Modeling Optimization Algorithm Runtime Behavior and its Applications

Qi Qi
UBRI, School of Computer Science and Technology
University of Science and Technology of China (USTC)
Hefei, Anhui, China 230027
qqi@mail.ustc.edu.cn

Thomas Weise*
Institute of Applied Optimization
Faculty of Computer Science and Technology, Hefei University
Hefei, Anhui, China 230601
tweise@hfuu.edu.cn

Bin Li
Department of Electronic Engineering and Information Sciences
School of Information Science and Technology, USTC
Hefei, Anhui, China 230027
binli@ustc.edu.cn

ABSTRACT

We model the time-quality relationship of optimization processes by either fitting curves or training artificial neural networks. On the example of the MAX-SAT problem, we investigate *1)* the interpretation of fitted curves based on the values of their parameters using their fixed semantics, *2)* the classification of performance measurements to algorithms, i.e., the detection of which algorithm was used to solve a given problem just by its runtime behavior, *3)* the prediction of how an algorithm may perform on a yet-unseen problem with yet-unseen features based on its performance on other problems, and *4)* the prediction of future progress based on models fitted to data measured so far.

ACM Reference format:
Qi Qi, Thomas Weise, and Bin Li. 2017. Modeling Optimization Algorithm Runtime Behavior and its Applications. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 3 pages.
DOI: http://dx.doi.org/10.1145/3067695.3076042

1 INTRODUCTION & APPROACH

Most of the available optimization algorithms are *anytime algorithms* [3] which can provide an approximate solution for a problem at any point during their execution. When applying such 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. We fit curves and train artificial neural networks (ANNs) to such time-quality relationships. We show that *1)* 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 (fitted-curve) model describing the algorithm behavior (and the model residuals), *2)* how the complete runtime behavior of algorithms can be predicted on yet-unseen problem instances by predicting model parameters, *3)* how the future behavior of an algorithm can be forecasted by a model fitted on its behavior up to now.

Let us first describe our modeling approach. The quality of a model M be its fitting residual $\Phi(M) = \frac{1}{n_s}\sum_{i=1}^{n_s}\frac{(M(t_i)-q_i)^2}{q_i}$, i.e., the weigted mean square error over all n_s measured time-quality samples. Such time-quality relationships in optimization processes often resemble sigmoidal curves and we test four such models:

Name	Shortcut	Formula
Logistic Model	LGM	$A + B/(1 + \exp(C * \ln(t) + D))$
Decay Model	DCM	$A + B * \exp\left(C * t^D\right)$
Exp-Linear Model	ELM	$A + B * \exp(C * \ln(t + D))$
Gompertz Model	GPM	$A + B * \exp(C * \exp(D * t))$

All models have four parameters and two curve shapes determined mainly by the sign of parameter B: Shapes with positive B are named with suffix "P" and those with negative B with suffix "N". The standard approach for non-linear curve fitting is the Levenberg-Marquardt algorithm. Due to the special characteristics of our models and data, multiple restarts and an intelligent initialization strategy are required. ANNs are another technology suitable for condensing the time-quality relationship of optimization processes into mathematical functions. We use feed-forward ANNs with one input neuron, a single hidden layer with 6 neurons, and one output neuron, i.e., 3-layer perceptrons.

2 CASE STUDY: MAX-SAT

We use the Maximum Satisfiability Problem (MAX-SAT) with n variables as case study. We investigate six setups of a trivial hill climber with two parameters, the search operator (1, 2, or m-bit flips) and whether or not restarts are applied. We apply the algorithms to 10 groups of 10 selected benchmark instances with fixed n from SATLib [1]. We perform 20 independent runs for each algorithm-instance combination and collect a (t_i, q_i)-sample whenever a run makes an improvement as well as at the end of the runs, measuring time in Function Evaluations (*FEs*). We find that the algorithm setups using restarts are slower (due to a restart policy intentionally designed that way), but can solve up to 90% of the problem instances while those without solve 40% at most. The search operation has much smaller influence.

*corresponding author

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3076042

The success rate (SR) be the fraction of problem instances for which a model M with $\Phi(M) \leq 4$ was produced. Our model fitting procedure reaches $SR > 90\%$ for all instances, algorithms, and models, except for model LGMN and ELMN, while ANNs have $SR = 95.86\%$. The LGMP models can best represent the algorithm behavior, followed by ELMP.

2.1 Model Parameters vs. Instance Features

We take a look at the LGMP models and observe that the algorithm setups using restarts have different model parameter settings than those without. Their parameter A is smaller, which means the model predicts better asymptotic final results for $t \to \infty$. Parameter B has an upward trend for all algorithms with a rising number of variables/clauses of the MAX-SAT instances, which is to be expected since the random initial solutions likely contain more false clauses in larger instances and, hence $q_1 - q_{n_s}$ grows. Parameters C and D both approximately have a negative linear association with the instances scale under the same algorithm: The time when the algorithms "accelerate" after their "initialization phases" increases and their progress speed decreases with rising instance scale.

2.2 Algorithm and Instance Classification

We ask the question *Given the data points collected from a run of an unknown algorithm setup on an unknown problem instance, is it possible to determine the algorithm which was used in the run?* This corresponds to a deep learning application in form of 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 A, B, C, D, and the fitting residual Φ, which can be measured without knowing the algorithm setup and problem instance. We use 80% of the models as training data and the remaining 20% as test data. We train three widely used classifiers, ANN with back propagation, Support Vector Machine (SVM) with linear kernel, and Gradient Boosting Tree, using 10-fold cross-validation. We find that SVMs perform slightly better than the other classifiers. Except ANN on ELMN, all three methods always have an accuracy well above 74%. This is astonishing, as the classifiers only receive the model parameters and model residuals as input, but *have no information* about the problem instance to which the algorithms were applied. In other words, it is possible, with high confidence, to detect which algorithm was used in an experiment on an unknown problem instance!

2.3 Model Parameter Prediction

The complementary question to data forensics is model prediction. We know that the model parameters are clearly related to the instance features. We want to obtain an ANN that predicts the model parameters based on the instance scale, i.e., the number n of variables in a MAX-SAT instance. We therefore employ an ANN 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 remove the ten uf150-645 and the ten uf250-1065 instances from our dataset. We confirm that a ANN (1 hidden layer with 6 neurons) trained on the rest can compute parameter values close to the average of the actual parameters on test data. Even better, both the predicted and the average (actual) models yield a success rate of $SR = 98.3\%$. In other words, the predicted behavior models are not worse than the average of the actual models! The predicted models have a high accuracy and are not much worse on on the test- than on the training data. This means that we can predict *the complete runtime behavior* of an algorithm on an unknown problem instance with reasonable accuracy. Given the number of variables of a MAX-SAT problem, we can predict how long a 2-flip hill climber with restarts would need to find a solution with, e.g., zero, one, two or ten *false* clauses.

2.4 Prediction of Future Progress

Assume that we have a (slowly-running) optimization process solving a given problem. While the process is running, we could naturally remember its progress and even fit models now and then. Based on the models, we could try to predict how the algorithm will further progress or where the second knee point in the $\curlywedge$-shaped model is after which significant further progress would be unlikely. As we propose to predict the future progress of an ongoing algorithm run, the modeling and prediction needs to operate on single runs.

In order to test our idea, we set two limit values for the runtime t: $train_t$ 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 setting $(train_t, test_t) = (50, 100)$ stands for predicting the complete algorithm behavior during 50 *FEs* in the future based on the data points collected during the first 50 *FEs*. This can be done with a very small fitting residual (containing square errors), even though any continuous model could hardly be expected to exactly predict an algorithm whose approximation quality changes in discrete steps in a randomized fashion. Increasing the prediction interval to nine times the training interval in setting $(10, 100)$ leads to an increase in the residuals, but is still often below half of the limit which we consider as successful based on curve similarity. Scaling the time range up to $(100, 1000)$ interestingly leads to an (still acceptable) increase in Φ, probably because the overall number of points increases and the discrete nature of the changes in measured quality. Here, the ANN models often have a smaller residual than the curve-based ones, but sometimes also are outperformed by them.

3 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 deep learning applications and we prove that these applications are viable and easy-to-implement with a detailed case study on the MAX-SAT problem.

Acknowledgments We acknowledge support from the National Natural Science Foundation of China under Grants 61673359, 61150110488, and 71520107002 and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] Holger H. Hoos and Thomas Stützle. 2000. SATLIB: An Online Resource for Research on SAT. In *SAT2000 – Highlights of Satisfiability Research in the Year 2000*. IOS Press, Amsterdam, The Netherlands, 283–292.

[2] Qi Qi, Thomas Weise, and Bin Li. Modeling Optimization Algorithm Runtime Behavior and its Applications. In *The Genetic and Evolutionary Computation Conference (GECCO'17) Companion, July 15–19, 2017, Berlin, Germany*. ACM Press, New York, NY, USA. DOI: https://doi.org/10.1145/3067695.3076042

[3] Thomas Weise, Raymond Chiong, Ke Tang, Jörg Lässig, Shigeyoshi Tsutsui, Wenxiang Chen, Zbigniew Michalewicz, and Xin Yao. 2014. Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem. *IEEE Computational Intelligence Magazine* 9, 3 (2014), 40–52. DOI: https://doi.org/10.1109/MCI.2014.2326101


```
@inproceedings{QWB2017MOARBAIA,
  author    = {Qi Qi and Thomas Weise and Bin Li},
  title     = {Modeling Optimization Algorithm Runtime
               Behavior and its Applications},
  booktitle = {The Genetic and Evolutionary Computation
               Conference (GECCO'17) Companion, }
               # jul # {~15--19, 2017, Berlin, Germany},
  address   = {New York, NY, USA},
  publisher = {ACM Press},
  doi       = {10.1145/3067695.3076042}
}
```