Expert Systems With Applications 187 (2022) 115948

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Improving the state-of-the-art in the Traveling Salesman Problem: An Anytime Automatic Algorithm Selection

Isaías I. Huerta [a], Daniel A. Neira [b], Daniel A. Ortega [a], Vicente Varas [a], Julio Godoy [a,*], Roberto Asín-Achá [a]

[a] Universidad de Concepción, Facultad de Ingeniería, Departamento de Ingeniería Informática y Cs. de la Computación, Concepción, Chile
[b] Universidad de Concepción, Facultad de Ingeniería, Departamento de Ingeniería Industrial, Concepción, Chile

ARTICLE INFO

Keywords:
Combinatorial optimization
TSP
Algorithm selection problem
Anytime behavior approach

ABSTRACT

This work presents a new metaheuristic for the euclidean Traveling Salesman Problem (TSP) based on an Anytime Automatic Algorithm Selection model using a portfolio of five state-of-the-art solvers. We introduce a new spatial representation of nodes, in the form of a matrix grid, avoiding costly calculation of features. Furthermore, we use a new compact *staggered representation* for the ranking of algorithms at each time step. Then, we feed inputs (matrix grid) and outputs (staggered representation) into a classifying convolutional neural network to predict the ranking of the solvers at a given time. We use the available datasets for TSP and generate new instances to augment their number, reaching 6,689 instances, distributed into training and test sets. Results show that the time required to predict the best solver is drastically reduced in comparison to previous traditional feature selection and machine learning methods. Furthermore, the prediction can be obtained at any time and, on average, the metasolver is better than running all the solvers separately on all the datasets, obtaining 79.8% accuracy.

1. Introduction

Combinatorial optimization problems arise in many real world applications, for example, in cloud computing resources deployment, optimal path search for transport, computational modeling of proteins, microchip manufacturing, X-ray crystallography, scheduling in schools and universities, and lately, in drone routing (Agatz, Bouman, & Schmidt, 2018; Davendra, 2010; MirHassani & Habibi, 2013). Although many of these problems are NP-hard, they need to be solved with two objectives in mind, often opposite: a good quality solution and a brief solving time. Usually, a small improvement in the quality of the solution or reduction of the execution time can save millions of dollars or a significant increase in productivity.

For many combinatorial optimization problems there is a wide corpus of research papers that present new solution approaches and report advances in the field. However, most of the reported performance improvements are linked to specific datasets or characterization of the problems. In most cases, when tested in different datasets, these performance claims disappear. This is known as 'No free lunch' Theorem (Wolpert & Macready, 1997). Therefore, different solving techniques or algorithms perform in a mixed way when applied to real-world benchmarks, and their behavior depends on *characteristics* or *features* of the benchmark (Guerri & Milano, 2004).

Combining multiple algorithms that solve the same problem can be achieved in practice by having a *predicting oracle* that knows, from a set of algorithms, which one of them is more suitable to solve a given instance. From now on, we will refer to this oracle as *metasolver*, being an algorithm composed of two stages: i) a predictive stage in which the best algorithm is selected, and then ii) the resolution of the instance using that algorithm.

The problem of choosing the best possible algorithm for a specific instance of a problem has been stated in the machine learning community and is known as the Automatic Algorithm Selection Problem (Rice et al., 1976). Usually, such problems are seen as classification problems (Kotthoff, 2016), where the goal is to learn the behavior of different solving algorithms (from now on, solvers) over a set of instances, that should be as diverse as possible, and generalize it for unknown instances. Given a specific instance, the metasolver will tell which solver gives the best solution.

In general, the metasolver will only consider the best solution found in a fixed runtime. This approach is ill-suited when a specific application imposes constraints on the available time to solve an instance,

* Corresponding author.
E-mail addresses: ihuerta@inf.udec.cl (I.I. Huerta), danieneira@udec.cl (D.A. Neira), daortega@udec.cl (D.A. Ortega), vvaras@inf.udec.cl (V. Varas), juliogodoy@udec.cl (J. Godoy), rasin@inf.udec.cl (R. Asín-Achá).

https://doi.org/10.1016/j.eswa.2021.115948
Received 16 March 2021; Received in revised form 8 September 2021; Accepted 19 September 2021
Available online 6 October 2021

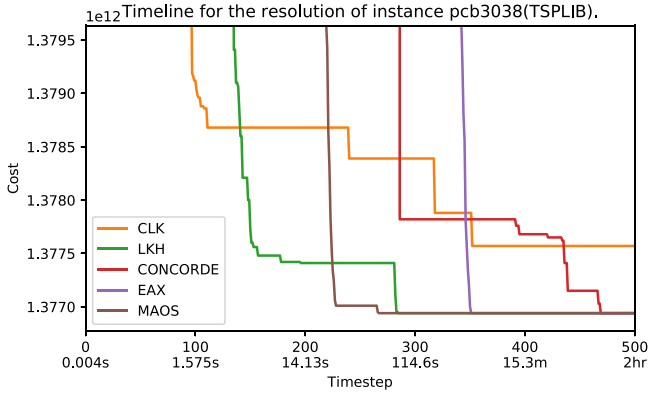

Fig. 1. Temporal behavior of five TSP solvers for a particular instance..

because the prediction could choose a solver that needs more than the considered time to achieve the sought objective. For instance, when looking for a solution with very little available time, a greedy algorithm could yield the best result, whereas if we had more time, the optimal objective value could be obtained using an exact algorithm.

Fig. 1 shows the temporal behavior of five solvers when dealing with an instance of the TSPLIB library (pcb3038). The x axis represents the time steps, while the y axis shows the cost of the solution. When observing different points along the x axis, we can see that the solver with the best solution changes over time. For example, before 5 s, the winner is CLK, then until around 20 s the winner is LKH and after that the best solution is found by MAOS.

In a previous work that deals with the Knapsack problem (Huerta et al., 2020), the authors propose a different approach to tackle Anytime Automatic Algorithm Selection. Results showed that the use of machine learning techniques is capable of predicting the best solver when an instance was given and constraints on the time limit were considered. Although these results were interesting, most solvers were capable of obtaining solutions in a very short time (under one second), and therefore, the time taken to predict the best solver for the given conditions was considerable in comparison to the execution time of the solvers, mainly due to the time required to compute the instance features, such as the mean, median or Pearson coefficient of the input values and weights. The short time required to solve large instances of the knapsack problem can be credited to its Weakly NP-complete nature, and the fact that the instances used for this problem generally use fixed representation numbers and not arbitrary precision, which allows for harder instance configuration.

This research aims to develop an Anytime Automatic Algorithm Selection for the well known and widely studied euclidean Traveling Salesman Problem (TSP). This is an NP-hard problem and, unlike the knapsack problem, even though there are pseudo-polynomial exact algorithms or arbitrary approximation algorithms (Arora, 1996, 1997; Mitchelly, 1997), solving it requires considerable computational resources. This makes it more interesting in an anytime framework, since execution times are longer and therefore deciding over which solver to use is more relevant and is proportionally negligible to the total execution time. The main question that this work seeks to answer is: Given an instance i and a time limit t, which solver should be used to solve the problem?

The main contributions of this work are:

(i) Proposing and testing a new metaheuristic for the euclidean TSP based on an Anytime Automatic Algorithm Selection model using a state-of-the-art set of solvers.

(ii) Developing a new convolutional neural network based classifier algorithm for the automatic selection of solvers for the euclidean TSP.

(iii) Presenting and testing a new spatial representation, in the form of a matrix grid, which can be used as input in a convolutional neural network, avoiding the costly calculation of features.

(iv) Designing a new compact representation for a specific instance solution that, on one hand, keeps the ranking of the solvers along time and, on the other hand, can be used as the output of a classifying neural network.

(v) Making available to the community 6689 new instances and their solutions for future work related to Automatic Algorithm Selection on TSP.

2. Preliminaries

In this section, concepts and issues relevant for the development of the metasolver are reviewed.

2.1. Traveling salesman problem

The Traveling Salesman Problem (TSP) is a classic NP-hard combinatorial optimization problem (Garey, Graham, & Johnson, 1976; Papadimitriou, 1997) that has been widely studied because it is easy to describe and instantiate, and at the same time is computationally challenging. Also, it is present or related to multiple practical applications and in simplifications of complex multidisciplinary problems (Davendra, 2010, Chapter 1). It is usually stated as follows: Given a set of n cities, a salesman wants to visit each one of them one time only, starting at any city and ending his path on the same one. The TSP looks to answer the question: Which route should the salesman take to travel the least possible distance? We assume that the distances for any pair of cities are known and can be conceptualized as money or time. From now on, the distances will be referred to as **costs**. Mathematically, the problem can be stated as: Given a set of cities $1, 2, \dots, n$ and a cost matrix $C = c_{ij}$, in which c_{ij} represents the cost of going from city i to city j, with $i, j = (1, 2, \dots, n)$, find a permutation $\{i_1, i_2, \dots, i_n\}$ of cities that minimizes the sum $c_{i_1 i_2} + c_{i_2 i_3} + c_{i_3 i_4} + \cdots + c_{i_{n-1} i_n} + c_{i_n i_1}$,where $i_j i_{j+1}$ represent the indexes of consecutive cities in the permutation.

The properties of the cost matrix can be used to characterize the problem:

(i) If $c_{ij} = c_{ji}$ for every i and j, the problem is **symmetric** , and otherwise is **asymmetric**.

(ii) If the triangular inequality is satisfied for the intercity distances ($c_{ik} \leq c_{ij} + c_{jk}$ for every i, j and k), then the problem is **metric**.

(iii) If c_{ij} are euclidean distances between points in the plane, the problem is **euclidean**. This problem is both symmetric and metric.

2.2. Automatic algorithm selection

Consider the algorithm selection problem for a decision or optimization problem P. Specifically, the algorithm selection by instance can be constructed as follows Rice et al. (1976): Given a set of instances I of problem P, a set $A = \{A_1, \dots, A_n\}$ of algorithms that solve P, and a metric $m : A \times I \to \mathbb{R}$ that measures the performance of any algorithm $A_j \in A$ over the set of instances I, the goal is to build a selector S that maps any instance $i \in I$ to an algorithm $S(i) \in A$ such that the average performance of S over I is optimal according to the metric m.

In general, the selector S can be seen as a machine learning classifier that requires a fixed size characterization of the instances as input. For this, a fixed set of N_f characteristics $F(i) = \{f_1(i), \dots, f_{N_f}(i)\}$ that represent each instance $i \in I$ should be defined. The functions in $F(i)$ must be carefully chosen to be as informative and as easy to compute as possible.

Fig. 2 shows the stages involved in the Automatic Algorithm Selection process (AASP). Once the problem is chosen several decisions must be made:

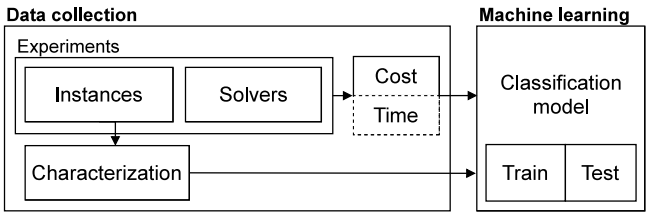

Fig. 2. General scheme of the AASP. Time is shown with dotted lines because it can be used in an *anytime* approach or just as a time limit for the execution.

(i) A set of solvers and instances must be selected to get the data for training and test.
(ii) Instance characterization: The set of features or representations of the instances must be defined.
(iii) Finally, the machine learning techniques to be used must be chosen.

Other important decisions are related to the available time and computational resources. In principle, one should use as many solvers as possible to account for different behaviors in specific domains and, in the same sense, a wide range of instances. Nonetheless, since we are dealing with an NP-hard problem, running every instance-solver pair until termination can be very costly.

2.3. Anytime automatic algorithm selection

Extending the definition given by Rice, Anytime Automatic Algorithm Selection can be defined as follows:

Given a set I of instances for the problem P, a set $A = \{A_1, \dots, A_n\}$ of algorithms that solve P, a set of k time steps $T \in \{t_1, t_2, \dots, t_k\}$ and a metric $m : (A \times I) \times T \rightarrow \mathbb{R}$ that measures the performance of any algorithm $A_j \in A$ over the set of instances I in the time steps T, the goal is to build a selector S that maps any pair $(i \in I, t_j \in T)$ to an algorithm $S(i, t_j) \in A$ such that the average performance of S over the pair (I, T) is optimal according to the metric m.

The additional temporal variable $T \in \{t_1, t_2, \dots, t_k\}$ that arises on this definition must be described depending on the nature of the problem, considering the time span $[t_1, t_k]$, the granularity of the time steps (value of k) and the distance between each time step t_i and t_{i+1}.

2.4. Deep learning

Neural networks are computational models inspired by the behavior of neurons in biological systems. They are composed of nodes (artificial neurons) that are connected with each other in a layered structure. The objective of a supervised neural network is to learn how to solve a task from examples given in a training stage (Hassoun et al., 1995). At this stage, the weights of the connections between the nodes are adjusted through the back-propagation algorithm (Kelley, 1960), that aims to reduce the loss or the cost function. Depending on the nature of the data to predict, a different type of neural network can be chosen. If the predicted value is discrete, then the network is a classifier, and when the output is continuous the network is a regressor.

Neural networks have been used to solve problems from different domains such as computer vision, voice recognition, text generation and recommender systems, achieving state-of-the-art results in many applications like object detection (Bochkovskiy, Wang, & Liao, 2020), speech recognition (Zhang et al., 2020) and NLP tasks (Brown et al., 2020) . There are different common neural network architectures that have led to gains in precision in various applications, some of them are convolutional architectures (CNN) (Fukushima & Miyake, 1982), recurrent neural networks (RNN) (Rumelhart, Hinton, & Williams, 1986) and transformers (Vaswani et al., 2017) which have improved the results in tasks regarding images, sequential data and text, respectively. The appropriate architecture of the network depends of the required task.

3. Literature review

Here we review the main algorithms for TSP and their characteristics. We also elaborate on related work that can be found in the literature regarding Automatic Algorithm Selection for the TSP.

3.1. TSP solvers

Exact algorithms are guaranteed to find the optimal solution for any instance and many of them can give an optimality proof when their execution ends, but with a high runtime requirement. The most effective exact algorithms are the *cutting-plane* and *facet-finding* algorithms proposed in Applegate, Bixby, Chvátal, and Cook (1995), Grötschel and Holland (1991) and Padberg and Rinaldi (1991). They are highly complex and require extensive computational resources. The best known one is Concorde (Applegate, Bixby, Chvatal, & Cook, 2006a), which is based on a sophisticated form of branch & cut, but because of its exponential worst case execution time, it is not well suited for large instances.

On the other hand, **approximation algorithms** aim to give a good solution in a short time, but this solution might not be optimal, although, in practice, some of them manage to deliver solutions that are very close to optimality. Many of these algorithms are used on applications where sub-optimal solutions may be acceptable (Rego, Gamboa, Glover, & Osterman, 2011) imposing strict constraints on the running time of the solver.

Approximation algorithms can be divided in three sub-groups: Tour construction algorithms, tour improving algorithms and compound algorithms (Applegate, Bixby, Chvatal, & Cook, 2006b). Tour construction algorithms work gradually by adding one city to the tour in each step. Improving algorithms change an existing tour aiming to improve it and compound algorithms combine both approaches.

The simplest example of a tour construction algorithm is the nearest neighbor algorithm. It starts in any city and then adds the closest city until there are no cities left to visit and then returns to the first one. This greedy algorithm is simple and fast, but has results that are usually far from the optimal value, with the first cities being very close to each other but later ones further apart. There have been many attempts to improve this algorithm (Laporte, 1992; Lawler, 1985; Reinelt, 1994).

Tour improving algorithms have been more successful. An example of this is the 2-opt algorithm. It begins with a given tour, then swaps two links of the path in a way such that the resulting path is shorter than the previous one and repeats this process until the path cannot be improved any further. A generalization of this algorithm is the λ-opt algorithm, based on the concept of λ-optimality:

> *A tour is λ-optimal (or λ-opt) if it is not possible to obtain a shorter tour by swapping any of its λ links for another λ links.*

From this definition it follows that any λ-optimal tour is also λ'-optimal for $1 \leq \lambda' \leq \lambda$. One disadvantage of this algorithm is that the λ value must be previously defined, considering that it will affect the execution time of the algorithm and quality of the solution.

The Lin–Kernighan algorithm (LK) (Lin & Kernighan, 1973) gets rid of this downside by allowing the value of λ to change during runtime. Following, Helsgaun added several changes to its implementation (LKH) (Helsgaun, 2000), mainly in the search strategy, allowing for larger and more complex steps when searching for a new solution. Furthermore, sensibility analysis was included to direct or restrict the search. The execution times of both algorithms grow at a rate of $\mathcal{O}(n^{2.2})$, nevertheless, LKH is more effective at finding optimal solutions in bigger problems. The latest improvement to this algorithm (POPMUSIC) speeds up the generation of candidate tour sets for large instances, obtaining high quality candidates in close to linear time (Helsgaun, 2018).

Another algorithm that uses the Lin–Kernighan idea as base is the Chained Lin–Kernighan algorithm (CLK) (Applegate, Cook, & Rohe,

2003). This algorithm sacrifices in part the quality of the LK algorithm to improve the execution time. Another variation consists of interchanging 4 edges (*double-bridge*) which allows solutions that LK would not have found. Another improvement to CLK is to pay special attention to the initial solution, so to start on a convenient state. This has a big impact, specially on large instances. CLK is used to find an initial solution in Concorde.

Several nature-inspired local search metaheuristics have been proposed to get approximated results for TSP (Kongkaew & Pichitlamken, 2014; Osaba, Yang, & Del Ser, 2020; Scholz, 2019). Some of them are Ant Colony Optimization (ACO) (Dorigo, Maniezzo, & Colorni, 1996; Stützle & Hoos, 2000), Particle Swarm Optimization (Eberhart, Shi, & Kennedy, 2001; Shi, Liang, Lee, Lu, & Wang, 2007), discrete bat algorithm (Saji & Barkatou, 2021), Discrete symbiotic organisms (Ezugwu & Adewumi, 2017), Harmony Search algorithm (Boryczka & Szwarc, 2019) and Genetic Algorithms (Baraglia, Hidalgo, & Perego, 2001; Merz & Freisleben, 2001; Sampson, 1976; Tsai, Yang, Tsai, & Kao, 2004). These last ones have been the most successful ones when they incorporate variants of Edge Assembly Crossover (EAX) (Nagata & Kobayashi, 1997, 2013): a recombination operator that uses edges from two parent solutions and adds short edges that were not initially considered in them. The first solver to use this operator was EAX, achieving a similar performance to LKH.

Another nature inspired algorithm is Multi-Agent Optimization System (MAOS) (Xie & Liu, 2008), that unlike the previously mentioned algorithms, is based on a multi-agent framework in which all agents have limited knowledge of the instance and explore possible solutions simultaneously. The agents update the shared environment and transmit their knowledge to other agents to improve the search process. Initially, each agent begins with a defined structure, which may be a sub-graph with its nearest neighbors. Then, Markov chains are used to generate a set of states.

In the latest years the usage of neural networks has also been explored for the TSP (Potvin, 1993). Some examples of this are Elastic Net (Chen, Zhang, & Chen, 2007), Self-Organizing Map (SOM) (Brocki & Koržinek, 2007) and the Hopfield–Tank network (Andresol, Gendreau, & Potvin, 1999). Despite the advancements and improvements, their results are not yet comparable to the state-of-the-art-solvers (La Maire & Mladenov, 2012; Potvin, 1993; Yang & Whinston, 2020). With the increase in, both, GPU and CPU parallel computational capacity, algorithms that take advantage of this technology and manage to reduce solution time of traditional algorithms in at least an order of magnitude have been developed (Al-Adwan, Mahafzah, & Sharieh, 2018; Al-Adwan, Sharieh, & Mahafzah, 2019; Fujimoto & Tsutsui, 2010; Mahafzah, 2014; Rocki & Suda, 2013).

3.2. Automatic algorithm selection for the TSP

One of the first successful Automatic Algorithm Selection systems was SATzilla (Xu, Hutter, Hoos, & Leyton-Brown, 2008), which for many years defined the state-of-the-art for the Boolean Satisfiability Problem (SAT), one of the most studied NP-complete problems. Because of this success, other related problems such as MaxSAT were also tackled with this approach (Malitsky, Sabharwal, Samulowitz, & Sellmann, 2013). It has also been applied to other computationally challenging solution approaches such as constraint programming (Kadioglu, Malitsky, Sabharwal, Samulowitz, & Sellmann, 2011), continuous black-box optimization (Kerschke & Trautmann, 2019), mixed integer programming (Xu, Hutter, Hoos, & Leyton-Brown, 2011) and AI planning (Vallati, Chrpa, & Kitchin, 2014).

In Huerta et al. (2020) the AAS problem was extended with an *anytime* approach applied to the Knapsack problem. In this research, three ways to present the problem using machine learning were proposed, based on the representation of the solution of an instance:

(i) Classification model (Best solver): for each time step only the label of the solver that has the best solution is considered.
(ii) Classification model (Ranking): the order of the solvers according to their objective value is considered for each time step.
(iii) Regression model (Closeness to the best solution): at each time step the obtained objective value for each solver is obtained and normalized to values between 0 and 1, where the maximum represents the best value obtained at the end of the execution for all solvers.

These models were applied to two sets of solvers, one with classic naive solutions and the other one with state-of-the-art solvers. The best results were obtained with the classifying model focused on the best solver. Nonetheless, the performance of predicting the best solver with the classifier and then running the selected solver is not competitive against the best solver, since the time for computing the features and running the classifier is not negligible.

To the best of our knowledge, three studies on ASS on the TSP can be found in the literature (Fukunaga, 2000; Kerschke, Kotthoff, Bossek, Hoos, & Trautmann, 2018; Pihera & Musliu, 2014). Fukunaga (2000) has an anytime approach for a portfolio of parameters of a genetic algorithm. In this case, the purpose of the Automatic Algorithm Selection is focused on selecting the best parameters of the genetic algorithm for different instances (Automatic Algorithm Configuration (Hutter, Hoos, Leyton-Brown, & Stützle, 2009)). The execution time is considered as an important feature when selecting the best algorithm. Ten uniformly random instances where generated with 40 cities and 54 different parameter configurations of the genetic algorithm were used. For six of the ten instances the model surpasses the performance of the best solver used.

Pihera and Musliu (2014) focus mainly on the construction of appropriate features for AAS applied to TSP. In this case, two solvers where used: LKH and MAOS, with a data set of 2163 instances. Five machine learning classifiers were trained, with *Decision Trees* and *Bayesian networks* obtaining the best results. Two instants in time where used to compare performance: at 60 s and at 1800 s, and the algorithm selection was successful without considering time as a feature. The authors propose, as future work, to reduce the number of features in order to reduce their calculation cost.

Kerschke et al. (2018) proposes an AAS model with the goal of improving the state-of-the-art. Five state-of-the-art solvers are considered: LKH, LKH+restart, EAX, EAX+restart and MAOS, and six sets of instances where used: TSPLIB, National, VLSI, RUE, Netgen and Morphed. The time limit for the resolution of the instances was set to one hour. Three approaches for Automatic Algorithm Selection were explored: classification, regression and *paired regression*, while using multiple machine learning techniques for each one. In the classification approach each instance was labeled with the best performing algorithm. In the regression approach five models are trained which predict the execution time of each solver. Then the solver with the shortest predicted execution time is selected. In the last approach the performance difference between solvers is predicted pairwise. The winning solver is the one with best performance difference to all the other solvers. An important part of this work is the sensibility analysis in the feature selection, because it leads to good results while having a small amount of features to compute. Results show that if the time needed to compute the features is overlooked, the state-of-the-art for approximation algorithms is improved for all data sets. However, if this time is considered, the best selector is an improvement in only three of the six data sets.

As can be seen, these three studies show considerable differences between each other, and focus on different characteristics: on the one hand the work of Fukunaga (2000) uses automatic selection to better define the parameters of an algorithm for each instance. This kind of work is also known as automatic algorithm configuration.[1]

[1] https://cran.r-project.org/web/packages/irace/index.html.

Pihera and Musliu (2014) focus on an extensive feature selection that maximizes the descriptive information of the instances, while Kerschke et al. (2018) attempt to improve the state-of-the-art through a compact feature set and machine learning models. The methodology of this work is more closely related to the last work (Kerschke et al., 2018) because of the used solvers and instances, although the anytime approach and the usage of neural networks for the model diverge from the more traditional AAS methodology. Furthermore, the number of instances we use increases from 1845 to 6689 and the exact solver Concorde is added.

4. Data collection

In this section we elaborate on the characteristics of the chosen solvers (Section 4.1), the instances (Section 4.2) we used and highlight some interesting characteristics of the data set which records the anytime behavior of all the solvers on all the instances.

4.1. Selected state-of-the-art solvers

We use five solvers considered state-of-the-art for the TSP until 2019. Initially, we had considered other interesting solvers such as Self-Organizing Maps[2] (Brocki & Koržinek, 2007), ACO[3] (Gavidia-Calderon & Castañon, 2020) and OR-Tools[4] (Perron & Furnon, 2019), however, preliminary tests showed that their performance is not comparable to the other five. Also, there are other state-of-the-art solvers that use parallelism but were not considered in order to focus our study on the performance of sequential solvers (Fujimoto & Tsutsui, 2010; O'Neil, Tamir, & Burtscher, 2011). Therefore, the solvers considered for this work are:

- Concorde: This is the state-of-the-art in exact solvers for TSP (Applegate et al., 2006a). It uses the Branch & Bound search strategy, as well as cutting planes (Branch & Cut) to reduce the search space. A good initial solution is found using CLK. In addition to generating and exploring limits, Concorde uses cutting planes to narrow and focus the search space. The code used for testing this solver is implemented in ANSI C and was last updated in 2003.[5]
- Chained Lin Kernighan (CLK): Proposed by Applegate et al. (2003), CLK is a form of iterative local search. Within an iteration, it find a local optimum with an imprecise quality because it uses a k-opt neighborhood. When it cannot find an improvement for a solution (i.e. it gets stuck in a local optimum), it restarts by disturbing the current solution. This disturbance consists of deleting intersecting edges, also known as *double-bridge*. The code used can be found as part of the Concorde library. It is implemented in ANSI C and was last updated in 2003.
- Lin Kernighan Helsgaun (LKH): LKH is an improvement over the Lin Kernighan algorithm, proposed by Helsgaun (2000). It has been shown that with these improvements the algorithm gets closer to the optimal value in some instances, however, this also increases its execution times. Among the novelties introduced by Helsgaun are sequential 5-opt moves and sensibility analysis in direct searches. The version of the algorithm used in this work is the last one proposed, which includes a new heuristic to generate the set of candidates called POPMUSIC (Helsgaun, 2018). The code[6] is implemented in C and was last updated in 2019.
- Edge Assembly Crossover (EAX): This is a genetic algorithm based on Edge Assembly Crossover (Nagata & Kobayashi, 2013), and it is considered one of the most effective recombination operators for the TSP. The usage of EAX allows for improvement in aspects such as localization, achieving higher effectiveness through the usage of local search algorithms to construct good parent solutions and therefore better children solutions. EAX generates this offspring by combining edges of two parents and adding some short edges, which are determined by a simple search procedure. The EAX code[7] used in this work is implemented in C++ and was last updated in 2018.
- Multiagent Optimization System (MAOS): This method is based on self-organizing agents, that work with limited declarative knowledge and procedures under ecologic reasoning (Xie & Liu, 2008). Specifically, the agents explore in parallel, based on Socially Biased Individual Learning (SBIL) and interact indirectly with other agents through public environmentally organized information. The MAOS code[8] used in this work is implemented in Java and was last updated in 2017.

The codes of the different solvers were modified only to standardize the instance input and to generate an output every time an improved solution is found, saving the new objective value and the elapsed time. In the case of Concorde, the best solution found is determined by the upper bound of the Branch & Cut algorithm. Default parameter values are used in each solver. A summary of the these implementations can be found in Table 1, where the column *Solver* stands for the algorithm, *Identifier* is the acronym/name by which it will be referred to throughout this paper; *Authors* refers to the authors of the algorithm; *Exact/Heuristic* defines whether it is an exact or heuristic algorithm; *Algorithm* refers to the type of algorithms used and *Language* is the programming language in which it is implemented.

4.2. Selected and generated instances

The instances used in this work can be divided in two groups: training and test. Training instances were generated by existing generators found in the literature, while test instances are known public instances. It should be noted that all these are euclidean instances (2D).

We used up to 6331 training instances corresponding to four sets: RUE, NETGEN, NETGENM and TSPGEN. The instances from RUE, NETGEN and NETGEN are the same used in Kerschke et al. (2018) and are publicly available.[9]

- **RUE** (Random Uniform Euclidean): There are 3150 instances obtained by placing n random points in a square. The range of integer values for the coordinates goes from 1 to 1,000,000. These instances were generated by Kerschke et al. (2018) using the *portgen* generator from the 8th DIMACS implementation contest.[10]
- **NETGEN**: Consists of 600 instances with their cities placed in 2, 5 or 10 clusters. It can be divided in 4 groups that contain 500, 1000, 1500 and 2000 cities. They were generated from the netgen generator, which is available as an R library,[11] using the function *generateClusteredNetwork()*. This function first places 2, 5 or 10 points in the space using an LHS sampling (McKay, Beckman, & Conover, 2000), then other points are sampled from normal distributions using the initial points as mean.

[2] https://diego.codes/post/som-tsp/.
[3] https://github.com/cptanalatriste/isula.
[4] https://developers.google.com/optimization/routing/tsp.
[5] http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm.
[6] http://akira.ruc.dk/~keld/research/LKH/.
[7] https://github.com/british-sense/Edge-Assembly-Crossover.
[8] https://github.com/wiomax/MAOS-TSP.
[9] https://tspalgsel.github.io/.
[10] http://dimacs.rutgers.edu/archive/Challenges/TSP/download.html.
[11] https://rdrr.io/cran/netgen/.

Table 1
Summary of solvers used.

Solver	Identifier	Authors	Exact/Heuristic	Algorithm	Language
Concorde	Concorde	Applegate et al. (2006a)	Exact	Branch & Bound(/Cut)	C
Chained Lin Kernighan	CLK	Applegate et al. (2003)	Heuristic	Local Search	C
Lin Kernighan Helsgaun	LKH	Helsgaun (2000)	Heuristic	Iterated Local Search	C
Edge Assembly Crossover	EAX	Nagata and Kobayashi (2013)	Heuristic	Genetic Algorithm	C++
Multiagent Optimization System	MAOS	Xie and Liu (2008)	Heuristic	Multiagent Optimization System	Java

- **NETGENM**: They are 600 instances that combine characteristics of RUE and NETGEN. They were generated using the *morphInstances* from the R library netgen with a *morphing* factor of $\alpha = 0.5$. The algorithm of this function has two stages. First, given two instances, one from RUE and another from NETGEN, each point p_1 from an instance is linked with the closest point of the other instance p_2. Then each of the linked points is replaced by another point $p_3 = (\alpha p_1 + (1 - \alpha)p_2)$. Note that $\alpha = 1$ would keep the RUE instance and $\alpha = 0$ would keep the NETGEN instance.
- **TSPGEN**: It has 1981 instances created by the TSPGEN generator[12] (Bossek et al., 2019), an evolution of the netgen generator used in the previously mentioned generators. This new generator is designed specifically to create varied instances that support the work of AAS over TSP. To generate the instances, an evolutionary algorithm framework is used, starting from a RUE instance and combining several mutation operators in a random and iterative way.

The **test instances** are 358 publicly available instances for competition to compare and test different TSP solvers. In many cases, the optimum value is known because it has been obtained by exact algorithms, however, in other cases it is unknown and only the best known upper bound is present.

- TSPLIB: It is a set of 88 instances from different sources and types. It is the most popular data due to the variety of its instances. The instances were obtained from the website of the Heidelberg University.[13]
- TNM: This is a set of 141 instances obtained from a tetrahedron instance generator, that has a higher difficulty for exact algorithms. The instances were downloaded from the creator's website.[14]
- NATIONAL: It corresponds to 27 subsets of the instances in World TSP grouped by country. Each instance is related to a country and the nodes are the main cities of the country. Only three instances have not yet been solved optimally. The data set was downloaded from the TSP website.[15]
- VLSI: This is a set of 102 instances of very large-scale integration (VLSI) of integrated circuits created by the Bonn University in Germany.[16] The TSP is relevant for this type of application because it allows for the construction of better performing chips and microprocessors.

Table 2 is a summary of the main characteristics of the data sets.

4.3. Solver-instance execution

Each of the solvers was executed with every instance. As mentioned above, the codes were only slightly modified to standardize the input and output. For every instance-solver pair, an output file with a sequence of pairs, consisting of costs and their associated execution time was created. A new pair is outputted each time the solver finds a better solution. The instances and results of the executions are openly available.[17]

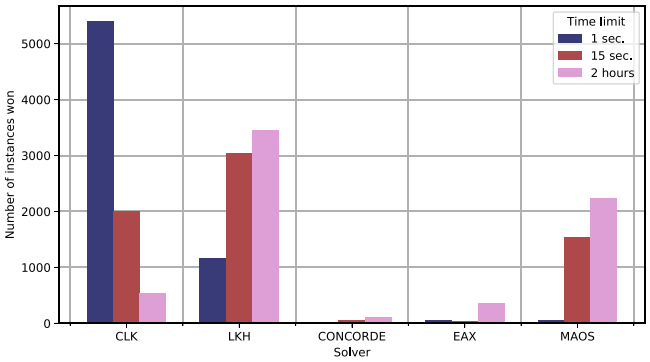

Fig. 3. Evolution of the winning solver versus execution time.

Fig. 3 shows how the best solver, defined as the one performing better in most instances, changes over time. As an example of how the best solver can vary depending on the time, we recall Fig. 1 which shows the anytime behavior of all solvers for instance "pcb3038" from TSPLIB. Fig. 4 illustrates such anytime behavior for each dataset. There, different colors represent the solvers that lead the search, the x axis depicts the instances in an ascending order according to the number of cities. The y axis represents the time in seconds. The Greedy solver (HK-Christofides algorithm described in Applegate et al. (2003)) is included, which is also the first solution for CLK. The instances of the TNM dataset are hard to solve for Concorde, hence no victories for this solver can be seen. It is clear that in most of the instances CLK gets the best results at first, then LKH is dominating at intermediate times and MAOS improves at the later stages of the execution.

4.4. Experimental setup

The execution of the instances with each solver, was performed on a cluster managed with *slurm*.[18] Each cluster node has a 3.80 GHz Intel Xeon E3-1270 v6 processor with 64 GB of RAM and Ubuntu 18.04 distribution. The instances were executed once per solver and with a time limit of two hours. A timeout condition was defined when the solver does not find a new solution in a span of 10 min. On the other hand, the training and testing with neural networks were run on a 2.9 GHz Intel Core i5-10400 processor with 16 GB of RAM and a Nvidia GeForce GTX 1650 SUPER GPU.

5. Anytime automatic algorithm selection for TSP

This work's approach is summarized in Fig. 5. First, the datasets to be used are selected. In this case, they were divided into training instances (generated) and test instances (public data). The usage of euclidean instances allows us to represent the instances as a matrix

[12] https://github.com/jakobbossek/tspgen.
[13] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/.
[14] http://www.or.uni-bonn.de/~hougardy/HardTSPInstances.html.
[15] http://www.math.uwaterloo.ca/tsp/index.html.
[16] http://www.math.uwaterloo.ca/tsp/vlsi/index.html.
[17] https://isaiash.github.io/anytime_tsp/.
[18] https://slurm.schedmd.com/documentation.html.

Table 2
Summary of the datasets used.

	Dataset	N instances	Mean N cities	Min N cities	Max N cities	Std dev. cities	25% cities	50% cities	75% cities
Test	TSPLIB	88	3041	14	85,900	10,185	134	428	1468
	NATIONAL	27	11,088	29	71,009	14,558	1800	8079	12,412
	VLSI	102	25,654	131	744,710	90,865	1926	3660	19,373
	TNM	141	4432	52	100,000	16,041	157	262	367
Train	RUE	3150	1452	500	2000	451	1000	1500	2000
	NETGEN	600	1250	500	2000	559	875	1250	1625
	NETGENM	600	1250	500	2000	559	875	1250	1625
	TSPGEN	1981	10,976	11	32,304	7373	4969	10,004	15,783

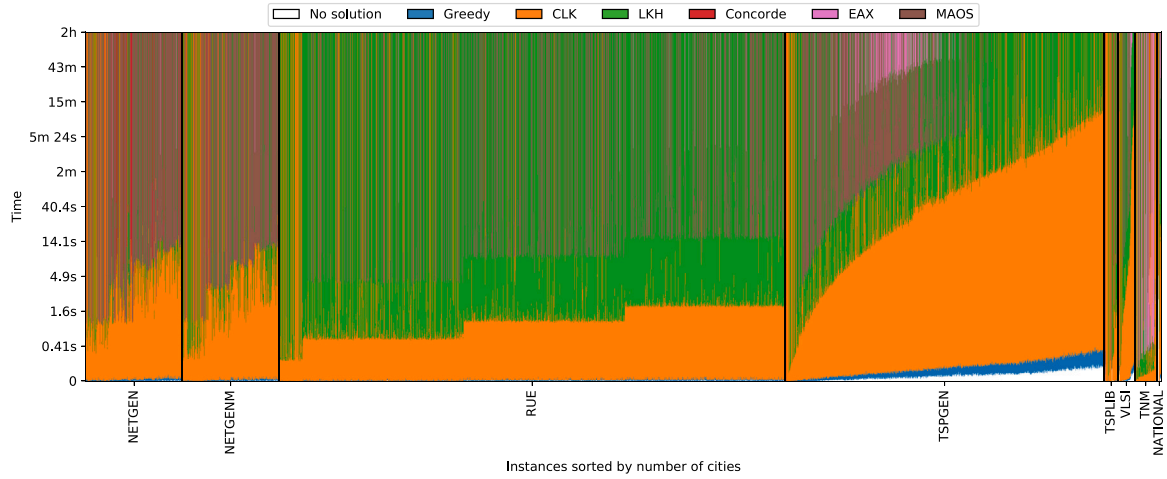

Fig. 4. Behavior of the best solver over time, grouped by dataset. The color indicates the solver that found the best solution first.

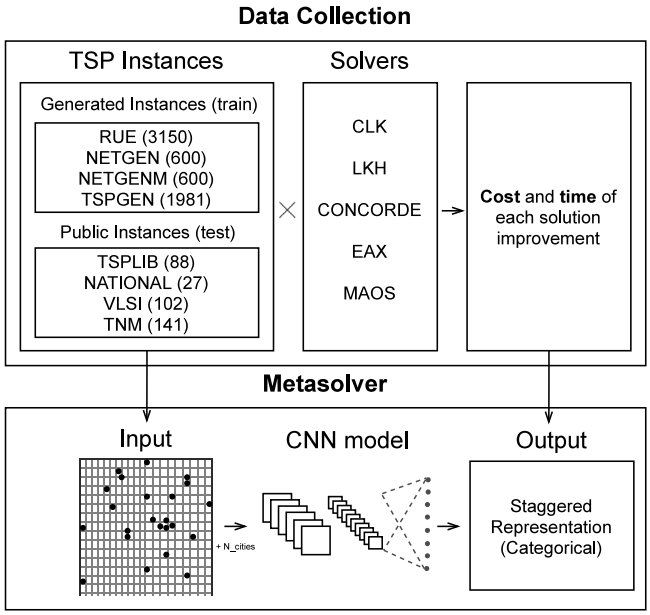

Fig. 5. General scheme.

(Section 5.1), this representation will be used as input for our convolutional neural network (Section 5.3). Each instance is solved by every one of the 5 selected state-of-the-art solvers (CLK, LKH, CONCORDE, EAX, MAOS) and every new objective value and time at which it was obtained during run time are stored. As we explain below, the direct usage of the objective value is not appropriate to train the neural network, so a different representation has been defined, which we call Staggered Representation (Section 5.2).

5.1. Input: instance representation

Looking for meaningful and low computational cost features is not a trivial task. In many cases the selected characteristics are not decisive in the algorithm selection process, misusing computational capacity. Furthermore, more important characteristics could be overlooked. This is mainly due to the arbitrary selection of features. A way to avoid this problem is to use neural networks, which have parameters that are automatically adjusted to consider relevant features.

The representation should both be fast and keep as much information of the instance as possible. For this we proposed a representation of the instances in which cities are spatially represented in a square matrix. In this fashion, the limits of the matrix match the coordinate limits and the cells of the matrix store the amount of cities within its corresponding area. Fig. 6 shows an example of how this matrix representation is used in an euclidean instance, considering the limits to assign to each matrix position. Then, the matrix of size $N\_grid \cdot N\_grid$ is constructed in which each cell stores the number of cities found in that area.

This representation of the instance follows similar principles as the one proposed in Zhu et al. (2021) which shows how the advance on state-of-the-art Deep Learning applied to image processing can be used to treat tabular data. Intuitively, in the special case of *Euclidean* TSP, a 2-D image, as the one we propose here, may be informative enough so to capture the relationship between clusters of closely related cities and, hence, characterize the type of instance. We point out that this representation takes the advantage of euclidean distances that preserve the triangular inequality, since the relative distances between clusters of cities are preserved. As shown in Dantzig, Fulkerson, and Johnson (1954), grouping close cities in one cell does not lose crucial information since, most probably, the optimal tour will not link the merged cities with cities outside the cell, besides the linking city (i.e. the city that connects two cells in the optimal path).

In order to normalize the matrix to values between 0 and 1, the maximum value for the cells must be known. Given that this value can

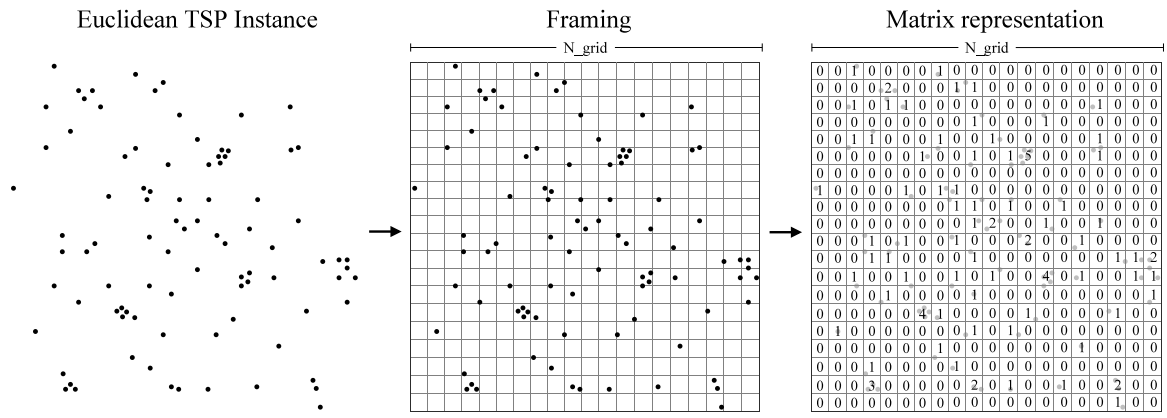

Fig. 6. Example of an instances characterization.

be very large in some instances, a limit *max_cities* is defined, which indicates the maximum number of cities that will be considered for a cell. If a cell has more than *max_cities* cities, its value will be restricted to *max_cities*. Having a value too large for this parameter could make cells with only one city to be negligible, while having a low value like $max\_cities = 1$ will lose cell city density information. The adjustment of N_grid and *max_cities* parameters will be later discussed.

To augment the training set, three rotations of each matrix representation were added, increasing from 6331 to 25,324 training instances. Given that the solution for rotated instances is identical, the expected output for these instances is copied from the original one.

5.2. Output: Instance's solution representation

The model output corresponds to a ranking representation of the cost (or objective value) achieved by each solver over discretized time steps. The number of time steps was set to 500, from zero to 7200 s (two hours). Time steps are distributed following an exponential curve to focus on low execution times. We do it in this fashion due to the concentration of changes in the ranking of the solvers at lower run times, after that, new solutions are scarcely found.

At each time step, we obtain the cost achieved by each of the five solvers, however, there are some problems in representing the neural network output with these values directly:

- Depending on the distances between cities, different instances will have different orders of magnitude in their costs.
- At the early instants of the execution, cost is infinite since no solution has been found, then, the values found by the solvers would become negligible.
- Later in the execution, the differences of costs between solutions are minimal, therefore they would be negligible in comparison to early deviations, even though they are important when we approach to the optimum value.
- If two or more solvers share the cost at some instant, we will have no information as to which one was obtained earlier.

A representation that eludes all of the aforementioned issues is what we call the Staggered Representation. In this representation, all the solvers start with a value of 0 and all their subsequent values will be integers. To build the time series with positions $P_s(i,t) \in \mathbb{Z}$, one must evaluate the cost $O_s(i,t) \in \mathbb{R}$ of each solver $s \in \{1,\dots,5\}$ and every instance $i \in I$ on time step t in relation to the previous time step $t-1$, where $t \in \{1,\dots,500\}$. If from $t-1$ to t a solver a surpasses the suboptimality degree of another solver b, then its new position is defined as $P_a(i,t) = P_b(i,t-1)+1$, and if another solver c already has that position, then the position of solver c is also updated to $P_c(i,t) = P_c(i,t)+1$. One exception is the case in which a is adjacent to b, i.e., $P_a(i,t) = P_b(i,t) \pm 1$, then the positions are swapped. To acknowledge faster solution-finder solvers, we define a tie-breaker: if $O_a(i,t) = O_b(i,t)$, but a found the solution first, then $P_a(i,t) > P_b(i,t)$. Notice that with these rules, solvers can only share the starting position at value 0.

An example of the Staggered Representation is shown in Fig. 7 where the ranking timeline of instance `pcb3038` can be seen (Same instance as Fig. 1 for comparative purposes). With this representation, each individual observation of an instant delivers the ranking of solvers (where the highest value has a better ranking) and at the same time holds the information of the solution improvements and solvers passing each other along time. The highest Position value used in this work is 12.

In Fig. 7 we can observe how the first solution is found by the greedy algorithm occupying the first place of a ranking composed of 1 solver with valid solutions. By second 0.004, CLK is able to beat the greedy solution and occupies the first place out of 2 solvers that were able to find at least one valid solution. Finally, from approximately second 110 onwards, CLK and Concorde alternate the first and second place of the ranking composed of 3 solvers with valid solutions.

5.3. Model

Using the matrix characterization allows the usage of Convolutional Neural Networks (CNNs). With this characterization, the information about the number of cities is not given directly, even more, when setting a low value for *max_cities* the size cannot be inferred due to the loss of density information. For this reason, this value is part of the input of the network. Note that when the characterization matrix is calculated, the number N_cities of cities can be obtained without additional calculations. This value will be fed through a second channel with the value in every cell of the matrix, so the input size is $2(N\_grid \cdot N\_grid)$.

As it was shown in Section 5.2, the representation of the solutions is a list for each solver with integer values that represent its ranking and increase when there are more overtakes. The size of this output is 2500 (five solvers in 500 time steps). This data is delivered to the convolutional network as a list of 2500 integer values. With these data the network can be trained within a regressive fashion and with a linear output. Nonetheless, the representation of the output values as real numbers does not match with the expected answer, i.e., to know the best solver. To avoid this issue, a one-hot encoding (Harris & Harris, 2010) is applied over the output list. The greatest value found in our datasets is 12, so the encoding results in an output of size (2500; 12). This way, by using a sigmoid activation function in the output, we obtain a classification model.

Fig. 8 shows the architecture of the used model in detail. The amount of parameters to train for the model is 9,866,864. For training, the Adam optimizer (Kingma & Ba, 2014) was used with a learning rate of 0.0001 and a binary cross-entropy loss function (Cox, 1958).

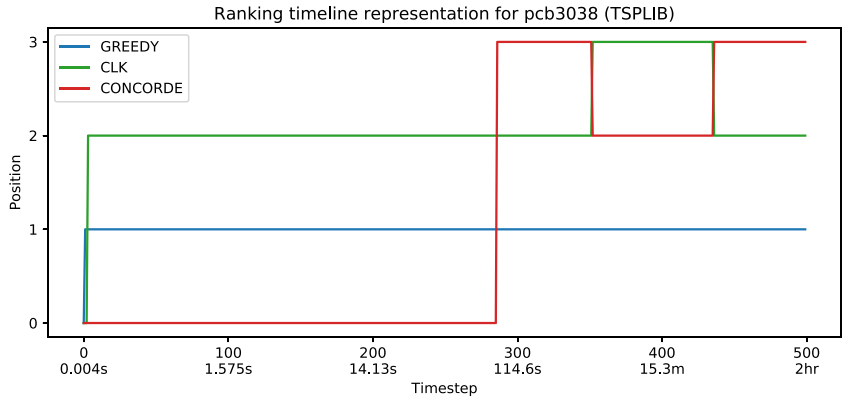

Fig. 7. Example of a Staggered Representation. The Greedy solver is included in the plot for explanatory purposes only and corresponds to the first solution reported by CLK..

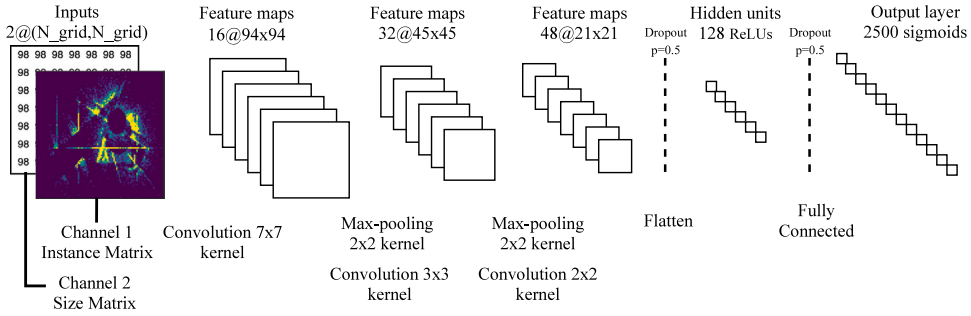

Fig. 8. Convolutional architecture.

To select the best parameters, several models were trained using different configurations for $N\_grid \in \{100, 200, 300\}$ and $max\_cities \in \{1, 5, 10, 20\}$. The best *categorical accuracy* was obtained with $N\_grid = 100$ and $max\_cities = 20$. Using these parameters, the model was retrained with a lower learning rate of $1 \cdot 10^{-5}$, which got a *categorical accuracy* of 66.8% at the end of training.

6. Results

With the CNN model set, the predictions were made over the test instances and, for each, the prediction time was measured, considering the reading of the instance, the creation of the matrix and the prediction of the model. Subsequent results are shown with two evaluation approaches: i) without considering the prediction time (w/o p.t.); and ii) adding it to the total execution time (w/ p.t.). Fig. 9 shows how much time on average it takes to predict and how much each sub process contributes to it. On average, it takes to read the instance, create the matrix and predict the solver 0.0026, 0.0073 and 0.0096 s, respectively.

The real *accuracy* of the solvers, including the metasolver, was calculated as the percentage of time steps at which the solver had the best objective value. Being $cost_s(t)$ the objective value obtained by solver s at the time step t, where $t \in \{1, 2, \ldots, 500\}$, and $best\_cost(t)$ the best objective value obtained for all solvers up until instant t, then the *accuracy* of solver s over an instance is defined as:

$$accuracy(s) = \frac{\sum_{t=1}^{500} equal(cost_s(t),\ best\_cost(t))}{500}$$

where

$$equal(a, b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases}$$

Recall that 500 is the amount of time steps considered. With this metric we can calculate the average over each test dataset. These results are shown in Table 3.

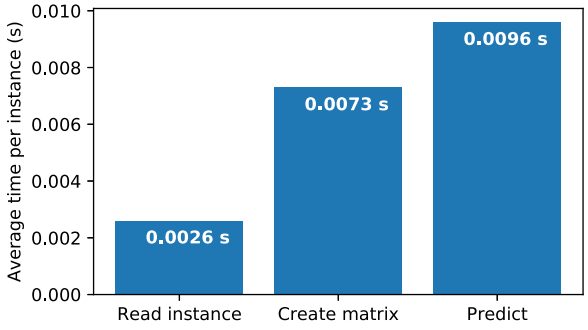

Fig. 9. Average time by sub process of the prediction time. All of the instances were used for this measurement. The total prediction time on average is 0.01954 s.

Table 3
Average accuracy of each solver on each test dataset..

Accuracy (%)	TSPLIB	VLSI	NATIONAL	TNM	ALL
Metasolver (without prediction time)	90.0	91.3	89.5	90.8	90.7
Metasolver (with prediction time)	**82.7**	**73.2**	**66.9**	**85.1**	**79.8**
CLK	63.1	57.5	61.8	77.2	67.0
LKH	62.1	55.0	44.0	42.4	51.0
CONCORDE	50.2	15.8	18.5	19.1	25.8
EAX	39.2	32.2	22.2	54.1	41.8
MAOS	46.2	42.2	01.4	67.6	50.1

Fig. 10 shows the temporal behavior of the *accuracy* of each of the solvers over the public data sets TSPLIB, VLSI, NATIONAL and TNM. The plots on the left show time series where the x-axis depicts the time steps and the y-axis shows the number of times that a solver has had the best objective value, while the plot on the right shows the area under this curve.

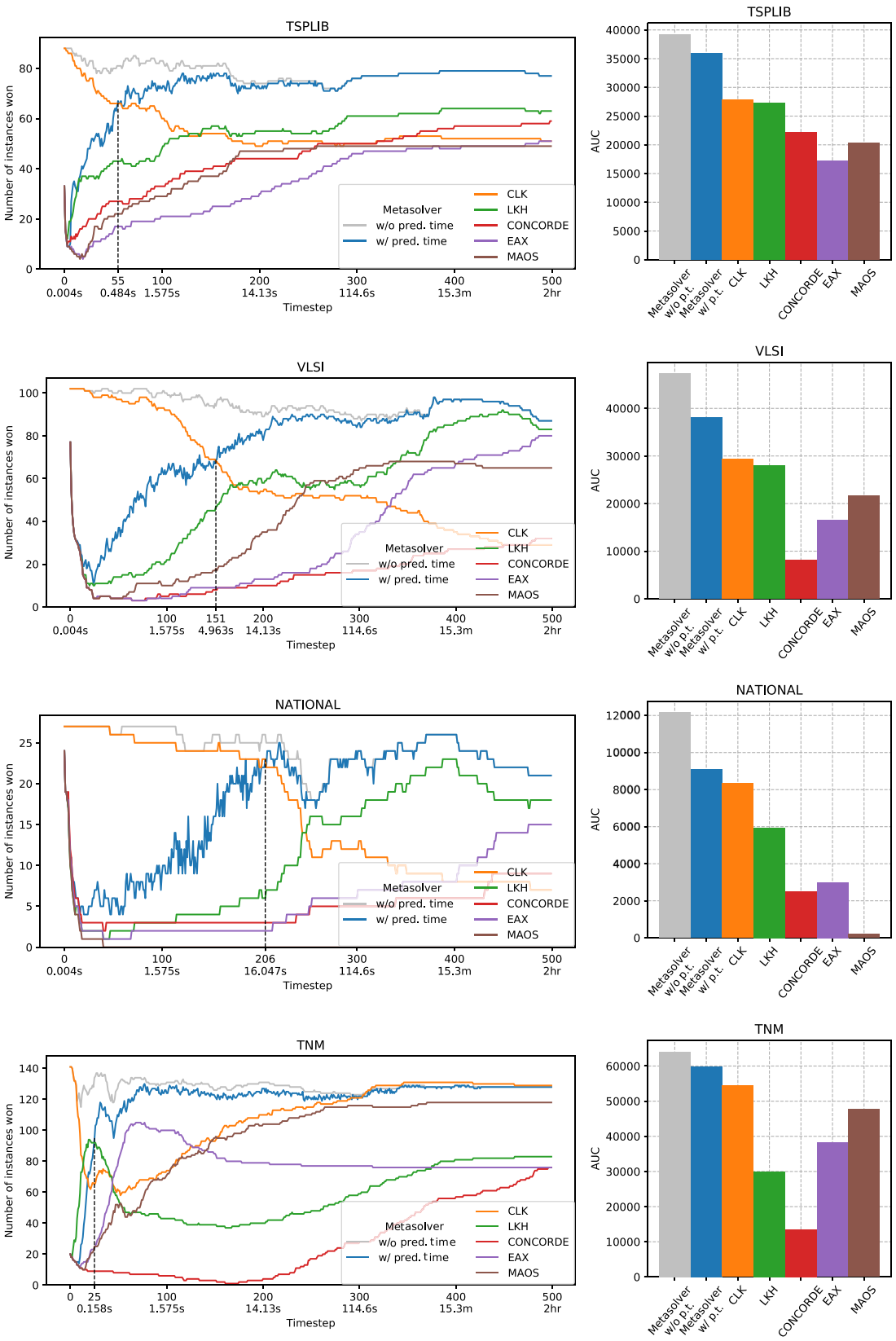

Fig. 10. Results for each of the public datasets.

The area under the curve plots show that the metasolver can be superior in all public data sets according to this metric. However, it must be highlighted that there is a threshold (dotted line) from which the metasolver can obtain on average better solutions than any other solver. In every data set, CLK manages to get solutions very quickly and is superior to the metasolver early in the execution. In TSPLIB this threshold is at 0.578 s, in VLSI at 5.18 s, in NATIONAL at 16.73 s, and in TNM, LKH is surpassed at 0.158 s. In TNM, CLK surpasses

the metasolver by a short margin after the 2 min. This might happen because TNM has instances that are very similar to each other, so if a solver is predominant for an instance (such as CLK) it will predominate in the entire data set.

These results (Fig. 10 and Table 3) support our working hypotheses. Mainly, the high accuracy obtained shows that our representation of the instances preserve enough information so to characterize them well. As a consequence, this representation is able to take advantage of current state-of-the-art deep learning techniques, which showed to be of use also in this domain. This representation also allowed us to characterize the instances with a simple linear algorithm, which permitted us to achieve one of the goals we had, which is to spend very little time in the solver recommendation process. Another important part of our approach is the design of the output of the network, using a staggered representation instead of the traditional estimation of the values. This design also provided appropriate ranking information, useful in predicting the best solver for a given time and instance.

7. Conclusions and future work

This work presents a new model for automatic algorithm selection which surpasses the performance of state-of-the-art solvers in all the considered public data sets, even when accounting for the prediction time. This model is capable of predicting the best of five state-of-the-art solvers at 500 different time steps that goes from 0 to 7200 s with an accuracy of 79.8%.

Previous related works (Kerschke et al., 2018; Pihera & Musliu, 2014) faced the issue of selecting an appropriate characterization for the instances. As discussed, selecting few features can lead to a poor representation of the instances, while selecting too many may greatly increase the computation time and therefore the overall prediction time. For example, computing the feature set TSPmeta[19] (Mersmann et al., 2013) (68 features) or full UBC (Hutter, Xu, Hoos, & Leyton-Brown, 2014) (50 features) can take between 9 to 12 s for each instance. In this work, we avoid feature computation and achieve an average prediction time of 0.02 s for each instance.

Although the solvers and instances used in Kerschke et al. (2018) are related to the ones used in this work, the total amount of instances was augmented from 1845 to 6689 and the exact solver Concorde was also used, without restricting ourselves to approximated algorithms. Furthermore, the execution time limit was extended from 3600 to 7200 s, giving a wider time perspective.

The anytime approach not only offers the possibility of answering time restricted predictions, but is also more instructive for the classifying model, allowing for the attainment of a higher accuracy and better generalization for unknown behaviors.

Furthermore, a framework for automatic algorithm selection over the TSP using neural networks was developed, which consists of a new matrix characterization of the instances, avoiding the costly traditional feature computation and enabling the usage of convolutional neural networks as the classification model. Finally, we designed a representation of an instance solution in which, on the one hand keeps the information of the ranking along the time, and on the other hand, can be used for the output of a classifying neural network.

Although the prediction times achieved in this work are quite low, it is possible to further optimize the prediction when the question to answer is "Which solver is best for instance i given t time?". In that case, we only need to know what happens at instant t and not at the other 499 time steps. Thus, irrelevant nodes in the output layer could be shut off when predicting, reducing the amount of parameters to compute.

A novelty of this work is the anytime approach, however, the practical application of the generated model only makes sense if it is used on the same architecture (CPU, GPU, memory, etc.) as the system in which the experiments were carried out, otherwise the prediction times will not match. We propose to explore the idea of mapping the times to other systems, including complex parallel architectures as the ones presented in Abdullah, Abuelrub, and Mahafzah (2011), Baddar and Mahafzah (2014) and Mahafzah, Alshraideh, Abu-Kabeer, Ahmad, and Hamad (2012).

Just as constraints on the runtime can be considered, the same idea can be extended to other computational resources. For instance, if a solver requires large quantities of RAM (like a dynamic programming algorithm) but the system does not have enough resources, it would be appropriate to consider this when selecting the best solver.

[19] https://github.com/berndbischl/tspmeta.

CRediT authorship contribution statement

Isaías I. Huerta: Investigation, Data curation, Methodology, Software, writing – original draft, Writing – review & editing. **Daniel A. Neira:** Methodology, Writing – original draft, Writing – review & editing. **Daniel A. Ortega:** Methodology, Writing – original draft, Writing – review & editing. **Vicente Varas:** Methodology, Writing – original draft, Writing – review & editing. **Julio Godoy:** Investigation, Methodology, Writing – original draft, Writing – review & editing. **Roberto Asín-Achá:** Conceptualization, Methodology, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abdullah, M., Abuelrub, E., & Mahafzah, B. (2011). The chained-cubic tree interconnection network. *International Arab Journal of Information Technology*, *8*, 334–343.

Agatz, N., Bouman, P., & Schmidt, M. (2018). Optimization approaches for the traveling salesman problem with drone. *Transportation Science*, *52*, 965–981.

Al-Adwan, A., Mahafzah, B. A., & Sharieh, A. (2018). Solving traveling salesman problem using parallel repetitive nearest neighbor algorithm on otis-hypercube and otis-mesh optoelectronic architectures. *The Journal of Supercomputing*, *74*, 1–36.

Al-Adwan, A., Sharieh, A., & Mahafzah, B. A. (2019). Parallel heuristic local search algorithm on otis hyper hexa-cell and otis mesh of trees optoelectronic architectures. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, *49*, 661–688.

Andresol, R., Gendreau, M., & Potvin, J.-Y. (1999). A hopfield-tank neural network model for the generalized traveling salesman problem. In *Meta-heuristics* (pp. 393–402). Springer.

Applegate, D., Bixby, R., Chvátal, V., & Cook, W. (1995). *Finding cuts in the TSP (A preliminary report), Vol. 95*. Citeseer.

Applegate, D., Bixby, R., Chvatal, V., & Cook, W. (2006a). Concorde tsp solver.

Applegate, D. L., Bixby, R. E., Chvatal, V., & Cook, W. J. (2006b). *The traveling salesman problem: A computational study*. Princeton University Press.

Applegate, D., Cook, W., & Rohe, A. (2003). Chained lin-kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, *15*, 82–92.

Arora, S. (1996). Polynomial time approximation schemes for euclidean tsp and other geometric problems. In *Proceedings of 37th conference on foundations of computer science* (pp. 2–11). IEEE.

Arora, S. (1997). Nearly linear time approximation schemes for euclidean tsp and other geometric problems. In *Proceedings 38th annual symposium on foundations of computer science* (pp. 554–563). IEEE.

Baddar, S. W. A.-H., & Mahafzah, B. A. (2014). Bitonic sort on a chained-cubic tree interconnection network. *Journal of Parallel and Distributed Computing*, *74*, 1744–1761.

Baraglia, R., Hidalgo, J. I., & Perego, R. (2001). A hybrid heuristic for the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, *5*, 613–622.

Bochkovskiy, A., Wang, C., & Liao, H. M. (2020). Yolov4: Optimal speed and accuracy of object detection. CoRR, abs/2004.10934. URL: https://arxiv.org/abs/2004.10934. arXiv:2004.10934.

Boryczka, U., & Szwarc, K. (2019). The harmony search algorithm with additional improvement of harmony memory for asymmetric traveling salesman problem. *Expert Systems with Applications*, *122*, 43–53.

Bossek, J., Kerschke, P., Neumann, A., Wagner, M., Neumann, F., & Trautmann, H. (2019). Evolving diverse tsp instances by means of novel and creative mutation operators. In *Proceedings of the 15th ACM/SIGEVO conference on foundations of genetic algorithms* (pp. 58–71).

Brocki, Ł., & Koržinek, D. (2007). Kohonen self-organizing map for the traveling salesperson problem. In *Recent advances in mechatronics* (pp. 116–119). Springer.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., et al. (2020). Language models are few-shot learners. CoRR, abs/2005.14165. URL: https://arxiv.org/abs/2005.14165. arXiv:2005.14165.

Chen, J.-S., Zhang, X.-Y., & Chen, J.-J. (2007). An elastic net method for solving the traveling salesman problem. In *2007 international conference on wavelet analysis and pattern recognition, Vol. 2* (pp. 608–612). IEEE.

Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *20*, 215–232.

Dantzig, G., Fulkerson, R., & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, *2*, 393–410.

Davendra, D. (2010). *Traveling salesman problem: Theory and applications*. IntechOpen.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *26*, 29–41.

Eberhart, R. C., Shi, Y., & Kennedy, J. (2001). *Swarm intelligence*. Elsevier.

Ezugwu, A. E.-S., & Adewumi, A. O. (2017). Discrete symbiotic organisms search algorithm for travelling salesman problem. *Expert Systems with Applications*, *87*, 70–78.

Fujimoto, N., & Tsutsui, S. (2010). A highly-parallel tsp solver for a gpu computing platform. In *International conference on numerical methods and applications* (pp. 264–271). Springer.

Fukunaga, A. S. (2000). Genetic algorithm portfolios. In *Proceedings of the 2000 congress on evolutionary computation. CEC00 (Cat. No. 00TH8512), Vol. 2* (pp. 1304–1311). IEEE.

Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets* (pp. 267–285). Springer.

Garey, M. R., Graham, R. L., & Johnson, D. S. 1976. Some np-complete geometric problems. In *Proceedings of the eighth annual ACM symposium on theory of computing* (pp. 10–22).

Gavidia-Calderon, C., & Castañon, C. B. (2020). Isula: A java framework for ant colony algorithms. *SoftwareX*, *11*, Article 100400.

Grötschel, M., & Holland, O. (1991). Solution of large-scale symmetric travelling salesman problems. *Mathematical Programming*, *51*, 141–202.

Guerri, A., & Milano, M. (2004). Learning techniques for automatic algorithm portfolio selection. In *ECAI, Vol. 16* (p. 475).

Harris, D., & Harris, S. (2010). *Digital design and computer architecture*. Morgan Kaufmann.

Hassoun, M. H., et al. (1995). *Fundamentals of artificial neural networks*. MIT Press.

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*, 106–130.

Helsgaun, K. (2018). *Using popmusic for candidate set generation in the lin-kernighan-helsgaun tsp solver, Vol. 7*. Roskilde Universitet.

Huerta, I. I., Neira, D. A., Ortega, D. A., Varas, V., Godoy, J., & Asín-Achá, R. (2020). Anytime automatic algorithm selection for knapsack. *Expert Systems with Applications*, Article 113613.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). Paramils: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*, 267–306.

Hutter, F., Xu, L., Hoos, H. H., & Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, *206*, 79–111.

Kadioglu, S., Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2011). Algorithm selection and scheduling. In *International conference on principles and practice of constraint programming* (pp. 454–469). Springer.

Kelley, H. J. (1960). Gradient theory of optimal flight paths. *Ars Journal*, *30*, 947–954.

Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., & Trautmann, H. (2018). Leveraging tsp solver complementarity through machine learning. *Evolutionary Computation*, *26*, 597–620.

Kerschke, P., & Trautmann, H. (2019). Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary Computation*, *27*, 99–127.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kongkaew, W., & Pichitlamken, J. (2014). A survey of approximate methods for the traveling salesman problem. *Kasetsart Engineering Journal*, *27*, 79–87.

Kotthoff, L. (2016). Algorithm selection for combinatorial search problems: A survey. In *Data mining and constraint programming* (pp. 149–190). Springer.

La Maire, B. F., & Mladenov, V. M. (2012). Comparison of neural networks for solving the travelling salesman problem. In *11th symposium on neural network applications in electrical engineering* (pp. 21–24). IEEE.

Laporte, G. (1992). The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, *59*, 231–247.

Lawler, E. L. (1985). *Wiley-interscience series in discrete mathematics, The traveling salesman problem: A guided tour of combinatorial optimization*.

Lin, S., & Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, *21*, 498–516.

Mahafzah, B. A. (2014). Performance evaluation of parallel multithreaded a* heuristic search algorithm. *Journal of Information Science*, *40*, 363–375.

Mahafzah, B. A., Alshraideh, M., Abu-Kabeer, T. M., Ahmad, E. F., & Hamad, N. A. (2012). The optical chained-cubic tree interconnection network: topological structure and properties. *Computers and Electrical Engineering*, *38*, 330–345.

Malitsky, Y., Sabharwal, A., Samulowitz, H., & Sellmann, M. (2013). Algorithm portfolios based on cost-sensitive hierarchical clustering. In *Twenty-third international joint conference on artificial intelligence*. Citeseer.

McKay, M. D., Beckman, R. J., & Conover, W. J. (2000). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *42*, 55–61.

Mersmann, O., Bischl, B., Trautmann, H., Wagner, M., Bossek, J., & Neumann, F. (2013). A novel feature-based approach to characterize algorithm performance for the traveling salesperson problem. *Annals of Mathematics and Artificial Intelligence*, *69*, 151–182.

Merz, P., & Freisleben, B. (2001). Memetic algorithms for the traveling salesman problem. *Complex Systems*, *13*, 297–346.

MirHassani, S. A., & Habibi, F. (2013). Solution approaches to the course timetabling problem. *Artificial Intelligence Review*, *39*, 133–149.

Mitchelly, J. S. (1997). Guillotine subdivisions approximate polygonal subdivisions: Part ii - a simple polynomial-time approximation scheme for geometric k-mst, tsp, and related problems. *SIAM Journal on Computing*.

Nagata, Y., & Kobayashi, S. (1997). Edge assembly crossover: A high-power genetic algorithm for the travelling salesman problem. In *Proceedings of the 7th international conference on genetic algorithms (ICGA)* (pp. 450–457).

Nagata, Y., & Kobayashi, S. (2013). A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing*, *25*, 346–363.

O'Neil, M. A., Tamir, D., & Burtscher, M. (2011). A parallel gpu version of the traveling salesman problem. In *Proceedings of the international conference on parallel and distributed processing techniques and applications (PDPTA)* (p. 1). The Steering Committee of The World Congress in Computer Science, Computer ....

Osaba, E., Yang, X.-S., & Del Ser, J. (2020). Traveling salesman problem: a perspective review of recent research and new results with bio-inspired metaheuristics. In *Nature-inspired computation and swarm intelligence* (pp. 135–164).

Padberg, M., & Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, *33*, 60–100.

Papadimitriou, C. H. (1997). The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, *4*, 237–244.

Perron, L., & Furnon, V. (2019). Or-tools. URL: https://developers.google.com/optimization/.

Pihera, J., & Musliu, N. (2014). Application of machine learning to algorithm selection for tsp. In *2014 IEEE 26th international conference on tools with artificial intelligence* (pp. 47–54). IEEE.

Potvin, J.-Y. (1993). State-of-the-art survey—the traveling salesman problem: A neural network perspective. *ORSA Journal on Computing*, *5*, 328–348.

Rego, C., Gamboa, D., Glover, F., & Osterman, C. (2011). Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research*, *211*, 427–441.

Reinelt, G. (1994). *The traveling salesman: Computational solutions for TSP applications*. Springer-Verlag.

Rice, J. R., et al. (1976). The algorithm selection problem. *Advances in Computers*, *15*(5).

Rocki, K., & Suda, R. (2013). High performance gpu accelerated local optimization in tsp. In *2013 IEEE international symposium on parallel & distributed processing, workshops and Phd forum* (pp. 1788–1796). IEEE.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Saji, Y., & Barkatou, M. (2021). A discrete bat algorithm based on lévy flights for euclidean traveling salesman problem. *Expert Systems with Applications*, *172*, Article 114639.

Sampson, J. R. (1976). *Adaptation in natural and artificial systems*. (john h. holland).

Scholz, J. (2019). Genetic algorithms and the traveling salesman problem a historical review. arXiv preprint arXiv:1901.05737.

Shi, X. H., Liang, Y. C., Lee, H. P., Lu, C., & Wang, Q. (2007). Particle swarm optimization-based algorithms for tsp and generalized tsp. *Information Processing Letters*, *103*, 169–176.

Stützle, T., & Hoos, H. H. (2000). Max–min ant system. *Future Generation Computer Systems*, *16*, 889–914.

Tsai, H.-K., Yang, J.-M., Tsai, Y.-F., & Kao, C.-Y. (2004). An evolutionary algorithm for large traveling salesman problems. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *34*, 1718–1729.

Vallati, M., Chrpa, L., & Kitchin, D. (2014). Asap: an automatic algorithm selection approach for planning. *International Journal on Artificial Intelligence Tools*, *23*, Article 1460032.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*, 67–82.

Xie, X.-F., & Liu, J. (2008). Multiagent optimization system for solving the traveling salesman problem (tsp). *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, *39*, 489–502.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2008). Satzilla: portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, *32*, 565–606.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)* (pp. 16–30).

Yang, Y., & Whinston, A. (2020). A survey on reinforcement learning for combinatorial optimization. arXiv preprint arXiv:2008.12248.

Zhang, Y., Qin, J., Park, D. S., Han, W., Chiu, C.-C., Pang, R., et al. (2020). Pushing the limits of semi-supervised learning for automatic speech recognition. arXiv: 2010.10504.

Zhu, Y., Brettin, T., Xia, F., Partin, A., Shukla, M., Yoo, H., et al. (2021). Converting tabular data into images for deep learning with convolutional neural networks. *Scientific Reports*, *11*, 1–11.