2023 IEEE Symposium Series on Computational Intelligence (SSCI)
Mexico City, Mexico. December 5-8, 2023

Using Reinforcement Learning for Per-Instance Algorithm Configuration on the TSP

1st Moritz Vinzent Seiler
Data Science: Statistics and Optimization
University of Münster
Münster, Germany
moritz.seiler@uni-muenster.de

2nd Jeroen Rook
Data Management and Biometrics
University of Twente
Enschede, Netherlands
j.g.rook@utwente.nl

3rd Jonathan Heins
Big Data Analytics in Transportation
TU Dresden
Dresden, Germany
jonathan.heins@tu-dresden.de

4th Oliver Ludger Preuß
Data Science: Statistics and Optimization
University of Münster
Münster, Germany
oliver.preuss@uni-muenster.de

5th Jakob Bossek
Chair for AI Methodology
RWTH Aachen University
Aachen, Germany
bossek@aim.rwth-aachen.de

6th Heike Trautmann
Data Science: Statistics and Optimization
Universities of Münster/Twente
Münster/Enschede, Germany/Netherlands
trautmann@wi.uni-muenster.de

***Abstract*—Automated Algorithm Configuration (AAC) usually takes a global perspective: it identifies a parameter configuration for an (optimization) algorithm that maximizes a performance metric over a set of instances. However, the optimal choice of parameters strongly depends on the instance at hand and should thus be calculated on a per-instance basis. We explore the potential of *Per-Instance Algorithm Configuration* (PIAC) by using *Reinforcement Learning* (RL). To this end, we propose a novel PIAC approach that is based on deep neural networks. We apply it to predict configurations for the *Lin–Kernighan heuristic* (LKH) for the *Traveling Salesperson Problem* (TSP) individually for every single instance.**

To train our PIAC approach, we create a large set of 100 000 TSP instances with 2 000 nodes each — currently the largest benchmark set to the best of our knowledge. We compare our approach to the state-of-the-art AAC method *Sequential Model-based Algorithm Configuration* (SMAC). The results show that our PIAC approach outperforms this baseline on both the newly created instance set and established instance sets.

***Index Terms*—Per-Instance Algorithm Configuration, Traveling Salesperson Problem, Deep Reinforcement Learning**

I. Introduction and Related Work

A Euclidean *Traveling Salesperson Problem* (TSP) instance consists of a set of nodes/cities and pairwise distances between these nodes. The optimization problem then is to find the shortest route through all cities such that every city is visited exactly once. The TSP is one of the classical $\mathcal{NP}$-hard combinatorial optimization problems and is highly relevant in transportation logistics, circuit board design, and many other disciplines. A plethora of algorithmic approaches have been developed for the TSP. One can differentiate between exact solvers that guarantee to find a globally optimal solution, e.g. Concorde [1], Branch & Bound approaches [2], or naive exhaustive enumeration, and inexact solvers, e.g. *Edge-Assembly-Crossover* (EAX) [3] and the *Lin–Kernighan Heuristic* (LKH) [4].

LKH defined the state-of-the-art in inexact TSP solving for a long period of time. It employs a variable-depth approach to generate intricate local search moves by constructing a sequence of edge exchanges based on heuristics. In 2015, Dubois et al. [5] introduced an enhancement incorporating restarts. More recently, EAX as an evolutionary algorithm utilizing improved versions of both local and global variants of the edge assembly crossover operator turned out to be highly competitive, specifically in its restart variant.

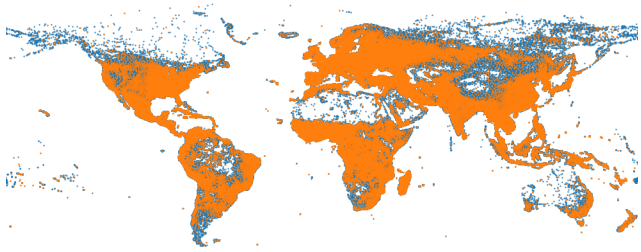


Fig. 1. Visualization of the World TSP Instance with 1 904 711 cities. The best-known tour so far has length 7 515 755 956 and was found by LKH (see https://www.math.uwaterloo.ca/tsp/world/ for more details). The 100 000 centroids for generating the instances are highlighted in orange while all other cities are shown in blue. The generated instances may overlap but differ in the local features. Please see Section II for more details.

The complementarity of both solvers and their restart variants on well-known and commonly used benchmark data sets such as *Random Uniform Euclidean* (RUE), clustered instances (Netgen) [6], Morphed [6], TSPLib, National, and VLSI, was shown in [7] and exploited for building a high-performing per-instance *Automated Algorithm Selection* (AAS, [8]) model to automatically decide on the best-suited solver for a given TSP instance. This trained selector significantly outperformed the *Single Best Solver* EAX(+restart). For AAS model training numerical, resp. tabular, features characterizing instance properties are commonly used stemming from the feature sets TSPMeta [9], UBC [10] and Pihera [11], e.g. based on statistics of the

978-0-7381-4408-5/23/$31.00 ©2023 IEEE

distance matrix or properties of minimum spanning trees or k-nearest neighbor graphs.

While, commonly, AAS on TSP focuses on minimizing computational runtime until the instance is solved to optimality, Bossek et al. [12] also considered anytime behavior of inexact TSP solvers based on the concept of empirical runtime distributions. As the main outcome, the performance ranking of solvers is heavily influenced by the specific level of approximation quality that is prioritized. The best-suited inexact TSP solver is highly dependent on the desired approximation level with LKH and its variants being highly competitive at the early and medium stages of the optimization phase, giving rise to a huge potential for solver hybridization and, in particular, *algorithm configuration* to improve convergence to optimality.

Since TSP solvers, like LKH and EAX, have performance-affecting parameters, multiple (potentially infinite) distinct variants of each solver exist, with each variant considered as a separate solver in the domain of AAS. Yet, in the context of *Automated Algorithm Configuration* (AAC), these variants are referred to as configurations. Therefore and orthogonal to AAS, AAC aims to find the one *static* configuration that yields the overall best average performance over all instances it is expected to solve, efficiently. Styles and Hoos [13] showed that applying AAC to LKH considerably improved performance compared to LKH with the default configuration on TSPLib as well as the potential for scaling performance. Their analysis did not focus on the actual found configurations, which prevents us from including them in our analysis. More recently, Rasku et al. [14] conducted an extensive empirical study on the effectiveness of different AAC methods to configure solvers for the *Vehicle Routing Problem*, a generalization of TSP. They showed that AAC in the VRP domain yields significant improvements and they recommended using *Sequential Model-Based Algorithm Configuration* (SMAC) [15] in this domain.

Based on the successes of AAS and AAC, we bridge the two domains by proposing two novel *Per-Instance Algorithm Configuration* (PIAC) [16] approaches for LKH where both approaches learn a model that determines the best configuration for every instance individually out of the infinitely large pool of variants. The latter property prevents us from using supervised learning. Instead, we utilize *Reinforcement Learning* to accomplish this task. Other existing PIACs are, to our knowledge, limited. Popular approaches [17], [18] build a portfolio of complementary configurations which is then deployed as an AAS framework. These portfolios are usually limited to only a few configurations of the complete configuration space and are therefore not considered as complete PIAC approaches.

Both our PIAC approaches utilize deep learning, where one is a *Multi-Layer Perceptron* (MLP), which relies on numerical instance features, and the other approach is a *Transformer* model that directly takes the Euclidean city coordinates as input [19]. A detailed description of both approaches and how they are fitted is given in Section III.

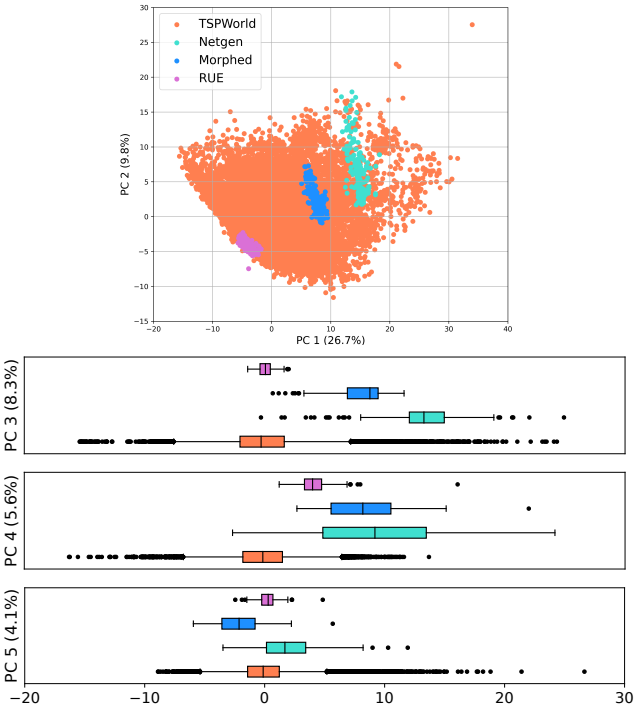


Fig. 2. Distribution of the different instance classes in the feature space represented by the first five PCs with the percentage of explained variance given in brackets. The first two PCs cover a combined variance of 36.5% while the first seven cover a large fraction of 54.5% of the variance in total.

Since training deep learning models requires large training sets and most TSP benchmark sets are of limited size, we also introduce a new diverse benchmark set in Section II, which is based on subsets of the *World TSP* instance. In Section IV we conduct an empirical study on the effectiveness of our PIAC approaches and compare their performance against static configurations. Finally, in Section VI, we discuss the results and set directions for future work.

II. Instance Dataset

One important premise for deep learning but especially when used in combination with reinforcement learning is having a large set of training- and validation data. Recent papers on AAS or AAC for the TSP domain usually considered rather small instance sets — between several hundred to a few thousand cities [7], [20], [21]. We found this to be far too small for our research task and, therefore, opted for a sophisticated approach to create a large and representative set of TSP instances. Several works have proposed approaches to generate diverse instances such as RUE, Netgen [6], Morphed [6], and Evolved [22]. Evolved, for example, uses an evolutionary algorithm that maximizes the performance ratio of two competitive solvers while yielding feature-diverse instances. Yet, we found that these generators are either quite costly (e.g. Evolved) or not really diverse (e.g. RUE or Netgen). Thus, we aimed for a different approach by sampling from a large, complex TSP instance as a donator.

To sub-sample a large instance into many smaller ones, we first defined a set of m starting points and then sampled for every starting point its $k-1$ nearest cities. This would result in m instances with k cities, each. However, there is a chance (even if the donator instance is sufficiently large) that the instances would overlap at their boundaries.To avoid these overlapping instances being too similar, one can increase the number of nearest cities (e.g. $2 \times k - 1$) and then randomly sample k cities. This way, instances would still overlap but they would differ in the local structure. We aimed for $m = 100\,000$ instances in total, giving us a large set of instances for training ($75\,000$) and validation ($25\,000$). In addition, we prefer larger instances of $2\,000$ cities to increase difficulty. Example instances can be found in Figure 2.[1]

We took one of the largest available TSP instances as the donator instance: the World TSP. It consists of $1\,904\,711$ cities around the world, and the current best found optimal tour has length $7\,515\,755\,956$ using LKH[2]. The coordinates of the cities are given in degrees of longitude and latitude. We used $100\,000$-means clustering algorithm in combination with the *haversine distance* [23], the angular distance of two points on a sphere, to evenly spread the centroids around the TSP World instance. Afterward, the centroids were used as starting points for the subsampling procedure. To avoid overlapping, we considered the $4\,999$ closest points of every starting point and uniformly at random sampled $k = 2\,000$ cities from these $5\,000$ (the starting point plus its $4\,999$ nearest neighbors), randomly. These $2\,000$ cities were then merged into one new TSP instance. Further, we mapped the cities into a Cartesian space with a size of $1\,000\,000 \times 1\,000\,000$.

Next, we executed Concorde on every instance to find the optimal tour with a cutoff time of one day. For $99\,998$ instances, Concorde returned the optimal tour within one day. However for two instances, `subset_world_29974.tsp` and `subset_world_60919.tsp`, Concorde was unable to find an optimal tour even within an increased cutoff time of seven days. Hence, we approximated the optimal tour by executing EAX with default settings 100 times on both instances and considering the respective best-found tour as optimal.

We analyzed the generated instances from a qualitative and quantitative standpoint. Visualizing the instances yields many complex structures: e.g. uniformly distributed, grid-like distributed, and clusters of cities (see Figure 3). To quantify the diversity captured by our instances, we compare their distribution in feature space with instances used in previous studies, namely RUE, Netgen, Morphed, and Evolved instances. To this end, using the R package `salesperson` [24] we compute all TSPMeta [9] and Pihera [11] features as well as most UBC [10] features (excluding the cost-intensive local search probing features and branch & bound features as these are not implemented in `salesperson`), for all of our instances and all other instances with a node size of $2\,000$. These features are numeric values with supposedly describe the characteristics of a TSP instance and are determined for each TSP instance individually. Note that there are approaches to normalize TSP instance features [21], [25] with regard to the node and bounding box size which would allow a comparison with different instances. However, many features are still only available in their unnormalized versions until now. This results in potentially missing important instance information. Consequently, we use unnormalized versions of all features in this study. The instances of the same group were found to be very similar with regard to their normalized feature distribution disregarding the size in both normalization studies. Thus, comparing our instances in the unnormalized feature space with instances of the same size is representative.

Assessing the distribution of instances regarding individual dimensions of the complete feature space with the dashboard provided in [21], we found that our instances exhibit diversity, capturing the main properties of all other instance groups. To visually represent these findings, we performed a *Principal Component Analysis* (PCA) with all instances and depict the results in Figure 2. The distribution of our instances across the most important principal components (PC) covers those of all other instance groups. An exception to this are a few Netgen and Morphed instances in PC 4 which however accounts only for 5.4% of the total variance. This shows that our instances are diverse, representative and cover a larger area of the feature space than previous classes of instances.

III. Methodology

LKH exhibits a large set of parameters that influence its performance. In this setting, we define performance as the computational runtime, measured in CPU time, to find the optimal tour. If LKH fails to find the optimal tour within a given time limit (cutoff time), it will receive a penalty. We choose the *Penalized Quantile Runtime 10* (PQR10) score [26] to measure the performance of LKH. The PQR10 score is defined as the median runtime over multiple runs if more than 50% of the runs were successful. Otherwise, the PQR10 score is 10-times the cutoff time. We choose the PQR10 score over the *Penalized Average Runtime 10* (PAR10) score as we found it to be more robust against timeouts, which is supported by the results in [27]. Usually, ten solver runs are considered as a standard to compute PQR10 and also PAR10 scores. Yet, we found that during training it is often more advantageous to execute less than ten runs and spend the spared time on evaluating more configurations without increasing the total number of LKH runs. Note that for validation we always consider the PQR10 score with exactly ten runs and seeds $\in [1, 10]$ to yield comparable results.

In the following, we differentiate between *Full-set Algorithm Configuration* and *Per-instance Algorithm Configuration*. Thereafter, we will compare both approaches against

[1] Upon acceptance, we will publish the dataset for open use.

[2] See https://www.math.uwaterloo.ca/tsp/world/ for more information on the World TSP instance.

Fig. 3. Selected examples of the generated instances. From left to right: similar to a RUE instance, RUE-like with grid-like structures, more structured but without strong clusters and with strong clusters.

TABLE I
WE CONSIDERED THE FOLLOWING PARAMETER SPACE FOR LKH. EXCESS IS A CONTINUOUS PARAMETER THAT LIES WITHIN 0 AND 1 AND DEFAULTS TO $\frac{1}{n}$ WITH n BEING THE NUMBER OF CITIES. YET, EXCESS IS USUALLY VERY BENEFICIAL CLOSE TO 0. THE OTHER SEVEN PARAMETERS ARE DISCRETE. NONSEQUENTIAL MOVE TYPE IS THE ONLY CONSTRAINED PARAMETERS. IT MUST BE SET LARGER OR EQUAL TO THE MOVE TYPE.

Parameter	Specification	Default
Excess	$]0, 1]$	$\frac{1}{n}$
Gain23	{Yes, No}	Yes
Max Candidates	{1, . . . , 10, 1 Symmetric, . . . , 10 Symmetric}	5
Move Type	{2, . . . , 5, 3 Special, 5 Special}	5
Nonsequential Move Type	{Default, 4, . . . , 10}	Move Type + Patching A + Patching B − 1
Patching A	{1, 2, 1 Restricted, 2 Restricted, 1 Extended, 2 Extended}	1
Patching C	{0, . . . , 2, 0 Restricted, . . . , 2 Restricted, 0 Extended, . . . , 2 Extended}	0
Population Size	{0, . . . , 10}	0

one another on the novel dataset as well as standard test sets from the literature.

A. Full-set Algorithm Configuration

The parameter space Λ of LKH holds all possible parameter configurations λ, which is the Cartesian product over all the individual parameters – described in Table I, excluding possible constraints on different parameter combinations. Given a set I of n instances with $|I| = n$ the goal of full-set AAC [28] is to find one optimal configuration $\lambda^* \in \Lambda$ that minimizes the aggregated PQR10 score, i.e. formally

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \frac{1}{n} \sum_{i \in I} \text{PQR10 } (\text{LKH}_\lambda, i).$$

The notation PQR10 (LKH_λ, i) means that the PQR10 score is measured over multiple runs of LKH with j random seeds and configuration λ on instance $i \in I$. In this work, the goal is to minimize the mean PQR10 score over all instances. We use SMAC [15] as state-of-the-art in this domain to find the best global configuration. SMAC alternates between two phases; intensification and *Bayesian optimization* (BO).

During intensification, new configurations are iteratively evaluated on instances and compared against the incumbent, i.e. the best configuration at that time. A new configuration is rejected when it yields a worse PQR10 score on the instances it ran on than the incumbent. When the new configuration is evaluated on all the same instances as the incumbent and yields a lower PQR10 score, it becomes the new incumbent.

In the BO phase, promising new configurations are discovered. Here, a surrogate model, fitted on the performed evaluations during intensification, is used to predict the mean PQR10 score of a configuration over all the instances. The surrogate model uses tabular instance features as described in Section II and a configuration as input. By using local- and random search techniques, new configurations are discovered and ranked based on their *Expected Improvement* (EI). To favor exploration, this ranked list of configurations is interleaved with randomly sampled configurations. For a more detailed description of SMAC, we refer to [15].

B. Per-Instance Algorithm Configuration

Contrary to the full-set algorithm configuration, PIAC aims to find an optimal policy $\pi^*_\theta \in \Pi$ with learnable weights θ that outputs for every instance $i \in I$ an optimal configuration $\lambda^*_i \in \Lambda$, individually. Formally, one can define the policy as $\pi_\theta : I \to \Lambda$. Hence, the overall goal can be defined as finding

$$\pi^*_\theta = \arg\min_{\pi_\theta \in \Pi} \frac{1}{n} \sum_{i \in I} \text{PQR10 } (\text{LKH}_{\pi_\theta(i)}, i).$$

In our case, π_θ is learned by a deep learning model. Therefore, we train a policy model with learnable weights θ that – in its ideal case – predicts $\pi^*_\theta : i \to \lambda^*_i$ given any instance i. Further, we consider two different scenarios: *(1)* the policy model takes tabular TSP-features as input (same features as for SMAC) (feature-based), and *(2)* the policy model takes the raw instance as input (feature-free) (see Figure 4). More on this will be discussed in Chapter III-B.

In total, we consider 1 140 480 possible different discrete configurations, and one additional parameter, Excess, which is continuous (see Table I). Therefore, it is impossible to test every single feasible configuration on every instance to create a reliable training dataset but instead, we rely on reinforcement learning (see Figure 4). We tested different strategies and we found that *Proximal Policy Optimization* (PPO) [29] works best for our setting. We use the loss function $\mathcal{L}$ as defined by the authors. In most reinforcement settings, a roll-out or run consists of multiple time-steps. In

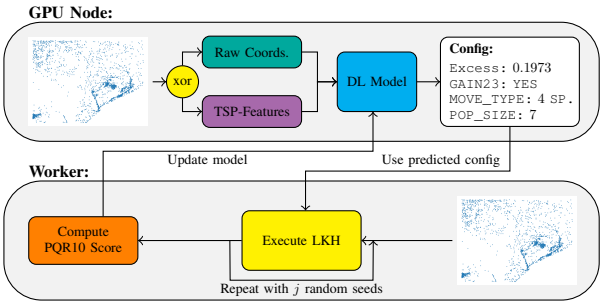


Fig. 4. The reinforcement learning process with distributed workers. First, the GPU node takes one training instance and predicts the "best" configuration. Afterward, the configuration and instance are sent to one free worker, where the instance is solved j times with j different seeds by LKH. Last, the PQR10-score is computed to update the model. Note: the DL model can be feature-free or feature-based.

our scenario, we do not have any time-steps as we consider the optimization process of LKH as a single time-step. Hence, we can simplify the loss function to

$$\mathcal{L}(\theta) = -1 \cdot \min\left(r(\theta) \cdot \hat{A}_i, \text{clip}\ (r(\theta),\ 0.8, 1.2) \cdot \hat{A}_i\right) \quad (1)$$

$$r(\theta) = \frac{\pi_\theta(\lambda_i^* \mid i)}{\pi_{\text{old}}(\lambda_i^* \mid i)}. \quad (2)$$

Here, r is the ratio between the current policy π_θ and the old policy π_{old}. The clipping function prevents from moving too quickly toward a promising policy. We used the default clipping parameters (0.8 as lower and 1.2 as upper boundary) as proposed in [29]. Next, $\hat{A}$ is the advantage defined as

$$\hat{A}_i = \mathcal{B}_i - \mathcal{R}_{\pi_i,i}$$

where $\mathcal{R}_{\pi_i,i} = \text{PQR10}\ (\text{LKH}_{\pi_\theta(i)}, i)$ and $\mathcal{B}_i = \text{PQR10}\ (\text{LKH}, i)$ with the default configuration. In other words, the advantage measures the absolute benefit of the current policy π_θ compared to the default configuration. π_{old} is the old policy that was used previously to determine $\text{PQR10}\ (\text{LKH}_{\pi(i)}, i)$. Note, that sampling a new configuration from π_θ and updating the parameters θ are two different processes.

In most *Actor-Critics*-based [30] reinforcement learning approaches, the baseline (usually referred to as V) is learned and predicted by the model. In our scenario, however, $\mathcal{B}_i$ is the PQR10 score of the default configuration on instance i which is stable and does not need to be learned by the model. Hence, we only consider an *Actor* in our setting.

IV. Experiments

A. Experimental set-up for SMAC

For the full-set configuration, we performed 15 independent, randomly seeded runs of SMAC3 [31], with each a wall time budget of 24h and 8 parallel workers. We used the `AlgorithmConfigurationFacade` in SMAC3, which closely follows the described procedure in [15]. Only the proportion of wall time budget for finding new configurations was set to 30% instead of 50%. We uniformly at random sampled 5 000 instances — which are 6.67% of all training instances — without replacement from the training set to reduce the computational load for predicting the PQR10 over all instances in the BO phase of SMAC. The surrogate model uses pre-computed tabular instance features to improve its predictive performance. From the 15 resulting incumbent configurations, we selected the configuration with the lowest PQR10 score after evaluating them on all 5 000 instances.

In total, 6 906 different configurations were considered and 150 202 times a configuration was evaluated. The final incumbents were on average evaluated on 191 instances.

B. Experimental set-up for Reinforcement Learning

For the two per-instance configuration approaches (feature-based and feature-free), we used 500 distributed workers with a maximum wall time of 24h each (see Figure 4 for a visualization of the distributed training process). The model and its learnable weights θ are located on a single server called the *GPU Node*. During training and also validation, the GPU Node takes a random instance i from the training or validation set and applies the model with the current policy π_θ to receive a configuration λ_i^*. The tuple (i, λ_i^*) is sent to a free worker which then applies LKH with j random seeds (j sampled from $\{1, \ldots, 10\}$) and configuration λ_i^* on instance i. The worker then may need up to 20 minutes to determine $\mathcal{R}_{\pi_i,i}$ — in the case of $j = 10$ and ten timeouts. π_{old} are stored at the GPU Node for later use. Afterwards, the worker will send $\mathcal{R}_{\pi_i,i}$ to the GPU node were the set $B_i := \{i, \pi_{\text{old}}, (\mathcal{R}_{\pi_i,i})\}$ is stored in a replay buffer. For training, a batch is sampled from the replay buffer to update the current policy π_θ using the process described in Chapter III-B.

We used a *Multi-Layer Perceptron* (MLP) with a ResNet-like [32] structure for the feature-based model (see Figure 5, a). The model receives all 167 standard TSP features (z-standardized based on the training set) as input [9]–[11]. As 21 features contain missing values, we performed mean imputation for these features and added 21 additional features that indicate whether a missing value was imputed. In total, the MLP model received 188 input features. Further, the MLP model consists of twelve ResNet layers with 576 hidden features. Two fully-connected layers with also 576 hidden features are used as the model's stem. Last, we used the *Gated Linear Unit* (GLU) [33] as the activation function throughout the model. As the GLU activation requires two neurons per output feature, we doubled the number of hidden neurons to 1 152 which is then reduced to 576 by the GLU activation.

For the feature-free model, we used a Transformer-based architecture very similar to [19] but with additional *Feed-Forward* layers after the *Multi-Head Attentions* (see Figure 5 b). Further, we pre-trained the model on the training instances by masking randomly some of the k-Nearest-Neighbor clusters of the embedding layer. The pre-training helps the model to pre-learn some important characteristics of TSP instances.

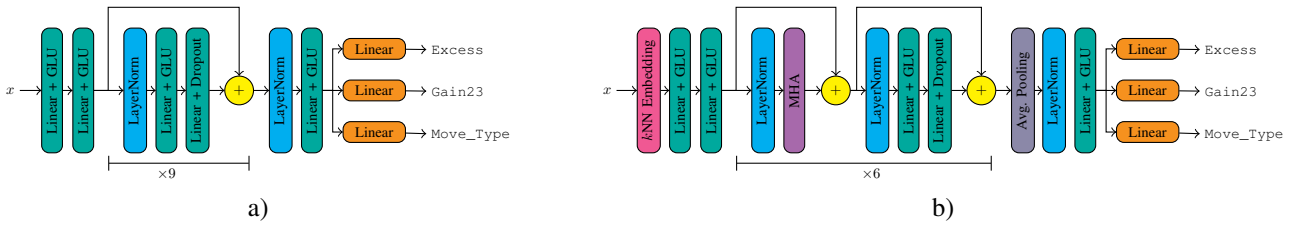


Fig. 5. This Figure illustrates the two used deep learning topologies: *a)* shows the feature-based approach, that takes the numerical TSP-features as input, with nine ResNet-blocks. *b)* is a feature-free approach, that only takes raw coordinates as input, with six *Multi-head Attention* and *Feed-Forward* blocks.

The last layer consists of multiple fully-connected layers — one for every parameter. If the parameter is categorical, the corresponding layer has one neuron per category and a *LogSoftmax* as the activation function. The outputs are the *log probabilities* which can be used directly in Equation (2) to compute the ratio. For continuous parameters, the corresponding layer contains two output neurons with Softplus activation — one for the *alpha* and one for the *beta* of a Beta distribution as it is proposed in [34]. The Beta distribution has two nice properties: *1)* it is bounded to $[0, 1]$ and can, hence, be scaled to any other range with a finite upper and lower bound, and *2)* the probability function of a beta distribution can have a variety of different shapes — ranging from bathtub-like shapes to normal-like or log-like shapes. The log probabilities for continuous parameters are calculated as outlined in [34]. Further, we used linear scaling of the search space with a lower bound of $\frac{1}{10\,000}$ and upper bound of 1 — exactly the same upper and lower bound as it was used for SMAC. Yet, we did not $\log_{10}$-scale the search space as we found that this decreases the training performance of the deep learning models.

During training, the algorithm samples from the output distributions. This guarantees the exploration of different policies. However, during evaluation, the maximum of the log probability for categorical parameters is used, and the mean of the Beta-distribution for continuous parameters. This ensures a deterministic behavior of the model during validation.

Last, we used *Adam with Weight Decay* (AdamW) [35] with a learning rate of $1e-4$, a batch size of 64, and a replay buffer size of $8\,192$ as it is commonly done. We found that two LKH runs in the first epoch are sufficient to train the model as the model predicts mostly random configurations. Yet, when the model becomes more sophisticated in its decisions, one has to increase the number of LKH runs to perform more accurate updates. We found that training with more than five runs does not yield any benefits. Therefore, the sweat-spot between training speed and accurate updates are $j = 2$ in the first epoch and $j = 5$ for all others.

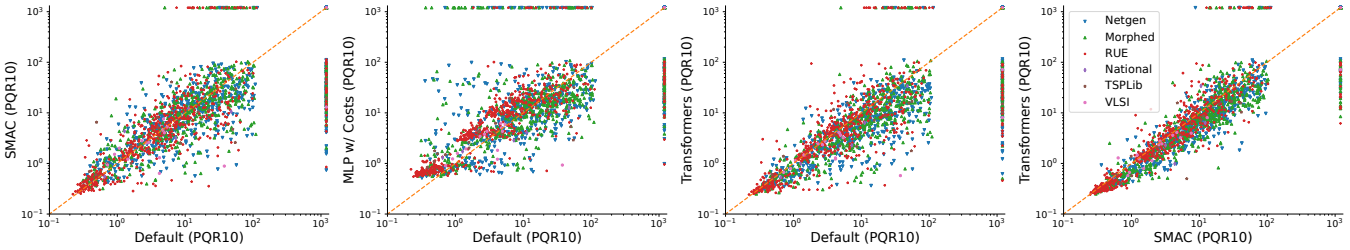
V. Results & Discussion

Table II presents the mean PQR10 scores and timeout percentage of the default configuration, the static configuration obtained with SMAC, and our two RL-based PIAC methods; MLP (feature-based) and Transformers (feature-free). For each dataset, all approaches are robustly ranked, i.e., ranked with statistical guarantees [36]. The robust ranking works as follows; For each instance set, multiple replicas ($10\,000$) are created using resampling. Over these resulting bootstrap samples the average PQR10 scores are computed and the approach which scored best in most of the bootstrap samples is ranked first. Then, all approaches that are not significantly worse (with $\alpha = 0.05$) are also ranked first. The procedure continues with the remaining approaches until all approaches are assigned a rank.

For the feature-based PIAC approach (MLP), we provide the results including feature costs. The costs reflect the time that is required to compute the TSP-features for an instance at hand. Note, that we do not include the runtime of the deep learning models themselves. We found it to be impossible in the current state to compare deep learning models that are optimized to be executed on GPUs with the runtime of LKH that is optimized to be executed on a single CPU core. However, there is a lot of research into reducing and simplifying trained deep learning models such that these complex models can be efficiently executed on small devices like mobile phones or even micro-controllers. Yet, we argue that these topics are out of scope for this paper and leave that for feature work.

In Table II it becomes evident that all used approaches clearly improve the baseline performance of LKH with default configuration. This demonstrates that there is a huge potential for optimizing the LKH and making it more competitive to other optimizers. Next, our Transformer outperforms or, at least, achieves compatible performance to the other approaches in all categories. Interestingly, all AAC and PIAC approaches as well as the default configuration are statistically tied to one another for RUE instances. We assume that different configurations have little impact on the performance as the random-uniform nature of these instances does not provide any structures that can be exploited by different configurations.

On one hand, this is surprising as SMAC achieves state-of-the-art performance in most AAC settings. On the other hand, it is expected for PIAC approaches to (theoretically) achieve better performance in comparison to set-based AAC as PIAC approaches can select from the whole configuration space while SMAC only outputs a single configuration for the whole set. However, PIAC training is especially difficult when the configuration space is large or even infinite.



a) b) c) d)

Fig. 6. Runtime comparison between different AAC approaches and the default LKH configuration (a-c). d) Runtime comparison between SMAC and the transformer approach. The equilibrium line (dotted in orange) symbolizes an equal performance between the two approaches while instances above (below) the equilibrium line indicate a better (worse) performance of the approach plotted on the x-axis the one on the y-axis.

TABLE II

RESULTS SO FAR. MEAN PQR10 FOR DIFFERENT INSTANCE GROUPS AND RELATIVE TIMEOUT (T/O) SCORES. AS T/O ARE COUNTED PQR10 SCORES THAT ARE 1 200. THE ROBUST RANK IS SHOWN IN BRACKETS. IF TWO ALGORITHMS ARE IN THE SAME RANK, THEY ARE STATISTICALLY TIED.

Instance Set	**#Cities**	**#Instances**	**Default Config** PQR10	T/o	**SMAC** PQR10	T/o	**MLP w/ Costs** PQR10	T/o	**Transformer** PQR10	T/o
Validation	2 000	25 000	342.51 (4)	27.1%	257.44 (2)	19.9%	287.76 (3)	22.1%	**228.74 (1)**	17.5%
Netgen	500	150	77.49 (3)	6.0%	5.32 (2)	0.0%	143.53 (4)	11.3%	**3.37 (1)**	0.0%
Netgen	1 000	150	180.25 (2)	14.0%	**41.85 (1)**	2.7%	139.80 (2)	10.7%	**31.40 (1)**	2.0%
Netgen	1 500	150	283.89 (3)	22.0%	**140.67 (1)**	10.0%	193.49 (2)	14.7%	**134.68 (1)**	10.0%
Netgen	2 000	150	464.04 (3)	36.7%	**235.85 (1)**	17.3%	311.84 (2)	24.0%	**215.51 (1)**	16.0%
Netgen	(All)	600	251.42 (3)	19.7%	**105.92 (1)**	7.5%	197.17 (2)	15.2%	**96.24 (1)**	7.0%
Morphed	500	150	37.20 (3)	2.7%	19.09 (2)	1.3%	205.70 (4)	16.7%	**2.55 (1)**	0.0%
Morphed	1 000	150	151.11 (2)	11.3%	139.98 (2)	10.7%	147.25 (2)	11.3%	**82.62 (1)**	6.0%
Morphed	1 500	150	347.82 (3)	27.3%	213.47 (2)	16.0%	**172.56 (1)**	12.7%	**163.58 (1)**	12.0%
Morphed	2 000	150	481.36 (2)	38.0%	**331.04 (1)**	25.3%	**290.03 (1)**	22.0%	**308.29 (1)**	24.0%
Morphed	(All)	600	254.37 (3)	19.8%	175.89 (2)	13.3%	203.89 (2)	15.7%	**139.26 (1)**	10.5%
RUE	500	150	9.13 (2)	0.7%	**0.68 (1)**	0.0%	0.99 (2)	0.0%	**0.63 (1)**	0.0%
RUE	1 000	150	86.25 (2)	6.7%	**22.55 (1)**	1.3%	**48.12 (1)**	3.3%	**36.98 (1)**	2.7%
RUE	1 500	150	**156.48 (1)**	12.0%	**141.27 (1)**	10.7%	**161.56 (1)**	12.0%	**132.95 (1)**	10.0%
RUE	2 000	150	**339.52 (1)**	26.7%	**337.18 (1)**	26.7%	**343.10 (1)**	26.7%	**345.60 (1)**	27.3%
RUE	(All)	600	**147.84 (1)**	11.5%	**125.42 (1)**	9.7%	**138.44 (1)**	10.5%	**129.04 (1)**	10.0%
National	734-1 979	5	13.60 (2)	0.0%	**6.17 (1)**	0.0%	**4.52 (1)**	0.0%	**8.85 (1)**	0.0%
TSPLib	574-1 889	21	182.59 (3)	14.3%	66.44 (2)	4.8%	120.84 (2)	9.5%	**62.80 (1)**	4.8%
VLSI	662-1 973	17	216.35 (2)	17.6%	144.18 (2)	11.8%	214.27 (2)	17.6%	**77.02 (1)**	5.9%
Real World	(All)	43	176.29 (3)	14.0%	90.16 (2)	7.0%	144.25 (2)	11.6%	**62.15 (1)**	4.7%
Test Set	(All)	1843	216.91 (4)	16.9%	134.68 (2)	10.1%	179.00 (3)	13.7%	**120.13 (1)**	9.1%

Next, the MLP without including feature costs provides comparable performance in most categories. However, when including the feature costs in the runtime, the performance of the same model becomes noticeably weaker — demonstrating again the need to investigate the potential of feature-free AAS and AAC also in other domains.

Last, we found that most of the performance gain is to the reduced number of timeouts. This is especially true for SMAC as the instances above and below the equilibrium line in Figure 6 a) are similarly distributed. However, in Figure 6 c-d) it becomes evident, that the PIAC approaches not only decrease the number of timeouts but also predict configurations that improve the runtime in comparison to the default configuration and also SMAC as many instances are below the equilibrium line.

VI. CONCLUSION & OUTLOOK

Our RL-based PIAC approaches have shown their ability to improve the performance of LKH. Applying them to other TSP solvers, such as EAX, would be an interesting direction for future work. The same holds for applying our approaches to other AI domains where AAC has an impact, like SAT and MIP solving, in order to – ideally – advance those fields further and to gain more insights into how RL-based PIAC performance generalizes. One limitation of the current implementation of our PIAC pipeline is that there is a discrepancy between the hardware for the neural network that infers the configuration (one GPU) and the LHK solver (one CPU-core). Composing and evaluating a kindred PIAC pipeline is possible, but requires a thorough analysis of how the NNs should be efficiently transferred (using compilation, pruning, and distillation), which we consider to be out of scope for this paper. The value of the cutoff has a large influence on the measured mean performance of solvers [12]. To overcome this, a multi-objective approach to configuration can be explored where the expectancy to solve an instance is set out against the actual running time.

Another contribution is the TSP World-based benchmark set that we used to fit our policies. Besides making this benchmark suite available to the community, we are also

planning to provide an instance generator to produce arbitrarily many different TSP instances that are measurably diverse, but also of varying size. The latter is particularly of interest when investigating how PIAC approaches are able to respond to the scaling of instances.

To conclude, we created a novel and highly diverse dataset for training and benchmarking TSP solvers. Further, we propose two novel deep reinforcement-based PIAC approaches, feature- and feature-free, that can clearly outperform current AAC approaches. The innovation of these approaches lies especially within the fact that our proposed methods can select from the full configuration space and are not limited to a pre-defined set of configurations.

REFERENCES

[1] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, Princeton, NJ, USA, 2007.

[2] R. Radharamanan and L. Choi, "A branch and bound algorithm for the travelling salesman and the transportation routing problems," *Computers & Industrial Engineering*, vol. 11, no. 1, 1986.

[3] Y. Nagata and S. Kobayashi, "A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Traveling Salesman Problem," *INFORMS Journal on Computing*, vol. 25, no. 2, 2013.

[4] K. Helsgaun, "An effective implementation of the Lin–Kernighan traveling salesman heuristic," *European Journal of Operational Research*, vol. 126, no. 1, 2000.

[5] J. Dubois-Lacoste, H. H. Hoos, and T. Stützle, "On the Empirical Scaling Behaviour of State-of-the-art Local Search Algorithms for the Euclidean TSP," in *Proc. of the Genetic and Evolutionary Computation Conference*, ser. GECCO'15, 2015.

[6] S. Meisel, C. Grimme, J. Bossek, M. Wölck, G. Rudolph, and H. Trautmann, "Evaluation of a multi-objective ea on benchmark instances for dynamic routing of a vehicle," in *Proc. of the Genetic and Evolutionary Computation Conference*, ser. GECCO'15, 2015.

[7] P. Kerschke, L. Kotthoff, J. Bossek, H. H. Hoos, and H. Trautmann, "Leveraging TSP Solver Complementarity through Machine Learning," *Evolutionary Computation (ECJ)*, vol. 26, no. 4, 2018.

[8] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated Algorithm Selection: Survey and Perspectives," *Evolutionary Computation (ECJ)*, vol. 27, no. 1, 2019.

[9] O. Mersmann, B. Bischl, J. Bossek, H. Trautmann, M. Wagner, and F. Neumann, "Local Search and the Traveling Salesman Problem: A Feature-Based Characterization of Problem Hardness," in *Proc. of the 6th International Conference on Learning and Intelligent Optimization (LION)*, vol. 7219, January 2012.

[10] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm Runtime Prediction: Methods & Evaluation," *Artificial Intelligence Journal (AIJ)*, vol. 206, 2014.

[11] J. Pihera and N. Musliu, "Application of machine learning to algorithm selection for TSP," in *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014*, 2014.

[12] J. Bossek, P. Kerschke, and H. Trautmann, "Anytime behavior of inexact tsp solvers and perspectives for automated algorithm selection," in *Proc. of the IEEE Congress on Evolutionary Computation (CEC)*, Glasgow, UK, 2020.

[13] J. Styles and H. H. Hoos, "Using Racing to Automatically Configure Algorithms for Scaling Performance," in *Learning and Intelligent Optimization*, G. Nicosia and P. Pardalos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, vol. 7997.

[14] J. Rasku, N. Musliu, and T. Kärkkäinen, "On automatic algorithm configuration of vehicle routing problem solvers," *Journal on Vehicle Routing Algorithms*, vol. 2, no. 1-4, Dec. 2019.

[15] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential Model-Based Optimization for General Algorithm Configuration," in *Proc. of the 5th International Conference on Learning and Intelligent Optimization (LION)*, 2011, vol. 6683.

[16] S. Adriaensen, A. Biedenkapp, G. Shala, N. Awad, T. Eimer, M. Lindauer, and F. Hutter, "Automated Dynamic Algorithm Configuration," *Journal of Artificial Intelligence Research*, vol. 75, Dec. 2022.

[17] L. Xu, H. Hoos, and K. Leyton-Brown, "Hydra: Automatically Configuring Algorithms for Portfolio-Based Selection," *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 24, Jul. 2010.

[18] Y. Malitsky and M. Sellmann, "Instance-specific algorithm configuration as a method for non-model-based portfolio generation," in *Proc. of the 9th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, ser. CPAIOR'12, 2012.

[19] M. V. Seiler, R. P. Prager, P. Kerschke, and H. Trautmann, "A collection of deep learning-based feature-free approaches for characterizing single-objective continuous fitness landscapes," in *Proc. of the Genetic and Evolutionary Computation Conference*, ser. GECCO'22, 2022.

[20] M. Seiler, J. Pohl, J. Bossek, P. Kerschke, and H. Trautmann, "Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2020.

[21] J. Heins, J. Bossek, J. Pohl, M. Seiler, H. Trautmann, and P. Kerschke, "A study on the effects of normalized tsp features for automated algorithm selection," *Theoretical Computer Science*, vol. 940, 2023.

[22] J. Bossek and H. Trautmann, "Evolving Instances for Maximizing Performance Differences of State-of-the-Art Inexact TSP Solvers," in *Proc. of the 10th International Conference on Learning and Intelligent Optimization (LION)*, vol. 10079 LNCS, Ischia, Italy, 2016.

[23] R. Sinott, "Virtues of the haversine," *Sky and Telescope*, vol. 68, 1984.

[24] J. Bossek, *salesperson: Computation of Instance Features and R Interface to the State-of-the-Art Exact and Inexact Solvers for the Traveling Salesperson Problem*, 2017, R package version 1.0.0.

[25] J. Heins, J. Bossek, J. Pohl, M. Seiler, H. Trautmann, and P. Kerschke, "On the potential of normalized tsp features for automated algorithm selection," in *Proc. of the 16^{th} ACM/SIGEVO Conference on Foundations of Genetic Algorithms (FOGA XVI)*, 2021.

[26] P. Kerschke, J. Bossek, and H. Trautmann, "Parameterization of state-of-the-art performance indicators: A robustness study based on inexact TSP solvers," in *Proc. of the Genetic and Evolutionary Computation Conference Companion*, Jul. 2018.

[27] J. Bossek and H. Trautmann, "Multi-Objective Performance Measurement: Alternatives to PAR10 and Expected Running Time," in *Proc. of the 12th International Conference on Learning and Intelligent Optimization (LION)*, vol. 11353, Kalamata, Greece, 2019.

[28] H. H. Hoos, "Automated algorithm configuration and parameter tuning," in *Autonomous Search*, 2012.

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[30] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, 1983.

[31] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, and D. Deng, "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization," *Journal of Machine Learning Research*, vol. 23, 2022.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. of the IEEE conference on computer vision and pattern recognition*, 2016.

[33] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *International conference on machine learning*. PMLR, 2017.

[34] I. G. Petrazzini and E. A. Antonelo, "Proximal policy optimization with continuous bounded action space via the beta distribution," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2021.

[35] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.

[36] C. Fawcett, M. Vallati, H. H. Hoos, and A. E. Gerevini, "Competitions in AI – Robustly Ranking Solvers Using Statistical Resampling," Aug. 2023.