# Strategies for Runtime Prediction and Mathematical Solvers Tuning

Michael Barry and René Schumann
*Smart Infrastructure Laboratory, Institute of Information Systems,*
*University of Applied Sciences Western Switzerland (HES-SO) Valais/Wallis, Rue de Technopole 3, 3960 Sierre, Switzerland*

Keywords: Mixed Integer Problems, Mathematical Solvers, Tuning, Runtime Prediction, Optimization, Machine Learning, Genetic Algorithm, Novelty Search.

Abstract: Mathematical solvers have evolved to become complex software and thereby have become a difficult subject for Runtime Prediction and parameter tuning. This paper studies various Machine Learning methods and data generation techniques to compare their effectiveness for both Runtime Prediction and parameter tuning. We show that machine Learning methods and Data Generation strategies that perform well for Runtime Prediction do not necessary result in better results for solver tuning. We show that Data Generation algorithms with an emphasis on exploitation combined with Random Forest is successful and random trees are effective for Runtime Prediction. We apply these methods to a hydro power model and present results from two experiments.

## 1 INTRODUCTION

Training data selection strategies aim to improve ML method performance by carefully selecting which training instances to include in the training data. This can be helpful in applications (He et al., 2008) where the data is imbalanced, i.e. some behaviors are not represented equally, or where the data must be generated by some costly process. Training Data Generation methods effectively create a bias towards training instances that have a better learning outcome. However, the extent to which an ML method can exploit this bias depends on the ML method. We argue that the *combination* of various Data Generation and ML methods must be considered, otherwise a bias is given and the performance of some ML methods may be underestimated. In this paper, we study the combination of Data Generation and ML method and apply it to a problem of parameter tuning for mathematical solvers. Mathematical solvers are complex software tools that use a variety of strategies to solve mathematical problems. Common mathematical solvers will have a good general performance, but may be fine tuned using a set of parameters for specific problems (IBM, 2016). Identifying which strategies work best for a given problem is not intuitive and solving completely is impractical. By using ML, the runtime of a given problem and solver parameters can be predicted. The best predicted runtime can then be used to solve the problem. Such a system can address two problems: *Runtime Prediction* and *Runtime Optimization*. Building a predictor (Hutter et al., 2014) will inevitably require a training set consisting of solved mathematical models. Building such a training set can be computationally expensive. However, if the predictor can be generalized to other models, or different configurations of the same model, investing computation time into a training phase pays off in the long run. This forms the training Data Generation problem, where a good training Data Generation strategy can improve a Machine Learning methods performance or reducing the size (therefore the computation time) of the training set. This paper aims to identify ideal pairs of Data Generation and ML methods applied to the problem of tuning mathematical solvers. We describe relevant background literature in Section 2 and formalize the problem in Section 3 and 4. We describe the the applied mathematical model in Section 6 and our method in Section 7. We present results from two experiments, where we describe the setup in Section 8 and results in 9.

## 2 BACKGROUND

Runtime prediction using ML is a well studied field. There are many comparisons of general ML methods applied to Runtime Prediction e.g. (Kotthoff et al., 2012; Hutter et al., 2014). However, such methods have only recently been applied to tuning math-



Barry, M. and Schumann, R.
Strategies for Runtime Prediction and Mathematical Solvers Tuning.
DOI: 10.5220/0007387606690676
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019)*, pages 669-676
ISBN: 978-989-758-350-6

ematical solvers (Kotthoff et al., 2012; Hutter et al., 2014). Currently, mostly off-the-shelf ML methods have been applied and extensively studied in (Kotthoff et al., 2012; Hutter et al., 2014). However, most advances in the field focus on tuning an existing framework for the specific problem of tuning mathematical solvers and feature selection (Xu et al., 2011). Algorithm Runtime Prediction falls into a category of ML problems where data must be produced by running the algorithm itself. This allows a method to carefully select which data is to be produced. Such a method is commonly used in active learning and has been applied in areas that suffer from insufficient or unbalanced data (Ertekin et al., 2007). Although most applications of Runtime Prediction for mathematical solvers use a random data generation strategy, recent work (Barry et al., 2018) has indicated the success when applying a novelty search approach. However, the study was only applied using Artificial Neural Networks We wish to extend this work by comparing various data generation methods applied to a variety of Machine Learning methods. Runtime Prediction has applications in job scheduling, but also Runtime Optimization (also referred to as algorithm tuning or configuration) (Hutter et al., 2014). Good Runtime Predictions allows a user to select the algorithm parameters that result in the desired algorithm behavior, most commonly a faster runtime, but also solution quality (López-Ibánez and Stützle, 2014). In this paper we intend to extend the field of Runtime Prediction by combining currently used ML methods with various Data Generation strategies.

## 3 RUNTIME PREDICTION

Runtime prediction in general concerns predicting an algorithms (or softwares) runtime based on a description of the problem and the algorithm applied. In the context of mathematical solvers, predictions are made based on a model description $m$ and a set of solver parameters $s$ applied to the solver. In addition to the resulting runtime $y$, we can describe each run $z$ as a tuple:

$$z = (m, s, y) \tag{1}$$

Where the resulting runtime $y$ is a function $f$ of the model description $m$ and a set of solver parameters $s$.

$$y = f(m, s) \tag{2}$$

Trained on a set of example runs $[z_1, ..., z_n]$, a predictor $D$ can give an estimated runtime $\hat{y}_i$.

$$\hat{y}_i = D_{z_1,...,z_n}(m_i, s_i) \tag{3}$$

As a result, the problem of Runtime Prediction is a common learning problem where a valid method should minimize the total error $E$ between the prediction $\hat{y}_i$ and actual value $y$ for a set of unseen instances $[z_n + 1, ..., z_s]$:

$$\min E_{prediction} = \min \sum_{i=n+1}^{s} | D_{z_1,...,z_n}(m_i, s_i) - f(m_i, s_i) | \tag{4}$$

## 4 SOLVER TUNING

The problem of solver tuning can be described as finding the set of solver parameters $\bar{s}$ with the lowest runtime for a given model $m$.

$$\bar{s} = \underset{s}{\operatorname{argmin}} f(m, s_1, ..., s_n) \tag{5}$$

However, running the solver for the entire set $[s_1, ..., s_n]$ to identify $\bar{s}$ is computationally expensive and requires $n$ runs to increase the solvers performance for a single run. Instead, using a predictor, one can find the set of solver parameters $\hat{s}$ with the lowest predicted runtime for a given model $m$.

$$\hat{s} = \underset{s}{\operatorname{argmin}} D_{z_1,...,z_n}(m, s_i) \tag{6}$$

The aim in solver tuning is to reduce the error between the runtime of the estimate best solver configuration $\hat{s}$ and the actual best solver configuration $\bar{s}$ for a given model $m$.

$$\min E_{tuning} = \min(f(m, \hat{s}) - f(m, \bar{s})) \tag{7}$$

## 5 TRAINING DATA

For both problems we wish to choose a predictor $D$ and training set $[z_1, ..., z_n]$ that produces the smallest error $E$. The more computation time is given to the training data generation stage, the more data is available and therefore we expect a higher accuracy for the predictor. However, as producing the training set is computationally expensive, we have the conflicting objective to reduce the size $n$ of the training set as much as possible.

## 6 APPLICATION

The aim of our approach is to create a predictor that can predict with a high accuracy the runtime for a given model instance and solver parameters. Secondly, we use the predictions to select the best solver parameters for a given model instance. We apply

these methods for solving a Hydro Power Operations model, that aims to identify when to utilize its water reserves to optimize its revenue based on energy market prices. The Mixed Integer Problem (MIP) model is used as it is highly configurable, varying greatly in complexity based on its configuration. Details are given in our previous work (Barry et al., 2015; Barry and Schumann, 2015). Model configurations include:

**Plant Topology:** The topologies of a hydro power plant describes details such as the size and number of the reservoir, turbines and water inflows.

**Time Period:** Different time periods results in varying data such as how much water flows into the reservoir and expected market prices.

To solve the model we use the mathematical solver CPLEX (Version 12.7) (GAM, 2017), as its performance is commonly accepted and represents an industry standard. CPLEX has a large set of parameters to configure. Hutter et al. (Hutter et al., 2010) identified a total of 76 parameters that can increase the performance of CPLEX. We reduce this set to the following parameters as initial results and literature (Hutter et al., 2009) show that they are good candidates to increase a solvers performance:

**startalg:** [1,2,3,4,5] Selects the algorithm used for the relaxation of the MIP

**subalg:** [1,2,3,4,5] Selects the algorithm used for the linear subproblem.

**heurfreq:** [-1,0] Selects how often CPLEX applies heuristics. This can be set manually to a specific value, but we only consider turning it off (-1) or setting it to automatic (0).

**mipsearch:** [1,2] Selects the search strategy for the branch and cut phase.

**cuts:** [-1,1,2,3] Selects how aggressively cuts are generated, ranging from not at all (-1) to very aggressive (3). Values 4 and 5 were omitted as they did not yield any speedup for any instances.

# 7 METHOD

The Data Generation strategy selects sets of model instances and solver parameters, which are then passed to the CPLEX solver. The resulting runtime for each set of inputs is then used to train the predictor. The overall Runtime Prediction and Runtime Optimization process is shown in Figure 1. For a given model instance and solver parameters, the predictor predicts the runtime. To optimize the runtime, the predictor predicts the runtime for every set of possible solver parameters. The best runtime with the corresponding solver parameters is then selected to run the CPLEX solver for the given model instance.

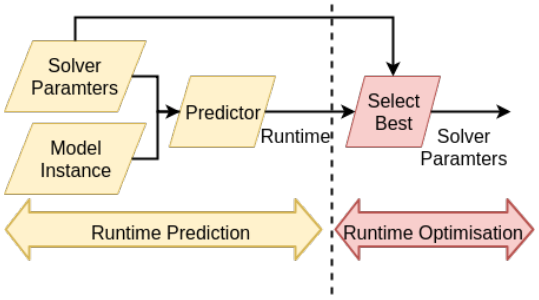


Figure 1: The overall Runtime Prediction and Runtime Optimization process.

## 7.1 Data Generation Strategies

To select which sets of model instances and solver parameters to use as training instances we consider three training Data Generation strategies: *Random*, *Optimal Runtime* and *Novelty*. Ideally, the strategy produces a set of training instances that is:

**Small:** Each training instance must be passed to the CPLEX solver to ascertain the solvers runtime, which is computationally expensive. Reducing the size of the training data reduces the computation time for the training phase.

**Representative:** The training data is a sample of the entire search space. For a predictor trained on this sample to be accurate, the sample must be representative of the entire search space.

### 7.1.1 Evolutionary Algorithm

All algorithms for the Data Generation are based on a Genetic Algorithm which can be summarized as shown in the listing of Algorithm 1. Genetic algorithms are distinctive from one and another based on the implementation of the representation, initialization, evaluation, selection, breeding (crossover and mutation operators) and termination criteria. In this paper, the first two algorithms (Minimal Runtime GA and Novelty GA) differ only by the evaluation function. The third algorithm only acts as a representation of random selection, where each generation a random set of individuals are added. As parameters for these algorithms we use a population size of 50, a mutation to crossover rate of 90% and a selection rate of 20 %. The parameters were chosen after some initial testing. We describe the general GA below, with more details given for the specific evaluation function in sections 7.1.2 and 7.1.3.

Algorithm 1: Genetic Algorithm.

```
P = initialise()
while Termination() do
    evaluate(P)
    selection(P)
    breed(P)
```

For more details on GAs, we refer to (Chawdhry et al., 2012).

**Representation:** Each individual represents a model and solver configuration as a string of numbers: $\{m_1, \ldots\ m_n, s_1,\ldots,s_k\}$, where each $m$ is a model configuration, $s$ is a solver configuration, $n$ is the total number of model configuration and $k$ is the total number of solver configurations.

**Initialization:** Random initialization is used, where each gene $m$ and $s$ are assigned a random value.

**Selection:** A roulette wheel selection is used where each individual is assigned a *pocket* proportional to their fitness. A survivor is chosen based on which pocket the ball (in this case a random number) falls into.

**Breeding:** Breeding involves creating a new individual through a *crossover* or *mutation* operator. Random mutation is used, where a survivor is copied except for one gene. This gene is mutated by randomly selected a new value. Random Crossover involves constructing an individual from two parents, where each gene is copied from one of the two parents.

**Termination criteria:** The number of unique solver runs are used as a termination criteria as running the solver is the computationally most expensive component of the algorithm.

#### 7.1.2 Minimal Runtime GA

The Minimum Runtime Genetic Algorithm (MRGA) uses an evaluation function that assigns high fitness values to individuals in the population that result in a low runtime for the solver. As a result, the population evolves towards individuals which apply a set of solver parameters that achieve a low runtime when applied to a particular model. To ensure that the algorithm does not simply prefer less complex model configurations, we normalize the runtime on the maximum runtime of the individuals k nearest neighbors ($k$ = 5). The evaluation function $F_{MRGA}$ is shown in equation 8, where $N$ is the set of $k$ nearest individuals to the individual represented by model configuration $m$ and solver parameters $s$.

$$F_{MRGA}(m,s) = \frac{f(m,s)}{\max f(m_k,s_k)} : z_k \in N \qquad (8)$$

The distance calculation used to determine the nearest neighbors is the number if genes that differ. For example, individuals $\{1,2,3,4,5\}$ and$\{1,4,3,2,5\}$ have a distance of 2. The assumption is that individuals with similar model and solver configurations will result in a similar runtime.

#### 7.1.3 Novelty GA

Another GA is used, but using an evaluation function based on the novelty of a solution. Novelty is described as how different an individual behaves to the rest of the population. In our case, it considers how much the runtime differs to other neighboring individuals. For this we calculate the sum of differences in their runtime between the individual and its nearest neighbors. The novelty evaluation function $F_{Novelty}$ is shown in equation 9.

$$F_{Novelty}(m,s) = \sum_{k=5}^{5} f(m_k,s_k) - f(m_i,s_i) : z_k \in N \quad (9)$$

This means it will also favor parameter settings which result in a high runtime, if it represents an aspect that has not been covered by other individuals. As a result, this algorithm will produce a population that represents all rare points in the search space.

#### 7.1.4 Random Algorithm

Random training Data Generation is most commonly used in literature (Hutter et al., 2014; Kotthoff et al., 2012), as it is the simplest to implement. We create an algorithm, based on a GA, that adds new random individuals in each generation to provide a comparison for varying training data set sizes.

### 7.2 ML Strategies

Given a set of solver parameters and a model instance, the predictor provides an estimate of the runtime without having to run the computationally expensive CPLEX solver. The predictor is built using one of the following ML strategies (Hutter et al., 2014; Eibe et al., 2016): Artificial Neural Networks (ANN), Ridge Regression (RR), Gaussian Process Regression (GPR), Random Tree(RT) and Random Forrest (RF). We chose this set of ML strategies as they are commonly used in ML and are also explored in (Hutter et al., 2014), but theoretically any ML method could be used. We use the implementations from the commonly used WEKA (Eibe et al., 2016) ML library and use WEKA default parameters for

each method. However, as ANNs require inputs to define their structure, we chose the parameters that performed generally well in a small pre-study. The structure applied consists of one hidden layer with as many hidden neurons as input neurons. We use a learning rate of 0.01, momentum of 0.1 and 2000 training iterations. The predictor gives the runtime as output. For inputs of the predictor, we use each solver parameters combined with a set of complexity measures. Complexity measures gives an indication of the size and structure of the mathematical model after it has been configured according to an individuals model description. We do not use the model configurations directly as input, as this would limit the predictor, allowing it to only predict runtime for combinations of already known configurations. Using the complexity measures allows it to generalize. The following complexity measrues are uses: The number of variables, functions, Non-zero functions and binary variables. These are obtained from the model statistics of the CPLEX solver.

## 8 EXPERIMENT SETUP

Our experiments analyze the combination of all Data Generation and ML strategies. For each combination, we analyze its predictors performance for different sizes of training instances. Therefore, for every 100 instances selected by the Data Generation strategy we train a predictor. We can then observe the predictors performance as the training sets size increases. We use a randomly selected set of model instances as test sets. To make comparisons between methods without being affected by this random selection, as well as each algorithms non-deterministic nature, we repeat each experiment 150 times and present here median values. Runtime is measured in CPLEX ticks, which is a stable measurement of how much computation CPLEX performs to produce a result. We use this measurement as it is independent of any other processes, allowing us to reliably run several models in parallel. We apply a maximum threshold of 4,000,000 CPLEX ticks for each run. We also measure the size of the training sets used for producing the predictor, as this indicates the number of models needed to be solved by CPLEX and represents the computationally most expensive process in the training phase. The experiments are run on a dedicated Linux server, using GAMS (GAM, 2017) as the modeling language and our own implementation of the experiment setup and Data Generation strategies in Java. We create two experiments, the first focusing on using the Machine Learning methods for *Runtime Prediction* and the second for *Runtime Optimization*. To measure performance for the purpose of Runtime Prediction we calculate the absolute difference between the predicted and actual runtime for each ML method. We select two model instances and all possible solver parameters for those instances as test data. This results in 800 test instances for each repeat. All strategies are tested using the same test data in each repeat. We make observations until a size of 2000 training instances, creating a predictor every 100 model solves. In total, for 150 repeats, 20 predictors per repeat and 800 test instances per predictor we make 2'400'000 observations per method. For the Runtime Optimization it is important to observe the runtime when using the best solver parameters suggested by the predictor. We compare the results between the different sets of ML and training Data Generation strategies, as well as the default settings of the CPLEX solver. Again, we select two model configurations as test instances. However, as the predictor is used to select the best solver parameters, this results in only two test instances. However, due to 150 repetitions and 20 individually trained predictors per repeat, we observe 6'000 data points per method. We also wish to measure the best possible runtime as the maximum potential possible of any method. However, this is only possible by a brute force approach and is not practical. However, as we repeat each experiment many times for scientific certainty, we almost do so regardless. Therefore, we verify that we have searched the space completely with an exhaustive search and include the best possible times in our results for comparison.

## 9 RESULTS

In our results we expect a well performing predictor to have an overall low prediction error when used for Runtime Prediction (Experiment 1) as shown in Figure 3. We also expect to see a low variance in these prediction errors and a stable learning curve. Furthermore, when these predictors are used for Runtime Optimization (Experiment 2) as shown in 2, we expect to see an overall low runtime. Overall we see that the Data Generation method has little effect on the prediction accuracy and only nominal effects on the error variance. However, we see that the learning method has a impact on the prediction errors. RFs are the best method for Runtime Prediction. More observations are described below. In Figure 3 we see that performances in Runtime Optimization do not reflect the performances for Runtime Prediction and therefore should be treated as a separate problem. We see

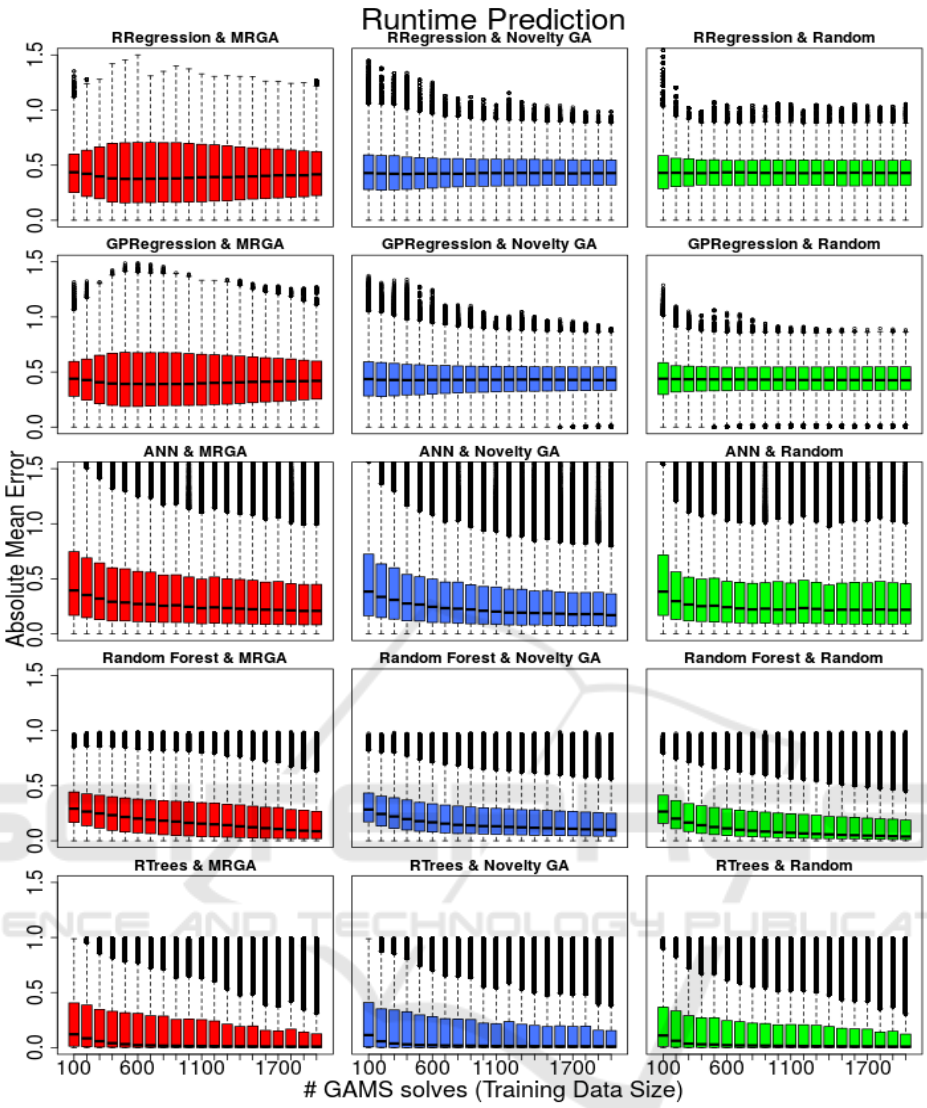


Figure 2: Graphic showing the error in Runtime Predictions.

that that the Data Generation and Machine Learning method has a large effect on the predictors performance. RF demonstrates the best and stable performance and is ideally combined with strategy that emphasizes performance, such as MRGA. Using such a strategy, the performance of the CPLEX solver can be increase by around 40% when compared to default parameters. However, potential exists for a much large performance boost and that the minimum runtime GA is not the best performing Data Generation method for all Machine Learning methods. The state of the art currently uses ANN with a random Data Generation strategy. We use this as a baseline to compare our results. For the Runtime Optimization, we also include the industry standard as a second baseline, which consists of using the default settings of the solver. For Runtime Predictions, the potential best is a 0% error with no variance. The errors are normalized by a maximum runtime threshold, which is applied to the solver. Therefore, as long as the predictor learns this threshold, we would not expect an error higher than 100%. The potential best for the Runtime Optimization can only be determined by solving the search space completely. The potential maximum is the threshold applied to the solver. The fact that Data Generation strategies have a larger effect for Runtime Optimization than prediction can be explained by the high variance in the prediction errors. The outliers in the predictor errors indicate that there is a small subset in individuals that are difficult to predict. They are a result of an imbalanced search space due to specialized parameters sets. Specialized parameter sets

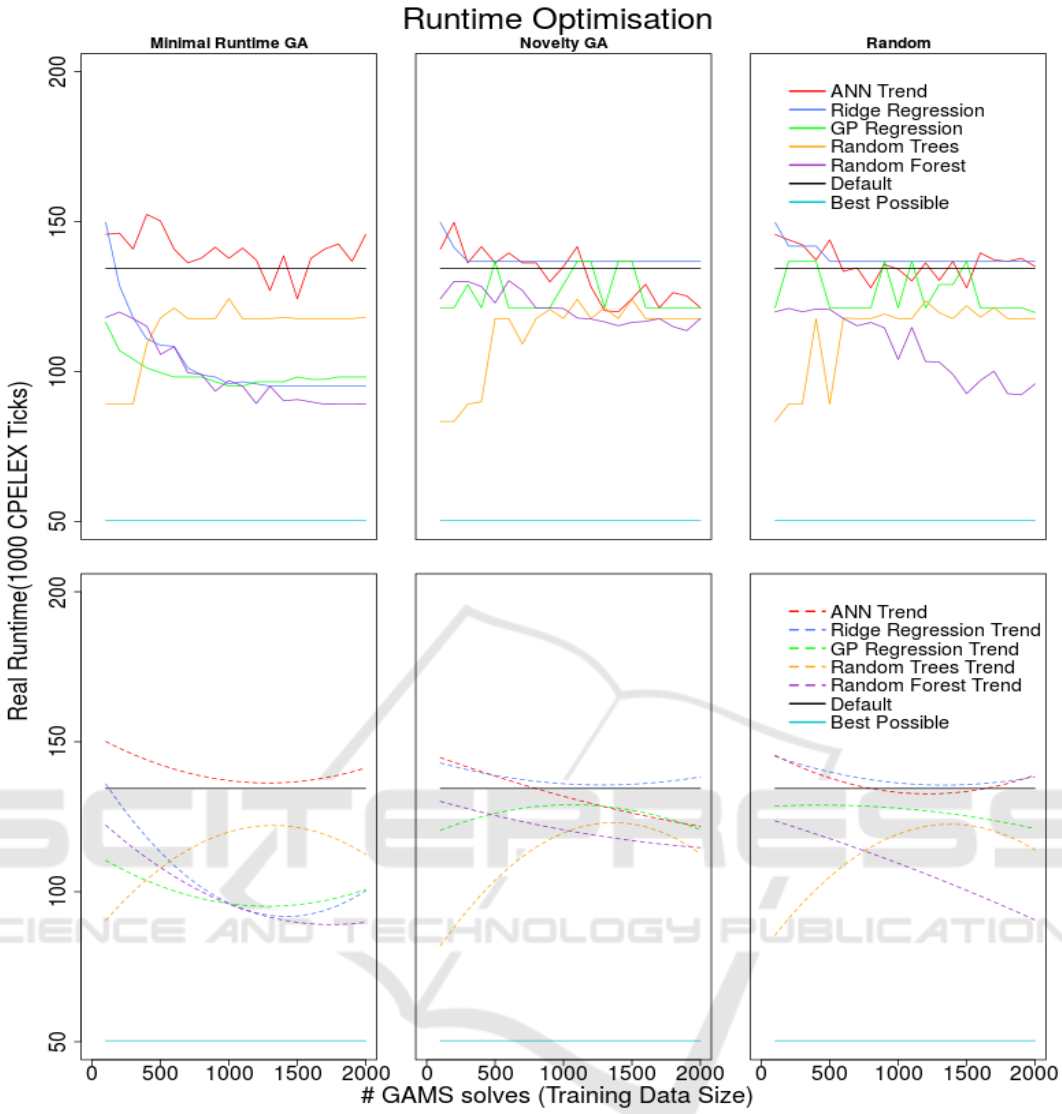


Figure 3: Graphic showing the runtime resulting from using a predictor to suggest solver parameters.

achieve a low runtime for specific models, but not for others. As such, specialized parameter settings can produce a low runtime when used for the correct models and are key for Runtime Optimization. However, they are also more difficult to predict. Therefore, a predictors ability to perform in Runtime Optimization is limited by the understanding of specialized parameter sets. As such, Data Generation such as MRGA that search for the specialized parameter sets will generally perform best for Runtime Optimization. The effect of the unbalanced search space is also noticeable for the Machine Learning methods. For example, RF are resistant to imbalanced training data and in addition to having a generally low prediction error. In comparison, the regression methods are affected by the imbalance except when a MRGA Data Generation method is used. The trade off between the training set size and performance is clearly shown in our results. Regression methods, when using MRGA, outperform RF when only small data sets are available. Decision trees in particular perform well, before suffering from over fitting. When novelty search is used for the purpose of Runtime Optimization, we can observe a convergence of performances. Novelty search emphasis all rare sets of individuals in the search space and therefore gives a more equal representation of all such rare sets. This shows that the Machine Learning performance is only determined by its ability to handle imbalanced data. The threshold applied to the CPLEX can be seen in Figure 2 as the the random forest and decision tree learn that no estimation should exceed the threshold. However, other methods esti-

mate runtime that exceed the threshold.

## 10 CONCLUSION

We have successfully tested several Data Generation strategies and ML strategies for predicting and optimizing the runtime of mathematical solvers. In summary, we can draw the following conclusions from our study. Runtime Prediction requires robust Machine Learning strategies, that are not effect by unbalanced data. Therefore, RF, RT and ANN are viable methods, but RF seems to be best suited. Data generation methods have almost no effect on a predictors accuracy. Predictors that perform well in predicting the runtime of a solver does not necessarily result in the best performing predictor for Runtime Optimization. The performance of a predictor used for Runtime Optimization depends on the Machine Learning method used as well as the Data Generation strategy. In our experiments RF performs best for all Data Generation strategies, but it is best combined with a MRGA. Other ML methods are strongly dependent on the Data Generation strategy. Regression methods are only effective if the unbalance in the training data is countered with strategies similar to the MRGA. ANN benefit best from a Novelty Strategy. Random selection strategies are discouraged, but can be used in combination with RF. For small data sizes, GPR with MRGA is recommended. Overall, a performance increase of around 40% can be achieved when compared to the default parameters. However, the maximum potential for such methods is around 70%.

## REFERENCES

(2017). GAMS/CPLEX 10 solver manual.

Barry, M., Abgottspon, H., and Schumann, R. (2018). Solver tuning and model configuration. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 141–154. Springer.

Barry, M., Schillinger, M., Weigt, H., and Schumann, R. (2015). Configuration of hydro power plant mathematical models. In *Energy Informatics: Proceedings of the Energieinformatik 2015*, volume 9424 of *Lecture Notes in Computer Sciences*. Springer.

Barry, M. and Schumann, R. (2015). Dynamic and configurable mathematical modelling of a hydropower plant research in progress paper. In *Presented at the 29. Workshop "Planen, Scheduling und Konfigurieren, Entwerfen" (PuK 2015)*.

Chawdhry, P. K., Roy, R., and Pant, R. K. (2012). *Soft computing in engineering design and manufacturing*. Springer Science & Business Media.

Eibe, F., Hall, M., Witten, I., and Pal, J. (2016). The weka workbench. *Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques*, 4.

Ertekin, S., Huang, J., Bottou, L., and Giles, L. (2007). Learning on the border: active learning in imbalanced data classification. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 127–136. ACM.

He, H., Bai, Y., Garcia, E. A., and Li, S. (2008). Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 1322–1328. IEEE.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, page 190.

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

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

IBM (2016). CPLEX Performance Tuning for Mixed Integer Programs.

Kotthoff, L., Gent, I. P., and Miguel, I. (2012). An evaluation of machine learning in algorithm selection for search problems. *AI Communications*, 25(3):257–270.

López-Ibánez, M. and Stützle, T. (2014). Automatically improving the anytime behaviour of optimisation algorithms. *European Journal of Operational Research*, 235(3):569–582.

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