Finding the Optimum," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. Rio de Janeiro: IEEE, Jul. 2018, pp. 1–7.

[23] S. Huber, "Persistent homology in data science," in *Proc. 3rd Int. Data Sci. Conf. (iDSC '20)*, ser. Data Science – Analytics and Applications, Dornbirn, Austria (virtual), May 2020.