Optimization Algorithm Behavior Modeling: A Study on the Traveling Salesman Problem

Qi Qi
School of Computer Science and Technology
University of Science and Technology of China
Hefei, Anhui, China
qqi@mail.ustc.edu.cn

Thomas Weise *
Institute of Applied Optimization
Faculty of Computer Science and Technology
Hefei University
Hefei, Anhui, China
tweise@hfuu.edu.cn

Bin Li
Dep. of Electronic Eng. and Information Sci.
School of Information Science and Technology
University of Science and Technology of China
Hefei, Anhui, China
binli@ustc.edu.cn

***Abstract*—Assume that there was an accurate model which can predict the solution quality that an optimization algorithm will reach if granted a certain amount of runtime on a given problem. Immediately, several applications in benchmarking, performance comparison, algorithm selection, algorithm analysis, and algorithm comparison arise. In this paper, two ways to obtain such models are introduced. The first one is by fitting nonlinear curves with fixed semantics and the second one is by training artificial neural networks on benchmark results. To validate the effectiveness of the models obtained by the two ways, a variety of use cases are investigated, including: *1)* the interpretation of fitted curves in terms of their parameters using their fixed semantics in order to better understand the algorithm performance and problem hardness, *2)* the classification of behavior data to algorithms, i.e., the detection of which algorithm was used to solve a given problem just according to its runtime behavior, *3)* the prediction of how an algorithm may perform on a yet-unseen problem instance with yet-unseen features, based on its performance on other problems, and *4)* the prediction of future algorithm performance based on models fitted to an (earlier) subset of the measured data. We present these use cases in a case study on the Traveling Salesman Problem. The experiment results show that all use cases are viable, easy-to-realize, and reliable.**

***Keywords*—Optimization, Benchmarking, Modeling, Model Fitting, Curve Fitting, Regression, Clustering, Classification, Experimentation, Traveling Salesman Problem**

I. Introduction

Most of the available optimization algorithms are *anytime algorithms* [2]: Evolutionary Algorithms (EAs) [3], [4], Local Search [4], Ant Colony Optimization [4], and even some exact methods like Branch and Bound all belong to this category. Anytime algorithms can provide an approximate solution for a problem at any point during their execution. The quality of this approximation may improve over time. Many machine learning approaches such as backpropagation [5] or k-means clustering also proceed in this manner.

In order to understand (and compare or benchmark) the performance of such algorithms, information about their whole runtime behavior is needed [6]. When applying an algorithm to a problem instance, such information can be collected as a sequence of tuples (t_i, q_i) relating an elapsed amount t_i of time to the quality q_i of the best solution discovered within t_i.

In this work, we show how such time-quality relationships in form of raw data can be condensed and represented as simple functional models. We show that this concept is straightforward, can easily be realized, has many advantages, and enables several important applications. We make the following contributions:

1) We introduce two ways of modeling algorithm runtime behavior into functional time-quality relationships, namely function fitting and artificial neural network (ANN) training.
2) We show that a classifier can be trained that, with high accuracy, can determine which algorithm was used to solve an unknown problem based on the parameters of the (fitted-curve) model describing the algorithm behavior and the model residuals.
3) We show how the complete runtime behavior of algorithms can be predicted on yet unseen problem instances by predicting the model parameters.
4) We show how the future behavior of an algorithm can be forecasted by a model fitted to its behavior up to now.
5) All use cases are demonstrated on the Traveling Salesman Problem (TSP), which is one of the most well-known classical optimization problems.

The remainder of this paper is organized as follows. In Section II we discuss the related work on time-quality relationships analysis. We then explain our two modelling procedures in-depth in Section III. In Section IV, the modeling procedure and subsequent analysis steps are applied to an example experiment on the TSP. A summary and future work are given in Section V.

II. Related Work

If an anytime algorithm *A* provides a better final solution than another anytime algorithm *B*, does this make *A* better? The traditional answer would be yes, but what if the best guess of *B* is better for a long time until *A* finally overtakes it? Due to their nature, anytime algorithms should not only be

* Corresponding author.

978–1–5386-4362-4/18/9/$31.00 © 2018 IEEE

assessed by a final solution and runtime requirement, but by their entire runtime behavior. This realization is fundamental for any benchmarking and experiments in this domain.

If we aim to compare the complete runtime behavior of algorithms, more data needs to be collected in the experiments. Several automated tools have been developed to compute statistics and draw diagrams based on such comprehensive datasets. None of them considers modeling the algorithm runtime behaviors, although this has several striking benefits.

The TSP Suite [6] provides a complete tool chain for optimization algorithm evaluation on the TSP. It allows for the implementation, testing, experiment execution, to the generation of freely-configurable reports with comprehensive algorithm ranking. However, it cannot be used to model the behavior of algorithms and thus does neither allow for predictions or generalizations of the accumulated information.

In [7], a feature learning method based on Slow Feature Analysis for analyzing the discriminative collective behavior of EAs has been proposed. It turns out that algorithm runtime behavior can be described with surprisingly few features and that this description is stable and can be used to distinguish both different algorithms and different problems. While our approach models the algorithm behavior in terms of its approximation quality over time, [7] focuses on the progress in the search space.

Hutter et al. [8] propose a novel approach for assessing the importance of hyperparameters by computing marginals of random forest predictions first and then qualifying the importance of main effects and interaction effects through functional ANOVA. We introduce a method for comparing any set of entirely different algorithms with completely different parameters. Hutter et al. [8] demonstrate that most of the variability of algorithm performance is often caused by only few hyperparameters. We first focus on modeling algorithm performance and then, in a second step, extract various kinds of information from these models.

In [9], Hutter at el.assess and advance the state of the art in predicting the performance of algorithms for hard problems. They propose new techniques for building predictive models, with a particular focus on improving the prediction accuracy for parameterized algorithms, and also introduce a wealth of new features for three of the most widely studied NP-hard problems, namely Satisfiability (SAT), Mixed Integer Programming (MIP), and the TSP. Based on these models obtained from experiments, we can predict the parameters of models on entirely new problem instances based on the instance features *as well as* how long it takes to solve them.

III. Modeling Procedures and Applications

The progress of a run of an *anytime algorithm* on a particular problem instance can be characterized by a sequence $((t_1, q_1), (t_2, q_2), \ldots, (t_{n_s}, q_{n_s}))$ of samples (t_i, q_i) of solution quality values q_i together with the runtime t_i at which they were attained. Assume that we perform several independent runs with each of n_A algorithms on n_I problem instances. We introduce two methods to condense the time-quality relationships measured in our experiments as mathematical functions. These methods need to meet three major challenges:

1) The models and data are highly non-linear and contain exponential relationships.
2) The data may cover a wide range of scale, spanning from large to very small objective values.
3) The overall runtime for the modeling procedures is limited, as a user would probably not wait much longer than one hour for it to finish for all models on a reasonably-sized dataset.

Without limiting the generality, we assume that the investigated optimization algorithms perform minimization with positive objective values, i.e., best-so-far objective values which decrease over time but never reach below 0[1]. As quality metric for models, we use a scaled error sum $\Phi(M)$ in order to ensure that all measure points equally contribute to the shape of the model M. We refer to the values of this metric as *residuals*. Without weights, the model shape would likely be determined largely by the first few points in the datasets, which usually have the largest objective values by far.

$$\Phi(M) = \frac{1}{n_s} \sum_{i=1}^{n_s} \frac{(M(t_i) - q_i)^2}{q_i} \tag{1}$$

where $M(t_i)$ is the value calculated by the model for time step t_i. In case that $q_i = 0$, we divide by 0.5 times the smallest non-zero objective value.

For all of our implementations and analyses in the following, we use out-of-the-box packages from the `R` language [10].

A. Curve Fitting

It is our experience that the time-quality relationships in experiments with optimization methods often resemble sigmoidal curves. After a certain setup time, the algorithm may improve the solution quality rather quickly for some period. Then, more and more time is required for smaller and smaller improvements, similar to the law of *diminishing returns* [11] in the economy. The exact nature of this sigmoidal behavior may be different from situation to situation. We therefore define four $\curlyvee$-shaped models:

$$\begin{aligned}
\text{Logistic Model (LGM):}\ & \alpha_1 + {}^{\alpha_2}/_{(1+\alpha_3 * x^{\alpha_4})} \\
\text{Decay Model (DCM):}\ & \alpha_1 + \alpha_2 * \exp\left(\alpha_3 * t^{\alpha_4}\right) \\
\text{Exp-Linear Model (ELM):}\ & \alpha_1 + \alpha_2 * \exp\left(\alpha_3 * \ln\left(t + \alpha_4\right)\right) \\
\text{Gompertz Model (GPM):}\ & \alpha_1 + \alpha_2 * \exp\left(\alpha_3 * \exp\left(\alpha_4 * t\right)\right)
\end{aligned}$$

All models have four parameters α_1 to α_4. Each model has two curve shapes resulting from the sign of parameter α_2: We call the shapes with positive α_2-value *P*-shapes. In them, α_1 represents the asymptotic best objective value that the modeled process would reach if infinite time was granted. If α_2 is negative (*N*-shapes), α_1 represents the expected objective value of the very first generated solution. The

[1] Some algorithms like Simulated Annealing may "lose" their best solution, but we assume that reasonable implementations will nevertheless remember it in an additional variable and always return the best-ever discovered solution to the user.

selected model shape then determines the meaning of the other parameters as well.

It is possible to compare the shapes of two instances of the same model type based on their parameter values without necessarily needing to plot them, which allows for automating this process. Below we provide an example for interpreting the model parameters of the positive-α_2 shape of the Gompertz model (GPMP).

- α_1: The vertical offset of the decreasing curve, i.e., the asymptotic modeled quality of the best solution.
- α_2: The vertical range of the curve, and thus, together with α_1, the quality of the initial solution.
- α_3: Determines the steepness of the decreasing curve, the larger the value, the deeper the curve, thus, the faster learning rates in algorithm.
- α_4: Determines the horizontal translation of the decreasing curve, i.e., when algorithms begins to find better solutions quickly: the larger the value, the earlier this happens.

The standard approach for non-linear curve fitting is the Levenberg-Marquardt (LM) algorithm [12]. Due to the special characteristics of our models and data, multiple restarts of the LM algorithm and an intelligent initialization strategy are required to produce good fittings. First, we attempt to get a crude estimate of the four parameters of a model as starting point for the LM algorithm. We therefore numerically solve a system of four non-linear equations of four unknowns (the parameters), for which we randomly select four data points from a data set.

In some situations this leads to bad results. In cases where no reasonably good fitting is found within ten independent trials, our system guesses the starting point by using the semantics of the parameters, for instance, the knowledge that α_2 is the vertical range (in all models except ELMN) and α_1 either is the horizontal starting point at t_1 or asymptote for $t \to \infty$ of the curves, which can easily be determined from a dataset. If this strategy should also fail ten times in a row, the initial model parameters are randomly sampled from a Gaussian distribution ($\mathcal{N}(0,1)$).

In an automated fitting procedure, all four models can be fitted to a given data set and from the fitting results, the fitting with the smallest residual can be chosen automatically.

B. Artificial Neural Networks

The second modeling technique we investigate are artificial neural networks (ANNs) [13]. Their advantage compared to the curve-fitting approach is that they can represent a much wider range of behaviors. Their disadvantages are that they have significantly more parameters and that the semantics of these parameters are too complex to be manually interpreted.

In our experiments, we use feed-forward ANNs with a single hidden layer, i.e., 3-layer perceptrons. In a set of preliminary experiments, ANNs with 6 neurons on the hidden layer showed the best performance. As there is one input node (for t-values) with a bias and connected to 6 hidden neurons, each having a bias value and being in turn connected to the single output neuron (for approximated q-values), there are $1 + 6 + 6 + 6 = 19$ weights. We use the ANN package `nnet` [14] from the `R` language for training the models. To achieve stable results, for each instance we run the ANN training 20 times with different random seeds and choose the model with smallest residual Φ.

C. Algorithm and Instance Classification

If an optimization process is modeled with the curve fitting approach, then we can assume that the parameter values of the fitted models should be different for different algorithm setups. This gives rise to the idea of behavioral forensics, to the question *Given the data points collected from a run of an unidentified algorithm setup on an unknown problem instance, is it possible to identify the algorithm which was used in the run?*

This question corresponds to a classification problem where each of the previously obtained models is labeled with the algorithm setup it belongs to. Each element to be classified has five features, namely α_1, α_2, α_3, α_4, and the residual Φ, which can be obtained from the raw experimental data without knowing the original algorithm setup and problem instance. We investigate three popular classification methods for this purpose: Support Vector Machines (SVMs), Random Forests, and XGBoost.

In our experiments, we use 80% of the models as training data and the remaining 20% as test data. We then train the classifiers discussed above using 10-fold cross-validation.

D. Model Parameter Prediction

The complementary question to data forensics is model prediction. If the model parameters exhibit clear trends with respect to instance features or algorithm parameters, these trends can be used for prediction and interpolation.

In other words, we want to obtain an ANN which can predict the parameters that a model representing the algorithm performance would have on a yet unseen problem. With this, it would be possible to estimate which solution qualities the algorithm could produce for *arbitrary* points in their runtime on the new problem. This way, we could predict the full behavior of a particular setup of an algorithm on new instances based on the measured algorithm behavior on already investigated instances.

We employ the ANN implementation of `R` with a linear activation function and use grid search for weight decay and the number of neurons in the hidden layer, using the metric Root Mean Square Error (RMSE) to choose the optimal model. We train ANNs with 1 hidden layer of 6 neurons. If this application is successful, the trained ANNs should be able to compute parameter values close to the average of the parameters that the actual fitted models would have on instances with the same features.

E. Prediction of Future Progress

One nice feature of representing algorithm behavior as function is that we can compute, for any point in time, which solution quality the algorithm likely has obtained. This also holds for the *future*. Assume that we have a (slowly-running)

optimization process solving a given problem. Based on the models, we could try to predict how the algorithm will further progress.[2]

Since the training data used in this application naturally stems from the earlier phase of the optimization process, it usually contains larger objective values whereas the solution qualities in the test data stemming from the latter phase of the search may approach or reach 0. This makes the residuals Φ on the test data rather large, even if the predictions are just slightly off. Thus, for the prediction application, we apply a modified version of the residual formula, called the prediction error Ψ:

$$\Psi(M) = \frac{1}{n_s} \sum_{i=1}^{n_s} \frac{(M(t_i) - q_i)^2}{\max\{1, q_i\}} \tag{2}$$

In order to test our idea, we set two limit values for the runtime t (here measured in *FEs*): $train_t$ and $test_t$. All data points with $t \leq train_t$ of a run are used for training, those with $train_t < t \leq test_t$ are used for testing, and those with $t > test_t$ are ignored, if any. The prediction error Ψ on the test data then determines how good the performance forecasts are.

To improve the stability of the prediction, we introduce a Weighted Model Combination (WMC) method: For each model, we set a weight according to the residual in the training data. If the sum of residuals over all eight models was $\sum_\Phi$, then the weight of a model M is $1 - \Phi(M) / \sum_\Phi$. The prediction by the overall model for a time step t_i is then the correspondingly weighted sum of the values computed by each of the eight different fitted models.

IV. Case Study: Traveling Salesman Problems

In a TSP, a salesman wants to visit n cities and return back to the city he departed at. The task is to find the city visiting order resulting in the minimal overall travel distance, which is $\mathcal{NP}$-hard. The most well-known benchmark dataset for the TSP is the TSPLib [15], from which we chose the 22 smallest symmetric instances to validate our method.

Tabu Search (TS) [16] is one of the most widely known metaheuristics for combinatorial problems. We investigate six simple setups of TS, which use objective value equality as Tabu criterion. They differ in two parameters: They may either accept the first improving non-tabu solution in each iteration or scan the whole neighborhood of the current solution and apply the best non-tabu move. The tabu list length is either 0, 10, or 10^7.

In the experiment, we conducted 20 runs for each setup, i.e., $6 * 21 * 20 = 2\,520$ runs in total, and obtained 192 981 samples. We normalize the objective values such that $q = 0$ corresponds to the global optimum and a twice-as-long tour has $q = 1$. In a manual analysis, we find that setups using the first-improvement strategy performed generally better while the tabu list length had less influence on the performance. It should be noted that we do not aim to advance the state-of-the-art on the TSP, but to investigate our proposed methods for algorithm behavior modeling.

[2] As we propose to predict the future progress of an ongoing algorithm run, the modeling and prediction needs to operate on single runs, whereas in the previous applications, we could use the concatenated list of all samples from all runs.

A. Suitable Modeling Technique

We found that the trained ANNs can better represent the TSP dataset than the fitted curves. This is caused by the wider range of behavior patterns and overall greater volatility of the optimization processes in this dataset.

The number of algorithm/instance combinations where the models fit best.

	GPMP	DCMP	LGMP	GPMP	ANNs
without ANNs	117	4	3	2	—
with ANNs	29	0	2	1	94

As can be seen in the above table, the GPMP model is the most suitable one among the curve models. We observe that the value of parameter α_2 of the fitted GPMP models becomes extremely large in some instances. As parameters α_2 indicates the range of solution qualities covered by the optimization process, outliers in this parameter can easily be detected, even without any knowledge of the algorithm or problem instance. We limited α_2 to 30 times the largest q value in a dataset for all models.

B. Model Parameters vs. Instance Features

We choose 22 benchmark instances which differ in the number n of cities and a variety of other parameters, including their geometry. Kotthoff et al, for instance, list 114 features characterizing TSP instances. We visualize the trends for the fitted parameters of model GPMP for the setup with a tabu list length of 10^7 using the first improvement strategy in Figure 1.

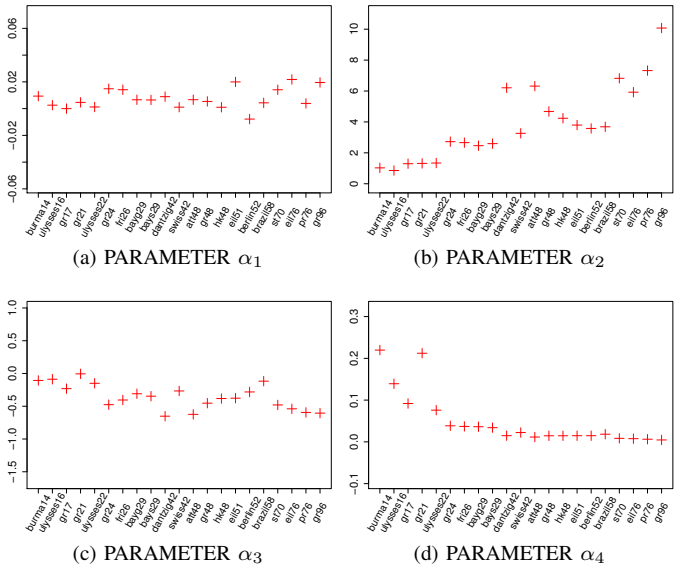
Since the problem instances are usually solved and objective value 0 is normally reached, we cannot expected the model parameter α_1 to exhibit any trend. Indeed, we can observe that it always takes on very small values around 0. The other parameters, however, do exhibit trends.

Parameter α_2 rises, since the range of possible tour lengths increases for larger problems. Parameters α_3 and α_4 decrease, meaning that the time cost of finding the optima increases with the problem scale. The development of the parameters is a bit noisy, because the scale n of a TSP instance alone is not enough to fully describe the problem hardness [17]. From this perspective, the trends are surprisingly clear.

C. Algorithm and Instance Classification

We know that the algorithm setups using the first improvement strategy solve the TSP more efficiently than those using the best improvement strategy. In Figure 2, we plot the trends of parameters α_3 and α_4 for the setups with first-improvement (f) and best-improvement (b) strategy of the GPMP model over the problem scale n.

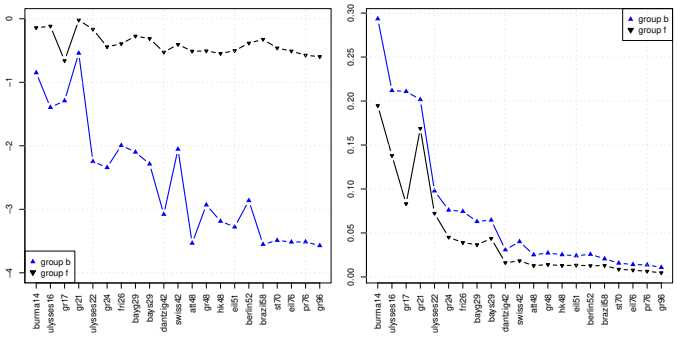
We find that both trend lines are clearly distinguishable, i.e., can confirm that the algorithm parameters and the model parameters are related and post-optimization forensics should be possible.



(a) PARAMETER α_1 (b) PARAMETER α_2

(c) PARAMETER α_3 (d) PARAMETER α_4

Fig. 1: TRENDS FOR THE MEAN VALUES OF THE FOUR PARAMETERS OF THE FITTED GPMP MODELS OVER THE NUMBER n OF CITIES OF THE PROBLEM INSTANCES FOR TABU SEARCH PICKING THE FIRST IMPROVING MOVE AND HAVING A TABU LIST LENGTH 10^7.



(a) TRENDS FOR PARAMETER α_3 (b) TRENDS FOR PARAMETER α_4

Fig. 2: THE TRENDS FOR THE PARAMETERS α_3 AND α_4 OF THE GPMP IN THE TSP FOR THE SETUPS WITH FIRST-IMPROVEMENT(f) AND BEST-IMPROVEMENT(b) STRATEGY.

We use the parameters α_1, α_2, α_3, α_4, and the residuals Φ from the fitted GPMP models as features for classification. We label all setups where the first improving moves were used with *1* and those where the best improving modes were used with 0. In the resulting binary classification problem, we use 80% of the fitted GPMP models for training and the remaining 20% for testing on this problem. We apply RF, XGBoost, SVM. Below we show the classification accuracies on the test data and find that all classifiers allow us to distinguish the different algorithm parameter settings with a very high confidence.

Accuracy: SVM: 1 RF: 1 XGBoost: 0.946

We now consider every single one of the 2520 runs as separate sample and want to compute the number n of cities of the corresponding TSP instance. We fit all eight models to each run and use their parameters as sample features. In the resulting 16-class classification problem, the best accuracy was obtained with XGBoost and was 0.64. This is much lower than in the binary algorithm parameter classification task above. However, a mis-classification here has different semantics: Classifying a 16-city problem as 17-city problem it is much less severe than classifying it as 48-city problem. Thus, for each sample classified to have m cities, we compute $|m-n|/n$, i.e., an inaccuracy measure representing the numerical nature of the classes. The arithmetic means over these values then reveal that our classification procedure was actually very precise, as the inaccuracy is below 6% for both RF and XGBoost.

Accuracy: SVM: 0.367 RF: 0.638 XGBoost: 0.648
Inaccuracy: SVM: 0.123 RF: 0.059 XGBoost: 0.059

In summary, we found that it is also possible, with high accuracy, to deduce both features of the problem *and* parameters of the algorithm applied to it from the measured performance data.

Figure 1 reveals that the fitted model parameters are related to the number n of cities in a problem instance. This time, the relationship is not as clear, for the reason that the problem scale alone does not describe the hardness of a TSP instance well. Nevertheless, we attempt to use this (noisy) relationship to predict algorithm behaviors (model parameters) for unseen instances.

We remove the instances with 48 and 90 cities to use them as test data and train ANN model predictors (with one hidden layer of 6 neurons) on the rest. For parameter α_1, we obtain a straight line around 0 which correctly and exactly describes the algorithm behavior with noise removed, as the algorithms always discover the optima. By visualization, we can see that the predicted parameters α_2 and α_4 fit well to our data, and α_3 lies on a reasonable trend line although it deviates slightly from the test data.

D. Prediction of Future Progress

From Table I, we can find that both prediction approaches defined in Section III-E are astonishingly accurate. Both methods work well with $train_t \in \{50, 100\}$. The ANN method has a smaller prediction error Ψ than the fitted curves. In the settings with $train_t = 10$, the prediction errors are nine times larger in the test interval for both methods. This is reasonable, since we only use the first ten algorithm steps to predict its future progress.

V. Conclusion

We provide two easy and general methods to represent the time-solution quality relationships of anytime algorithms: function fitting and ANN training. We show that such performance models have a wide variety of viable and easy-to-implement applications on a detailed case study on the

TABLE I: THE MEDIAN PREDICTION ERROR Ψ BOTH OF WMC AND ANN MODELS FOR SETTINGS $(train_t, test_t)$ OF TRAINING TIME LIMITS $train_t$ AND TEST TIME INTERVALS $(train_t + 1)...test_t$ FOR THE ALGORITHMS WITH BEST-IMPROVEMENT STRATEGY AND A TABU LIST LENGTH OF 10, WITH THE BETTER PREDICTOR MARKED IN Green.

	WMC			ANN		
Instance	$(10, 100)$	$(50, 100)$	$(100, 1000)$	$(10, 100)$	$(50, 100)$	$(100, 1000)$
burma14	0.00257	0.00268	0.00268	0.00119	0.00210	0.00161
ulysses16	0.00024	0.00079	0.00192	0.00119	0.00060	0.00178
gr17	0.00036	0.00069	0.00082	0.00035	0.00109	0.00067
gr21	0.00182	0.00427	0.00427	0.00686	0.00488	0.00437
ulysses22	0.00419	0.00098	0.00154	0.00167	0.00096	0.00153
gr24	0.00671	0.00146	0.00183	0.00287	0.00095	0.00125
fri26	0.00710	0.00127	0.00188	0.00736	0.00100	0.00048
bayg29	0.03006	0.00186	0.00193	0.01200	0.00123	0.00144
bays29	0.02468	0.00147	0.00001	0.00850	0.00140	0.00005
dantzig42	0.04709	0.00127	0.00287	0.03903	0.00050	0.00051
swiss42	0.03742	0.00127	0.00154	0.04487	0.00057	0.00123
att48	0.13983	0.00115	0.00113	0.08355	0.00052	0.00059
gr48	0.10958	0.00219	0.00190	0.10147	0.00083	0.00140
hk48	0.07905	0.00186	0.00588	0.11502	0.00068	0.00147
eil51	0.07627	0.00085	0.00234	0.10236	0.00032	0.00139
berlin52	0.03497	0.00102	0.00101	0.09963	0.00044	0.00037
brazil58	0.15177	0.00046	0.00048	0.14491	0.00022	0.00020
st70	0.34998	0.00115	0.01992	0.97810	0.00050	0.00120
eil76	0.26825	0.00119	0.00082	0.68198	0.00025	0.00015
pr76	0.14406	0.00344	0.00195	1.21302	0.00091	0.00070
gr96	0.21258	0.01582	0.00694	2.79420	0.00514	0.00072

TSP. They are particularly suited as basic representation of experimental data for benchmarking, performance comparison, and algorithm behavior analysis.

We find that there is a very clear relationship between the algorithm performance and the problem hardness and the parameters of the fitted models. This relationship, learned from models on available performance data, can be used to detect which algorithm was used to solve a given problem and even to predict how an algorithm will perform on a yet unseen problem instance with yet unseen features. Both fitted curves and ANNs can be used to forecast the future behavior of an algorithm if trained on its progress so far.

In our future work, we will try to improve the curve-fitting based models by fitting them in a transformed space. Our models all feature exponential relationships. We could log-scale the data to remove this exponentiation and "translate" the models back after fitting. We will also investigate applying stronger continuous optimization methods for optimizing the model parameters and hence improving the model quality. Finally, we test our approach on a variety of different optimization problems and algorithms in order to assess its utility on a broader basis and to collect more experience and data.

Acknowledgements. We acknowledge support from the National Natural Science Foundation of China under Grants 61673359, 61473271, 61150110488, and 71520107002.


```
@inproceedings{QWB2018OABMASOTTSP,
  author    = {Qi Qi and Thomas Weise and Bin Li},
  title     = {Optimization Algorithm Behavior Modeling:
               A Study on the Traveling Salesman Problem},
  booktitle = {Proceedings of the Tenth International
               Conference on Advanced Computational
               Intelligence (ICACI 2018), } # mar #
               {~29-31, 2018, Xiamen, Fujian, China},
  address   = {Los Alamitos, CA, USA},
  publisher = {IEEE Computer Society Press},
  pages     = {861--866},
  isbn      = {978-1-5386-4362-4},
  doi       = {10.1109/ICACI.2018.8377576},
}
```

REFERENCES

[1] Q. Qi, T. Weise, and B. Li, "Optimization algorithm behavior modeling: A study on the traveling salesman problem," in *Proceedings of the Tenth International Conference on Advanced Computational Intelligence (ICACI 2018), Mar. 29-31, 2018, Xiamen, Fujian, China*. Los Alamitos, CA, USA: IEEE Computer Society Press, pp. 861–866.

[2] M. S. Boddy and T. L. Dean, "Solving time-dependent planning problems," Brown University, Department of Computer Science, Providence, USA, Tech. Rep. CS-89-03, 1989. [Online]. Available: ftp://ftp.cs.brown.edu/pub/techreports/89/cs89-03.pdf

[3] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, USA: Addison-Wesley., 1989.

[4] T. Weise, *Global Optimization Algorithms – Theory and Application*. it-weise.de (self-published), 2009. [Online]. Available: http://www.it-weise.de/projects/book.pdf

[5] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[6] T. Weise, R. Chiong, K. Tang, J. Lässig, S. Tsutsui, W. Chen, Z. Michalewicz, and X. Yao, "Benchmarking optimization algorithms: An open source framework for the traveling salesman problem," *IEEE Computational Intelligence Magazine (CIM)*, vol. 9, no. 3, pp. 40–52, Aug. 2014.

[7] C. Pang, M. Wang, W. Liu, and B. Li, "Learning features for discriminative behavior analysis of evolutionary algorithms via slow feature analysis," in *18th Genetic and Evolutionary Computation Conf. (GECCO 2016), Jul. 20–24, 2016, Denver, CO, USA*. New York, USA: ACM, pp. 1437–1444.

[8] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," *31st Intl. Conf. on Machine Learning, Jun. 22-24, 2014, Beijing, China, PMLR*, vol. 32, pp. 754–762, 2014.

[9] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[10] R Development Core Team, *R: A Language and Environment for Statistical Computing*, 2008. [Online]. Available: http://www.R-project.org

[11] P. A. Samuelson and W. D. Nordhaus, *Microeconomics*. Boston, USA: McGraw-Hill Irwin, 2001.

[12] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *SIAM Journal on Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[13] B. C. Csáji, "Approximation with artificial neural networks," Master's Thesis, Faculty of Sciences, Eötvös Loránd University, Budapest, Hungary, 2001.

[14] W. N. Venables and B. D. Ripley, *Modern Applied Statistics with S*. New York: Springer, 2002.

[15] G. Reinelt, "TSPLIB 95," Universität Heidelberg, Institut für Angewandte Mathematik, Heidelberg, Germany, Tech. Rep., 1995. [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf

[16] F. Glover and E. Taillard, "A user's guide to tabu search," *Annals of Operations Research*, vol. 41, no. 1, pp. 1–28, 1993.

[17] L. Kotthoff, P. Kerschke, H. Hoos, and H. Trautmann, "Improving the state of the art in inexact TSP solving using per-instance algorithm selection," in *Learning and Intelligent Optimization, Revised Selected Papers from the 9th Intl. Conf. (LION 9), Jan. 12–15, 2015, Lille, France*, ser. LNCS, vol. 8994. Springer, pp. 202–217.