Universiteit Leiden
Opleiding Informatica

A Metalearning Approach to the

Chip Production Scheduling Problem

Name: E. Partodikromo

Date: 15/08/2017

1st supervisor: Prof.Dr. J.Kok
2nd supervisor: Dr. C.Soares

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

The Chip Production Scheduling Problem concerns finding an optimal scheduling solution of job-assignments to a set of machines that reduce the total production time, called the makespan. We acquire the solution by using a Genetic Algorithm, but the quality depends on the values chosen for the parameters of the Genetic Algorithm. Given the large search-space of possible parameter values makes this a challenging problem. This would amount to testing a large number of algorithms. Therefore, in this research, we address the problem of selecting one of two Genetic Algorithm variants to the problem of Chip Production Scheduling Problem using metalearning.

The metalearning approach is based on simple and landmarker meta-features. We thereby carry out two metalearning tasks which are, 1) predicting the makespan of each meta-heuristic variant and 2) selecting the best meta-heuristic for a given instance. These represent a regression and classification task respectively. The classification task is furthermore split into a straightforward classification and classification via regression variant. We use different methods to evaluate the performance of both tasks. For regression, it is the RMSE (Root Mean Squared Error) and for classification prediction accuracy. We train five algorithms (Random Forest, k-Nearest Neighbour, Linear Model, Multilayer Perceptron, Decision Tree) with their predefined setup on a reduced set of meta-data with the Leave-One-Out method to learn meta-models, and ran the best ones on the test data.

What appears from the preliminary findings is that the metalearning approach we designed obtained promising results for recommending i.e. predicting the best Genetic Algorithm for the instances of the Chip Production Scheduling Problem.

Acknowledgements

My gratitude goes to Professor Carlos Soares, Luís Guardão, Pedro Abreu and the research-team at the Centro de Engenharia de Sistemas Empresariais (CESE) INESC-TEC Porto, for their support of this project. I also would like to thank Dylan for his time and constructive feedback on this thesis.

Contents

Introduction

Large factory-plants deal with complex planning schedules of *production-orders* every day. A production-order, better known as a *job* is a process-flow of machine-operations which fabricate a product. These plants typically execute a high number of multiple jobs, which in the case of inefficient planning leads to a lot of costs. Subsequently, the aim for these plants is to obtain high-turnout and profit, while minimizing production expenses. An important expense that needs to be reduced is the production time or *makespan*. Essentially, the makespan worsens when multiple jobs need to access the same machine and when machines run idle. Therefore it is necessary to find a scheduling assignment of jobs that minimize these problems which lead to a reduced makespan. In operations-research and machine-learning this problem is known as the *Job-Shop Scheduling Problem*. In this thesis, the aim is to solve an instance called the Chip Production Scheduling Problem. This instance includes jobs that produce different chip and electronic components.

Various approaches [1–3] using heuristic algorithms to solve Job Shop Scheduling Problem instances have been implemented with good performance results. The scheduling solutions found by these approaches were subsequently the best ones for their respective instances. Given these results, we decide to implement a heuristic algorithm to find a good scheduling solution for the Chip Production Scheduling Problem. We thereby choose the Genetic Algorithm, as the heuristic algorithm.

To find a good solution we need to set the values for the parameters of the Genetic Algorithm. After all, these have a big impact on the performance of the algorithm. The search space of possible parameter values is unfortunately too large to explore with limited resources and time. Therefore we will only investigate two sets of parameter values

of the algorithm. In other words, this means we investigate two Genetic Algorithms with distinct parameter values. Nevertheless, we need to decide on the best algorithm for each Chip Production Scheduling Problem instance because they find different solutions.

One approach to this problem is using machine learning i.e. data-mining on historical data. More specifically, by applying data-mining techniques to the problem of selecting two different Genetic Algorithms. We are in particular interested in metalearning because of its successful application to a diverse range of algorithm-selection instances.

In this research, we are addressing the problem of selecting one of two Genetic Algorithm variants to the problem of Chip Production Scheduling Problem using metalearning. The main contribution of this thesis is the design of a metalearning approach for the Job Shop Scheduling Problem instance, the Chip Production Scheduling Problem.

1.1 Thesis Overview

In chapter 2 we define and formalize the Job Shop Scheduling Problem instance of this research, describe the optimization algorithm and metalearning.

In chapter 3 we discuss the metalearning solution to the Job Shop Scheduling Problem instance. Chapter 4 relates the results of the experiments.

Last, we summarize the findings and conclusions and pose suggestions for future work in chapter 5.

Background

This research focuses on developing an integrated approach of a meta-heuristic and metalearning to solve a Job Shop Scheduling Problem approach utilized for algorithm selection in more detail.

2.1 Other Approaches

Researchers implemented heuristic approaches to solve various Job Shop Scheduling Problem instances. These include Ant Systems [4], Ant Colony Optimization [1], heuristic backtracking [5] and modified Filtered Beam Search [6]. Since these methods were widely used and applied with success to Job Shop Scheduling Problem instances, we decided as well to implement a heuristic algorithm to solve the Job Shop Scheduling Problem. Therefore, we chose to implement a standard Genetic Algorithm (GA).

The good results obtained by the methods are consequences of selecting parameter values which guarantee the best performance on the problem. The Genetic Algorithm has four parameters, namely the mutation probability p_m, crossover probability p_m, population size μ and the number of iterations n. The performance of the algorithm is in particular affected by the crossover and mutation probabilities. For instance, a too high mutation probability leads to the algorithm finding random solutions and not converging to an optimal solution. On the other hand, a small mutation probability may lead to fast convergence. As for the crossover probability, too high values might cause the algorithm to converge too fast, thus skipping potential better solutions. A basic approach to finding good a good parameter values is Grid-Search, which simply searches all possible values for each parameter. However, the Genetic Algorithm has a

large search space of possible parameter values and given the limited amount of resources and time, Grid Search would be too costly and inefficient. This amounts to trying out a large number of algorithms. So, in order for the Genetic Algorithm to find the best solution to the Job Shop Scheduling Problem instance, we have to 1) select a Genetic Algorithm which provides an optimal solution and 2) improve the search efficiency. We thereby limit the number of algorithms to two, but still, have to make an informed decision about the best one for a given Job Shop Scheduling Problem instance. After all, the Job Shop Scheduling Problem contains different instances. To realize this, we will use *metalearning*.

2.2 Genetic Algorithm

Genetic Algorithms are heuristic search algorithms based on principles of the natural evolution theory and genetics [7]. Genetic Algorithms are able to search for potential solutions in large search spaces quickly while exhaustive or brute-force search methods cannot. Thus, these algorithms are often employed in many practical optimization problems with large solution spaces. A Genetic Algorithm consists of 3 main operations: selection, crossover, and mutation. It evolves a selection of candidate solutions into better solutions by means of its operations, until a stopping criterion is verified. Algorithm 1 illustrates the concept: An initial population $P(t)$ of encoded candidate solutions are

Algorithm 1 Genetic Algorithm

```
 1: t := 0;
 2: initialize P(t);
 3: F(t)' := evaluate P(t);
 4: while not terminate do
 5:     P'(t) := select(F(t));
 6:     P"(t) := crossover(P'(t),p_c);
 7:     P"'(t) := mutate(P"(t),p_m);
 8:     F(t) := evaluate(P"'(t));
 9:     P(t+1) := P"'(t);
10:     t := t + 1;
11: end while
```

created for the Genetic Algorithm. Next, an evaluation function assigns fitness scores to each solution. The higher the value of the score, the better its corresponding candidate is. For a given number of iterations, the Genetic Algorithm generates new individuals from the initial population with its selection, crossover and mutation operators. The fitness scores of the new population are then evaluated once again. The evaluation of the new population continues into the next round of transformations and so forth. What

follows is a brief description of the common implementations of Genetic Algorithms operators.

Selection

The selection method chooses candidate solutions from the set of solutions $P(t)$ in the current iteration. We use elitist selection, which lets a small number of solutions with the best fitness values pass to the next generation, avoiding the crossover and mutation operators. This prevents the random destruction of good solutions.

Crossover

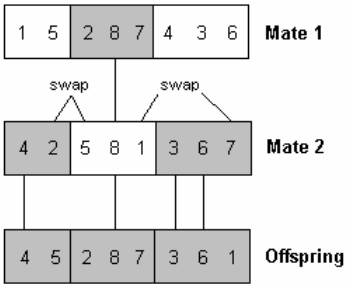

FIGURE 2.1: PMX crossover example, derived from [8].

Crossover is applied with probability p_c to obtain new solutions. This course eventually leads to better solutions. Per two parent solutions, a partially mapped crossover (pmx) operation is performed. This method extracts a part of one parent for a new solution. The corresponding part of the new solution filled in. The same process is repeated by switching the two parent solutions. In the end, two new solutions are combined from the parent solutions.

Mutation

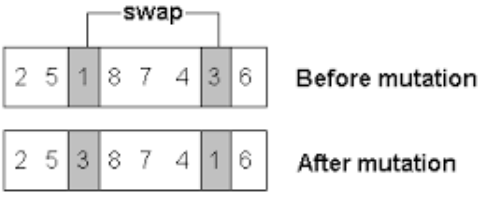

FIGURE 2.2: Swap mutation example, derived from [9].

Mutation is performed on the solutions $P''(t)$ generated by the crossover operator, to generate solutions which are more diverse. In the swap mutation, each bit has a mutation probability p_m of being inverted. Finally, the mutated solutions $P'''(t)$ will become the new candidate solutions $P(t+1)$ for the next iteration.

2.3 Semiconductor Chip Production Scheduling Problem

The essential chip components that guarantee the working functionality of electronic products are manufactured by semiconductor wafer fabrication facilities. Process-flows, known as *jobs* produce the components. These jobs have to be scheduled appropriately to reduce the total makespan. The total makespan refers to the processing time of all jobs for a set of orders to produce many components. This describes a classic Job Shop Scheduling Problem with a single optimization objective, that is defined as follows:

Let a set $\mathcal{R}$ be m jobs J : $\{j_1, j_2, ...j_m\}$ with a corresponding set of n machines M: $\{m_1, m_2, ..., m_n\}$. Each job consists of a set operations O: $\{O_1^j, O_2^j, ...O_n^j\}$ to fabricate its component by utilizing the machines with execution times $d(O_k^j)$ in time units. Furthermore, the number of N_{ij} operations for all jobs is different. Within each job, various operations can also use the same machine. Subsequently, the objective is to plan a schedule s of job assignments to M such that the makespan of producing $\mathcal{R}$ is minimized: $\mathbf{min}(\mathbf{makeSpan}(\mathbf{s}))$. An example of a scheduling assignment is displayed by the following figure:

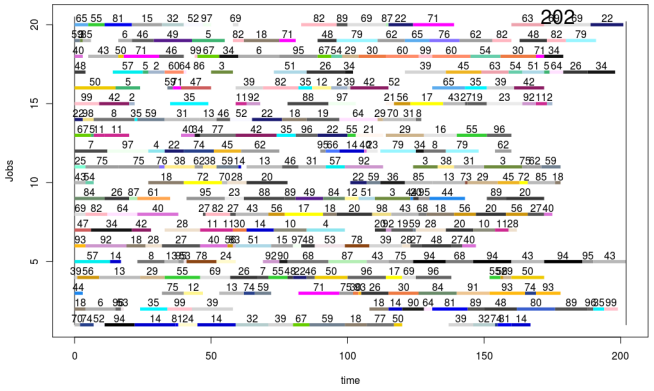

FIGURE 2.3: Scheduling assignment of a set of 20 jobs. The job-operations (colored) are arranged such that a minimum makespan (total production time) of 202 minutes (visible right-above) is acquired.

The following section explains the Genetic Algorithm approach to this problem.

Genetic Algorithm Approach to Chip Production Scheduling Problem

The representation of the candidate solutions, the choice of the fitness function and the maximization or minimization objective(s) need to be set for the Chip Production Scheduling Problem. As explained before, the input for the genetic algorithm population $P(t)$ consists of a set of randomly generated candidate solutions that are evolved over the course of many evaluations. Given that this is the approach followed in this project, we describe it in more detail. The input data of the problem instances are converted into different priority lists, i.e. $n \times k$ matrices having n rows of machines with the corresponding list of at most k jobs assignments. An example of the input is shown in table 2.1.

TABLE 2.1: Example of input representation of the Chip Production Scheduling Problem data for the Genetic Algorithm .

Machines	*Jobs*		
m_1	job_A	job_B	job_C
m_2	job_B	job_C	job_B
m_3	job_A	job_C	

Hence, the crossover and mutation operations described earlier can be performed on these lists. Since in this case, a job can have multiple operations n_i on the same machine, k varies for each row. The fitness values for this problem are the makespan values associated with the execution of a schedule s. These schedules are obtained from the priority lists using the Giffler-Thompson algorithm [10]. At the same time, the algorithm returns the calculated makespan. Thus, for the aforementioned problem, the fitness value is the makespan of the schedule obtained with the Giffler-Thompson algorithm.

2.4 Metalearning for Algorithm Selection

The research field of metalearning recently emerged. It successfully addressed different Machine Learning tasks and challenges. In general, it concerns research that aims to

learn from past knowledge of experiments with machine learning processes. For clarification, a more precise definition by Brazdil et al. [11] can be considered: "The field of Machine Learning exploits the relation between tasks or domains and learning algorithms". Simply put, the core purpose here is to obtain insight into which algorithm and corresponding set of parameter values are suitable for some given data set. This problem can be looked at as the *algorithm selection problem*, which is explained in the next section.

2.4.1 Algorithm Selection Problem

The Algorithm Selection Problem was introduced by Rice's framework [12]:

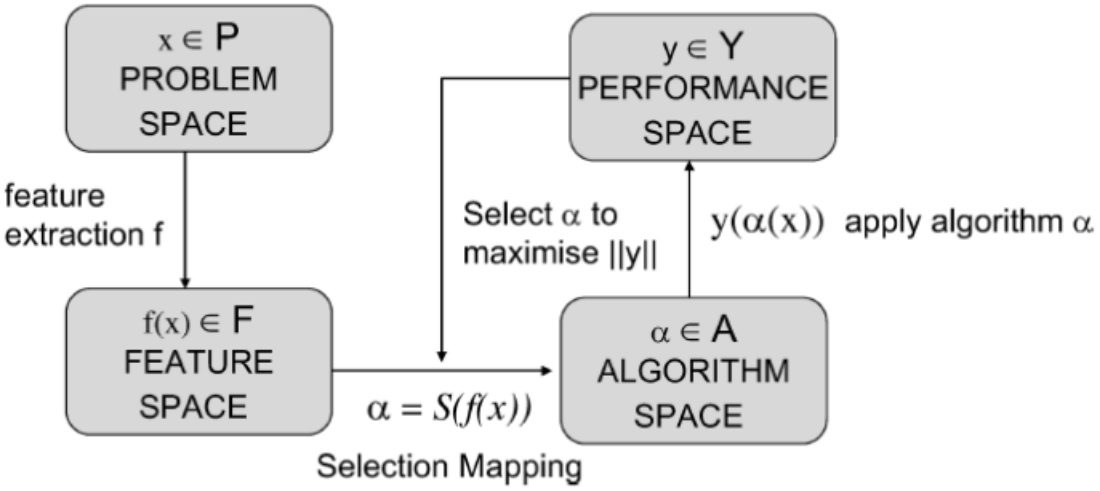

FIGURE 2.4: Rice's Algorithm Selection framework, taken from [13]

The problem space $\mathcal{P}$ that represents the set of instances of a problem class. The feature space $\mathcal{F}$ contains measurable characteristics of the instances generated by a computational feature extraction process applied to $\mathcal{P}$. Next, the set $\mathcal{A}$ consists of all algorithms relevant to solving the problem. Finally, the performance space $\mathcal{Y}$ represents the mapping of each algorithm to a set of performance metrics. The problem is subsequently defined as follows:

> *" For a given problem instance $x \in \mathcal{P}$, with features $f(x) \in \mathcal{F}$ find the selection mapping $S(f(x))$ into algorithm space, such that the selected algorithm $a \in \mathcal{A}$ maximizes the performance mapping $y(a(x)) \in \mathcal{Y}$ "*

Essentially, the definition relates to finding an algorithm with maximal performance on a problem instance described by certain features. This is difficult because there are usually various algorithms available for solving a specific problem. Moreover, the No Free Lunch Theorem [14] states that "(...) for any algorithm, any elevated performance over one class of problems is offset by performance over another class". In other words, this means that one algorithm will perform well on particular data sets, but poorly on others. Another important issue of the algorithm selection problem is the selection of appropriate features. The framework illustrated in figure 2.3 provides an abstraction of the approach to the algorithm selection problem and also serves as the backbone of the metalearning procedure explained next.

Metalearning first collects meta-data from a set of problem instances and a set of candidate learning algorithms. The meta-data consists of *meta-features* and *meta-targets*. The meta-features are characteristics of the problem extracted from the data. The meta-targets contain information about the performance of the algorithms on the data sets, also known as *base-level* performance data. Afterward, a meta-model is generated from the obtained meta-data. Figure 2.4 illustrates this concept:

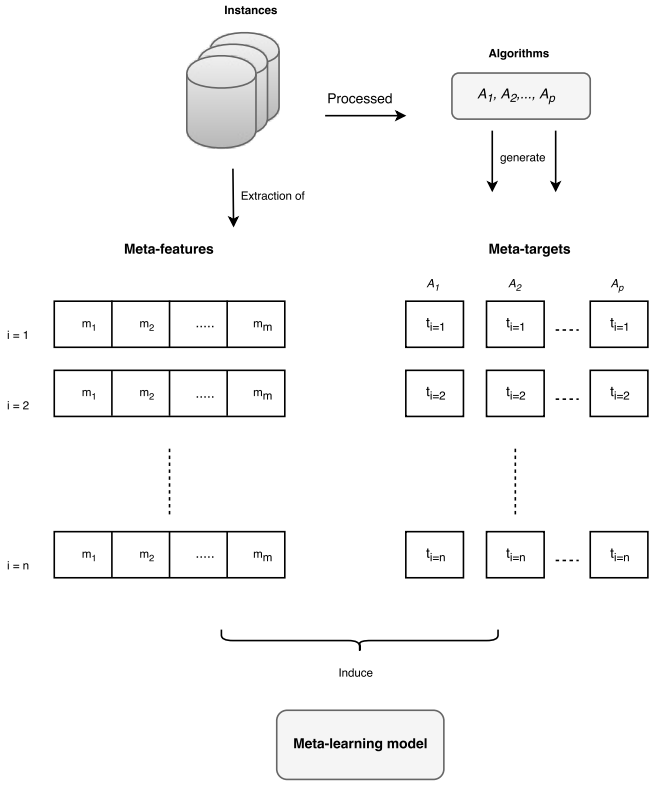

FIGURE 2.5: Metalearning framework

Consequently, the model can make predictions concerning the relative performance of the candidate learning algorithms on new instances.

2.4.2 Meta-features

Meta-features are characteristics of the problem extracted from the data. We can divide these into 3 main classes. The first class contains *simple, statistical information* features. These meta-features are estimated from the data set. They include, for example, variance, kurtosis, the number of classes and the median.

The second class of meta features is *model-based* features. They are based on a simple learning model and then some characteristics of that are measured and stored. If for instance, the meta-model is a simple decision tree one can measure the number of leaves and the number of nodes.

The final class consists of *landmarker* features. Landmarkers are fast estimates of algorithm performances. A simplified version of the algorithm runs which collects various performance measures. These measures provide richer data-characterization than simple, statistical features which merely describe the problem instance. By directly measuring various performance data, landmarkers can give a good indication of the overall performance of the algorithm on the problem instance.

2.5 Summary

In this chapter, we introduced the Chip Production Scheduling Problem (Chip Production Scheduling Problem), the Genetic Algorithm and metalearning for algorithm selection. The goal of the Chip Production Scheduling Problem is to reduce the total processing time, makespan of the jobs to be produced. To achieve this, a meta-heuristic such as a Genetic Algorithm searches for the corresponding s. To ensure finding the best scheduling solution, certain parameters of the Genetic Algorithm need to be tuned. More precisely, we need to find a good set of parameter values for the algorithm.

Approach

In this chapter, we explain the metalearning approach to the Chip Production Scheduling Problem. We adapt the algorithm selection problem and the metalearning approach to the Chip Production Scheduling Problem as follows. We have a set $\mathcal{P}$ of the problem instances from the Chip Production Scheduling Problem. These instances have a feature space $\mathcal{F}$, that consists of the computed meta-features. Next, we have a set $\mathcal{A}$ of two Genetic Algorithms. For each algorithm we compute the meta-targets for the instances. The meta-features and meta-targets together form the meta-data. Next, we train a meta-model with the meta-data for regression and classification. We then apply the model to new instances and review its performance by looking at the regression and classification predictions for the new instances. The better the performance of the model, the more likely the predicted, i.e. recommended algorithm (in this case one of the two Genetic Algorithms) is indeed the best choice for a new instance.

3.1 Metalearning tasks

The two Genetic Algorithms we are implementing, are derived from the standard Genetic Algorithm parameter setup of deJong [15] and from Grefenstette [16]. Here, we define a parameter setup as a set of values assigned to the parameters of the Genetic Algorithm. The choice of metalearning models is a simple Decision Tree (DT), Multilayer Perceptron (MLP), k-Nearest Neighbour (NN), Random Forest (RF), Linear Model (LM). The following sections give an overview of the models and elaborate in more detail the regression and classification tasks.

3.1.1 Models

- A **Decision Tree** builds regression or classification models in the form a tree structure. The tree consists of decision nodes and leave nodes, where the decision nodes have two or more branches and the leave nodes represent a prediction or decision.

- A **Random Forest** [17] is an ensemble learning method for classification and regression problems. It consists of an 'ensemble of weak learners', i.e. group of tree predictors. It returns the mode of all predictions from the decision trees.

- The ***k*-Nearest Neighbour** [18] algorithm is a method used for classification and regression tasks. In the case of classification, an entity is classified by a majority vote of its k nearest neighbours with the class that is common among these neighbours. With regression, the entity gets assigned the average of the output of its k nearest neighbours.

- A **Multilayer Perceptron** [19] is a supervised learning algorithm that learns a function by mapping a set of input data onto a set of suitable outputs, with training data. Given a set of input features and target values, it can approximate a nonlinear function for either classification or regression tasks. For classification, the target values can consist of binary or multiple classes. For regression, the target values are numeric predictions.

- A **Linear Model** is a method that computes a linear relationship between the input variables and a single output variable and is used for regression and classification predictions. For a regression task, it predicts a numeric output value. Because classification tasks have class targets, the Linear Model uses a logistic function to model the output into target classes.

3.1.2 Regression

For the regression task, the target-value y_r is equal to the gain between the best fitness values (makespans) r^{opt} of the start-population and f^{opt} generated by the best solution

returned by the Genetic Algorithm:

$$y_r = \frac{r^{opt} - f^{opt}}{r^{opt}} \tag{3.1}$$

We calculate two variants of y_r, y_{rdJ} and y_{rG}. The y_{rdJ} feature is computed for the deJong Genetic Algorithm, the y_{rG} for the Grefenstette Genetic Algorithm. These are the ground-truth target values for the model. Next, the recommended meta-targets $y_{recommend}$ are acquired as follows. Given the training metadata, a meta-model is learned for each regression task. The meta-model is applied to the test examples to obtain a prediction of the gain obtained by the corresponding Genetic Algorithm on the given instance. RMSE is used to evaluate the accuracy of the meta-model, by comparing the predicted to the true values. Afterwards, we evaluate the performance of the model with a baseline method. For the **baseline**, we calculate the **ZeroRule** as follows for both Genetic Algorithms:

$$r^i = (y_r^i - \overline{Y})^2 \tag{3.2}$$

$$ZeroRule = \frac{(\sum_{i=1}^{n} r^i)}{n} \tag{3.3}$$

3.1.3 Classification

For the classification task, we need to compute a different target attribute. The target y_{cl} for the true prediction is determined between the best fitness value f^{opt} that is returned by either the first or the second Genetic Algorithm (deJong and Grefenstette), which is represented by the following equation:

$$\textbf{if } f_{deJong}^{opt} < f_{Gref.}^{opt} \textbf{ then } 1 \textbf{ else } 0 \tag{3.4}$$

The **baseline** we use in the case of classification is the **Majority Class**. That is the most represented class of the true predictions.

We use two approaches to obtaining the classification. The first is using a straightforward classification approach, i.e. learning a model by applying a classification algorithm to the data with the classification target. We refer to the predicted classes following this

approach as $y_{cl-normal}$. We also use the predictions of the regression models to obtain the predicted class $y_{cl-regression}$. The $y_{cl-regression}$ and $y_{cl-normal}$ targets are afterwards evaluated by comparing them to the observed target value y_{cl} with a simple accuracy measure. This measure calculates the percentage p of correctly predicted targets. First, the number of correctly predicted target values $targ_{corr}$ are obtained by comparing the predicted target values with the observed target values. Afterward, we obtain p by dividing $targ_{corr}$ by the total number of predicted target values. We compare the estimated classification accuracies with the baseline, which is the **Majority Class**. Next, we describe the sets of meta-features considered in this work.

3.2 Meta-features Chip Production Scheduling Problem

The first set of meta-features concerns the features that are based on simple measures extracted from the instance. The second set consists of the experimental landmarker features based on Genetic Algorithm properties, which includes proposed and adapted features from the state-of-the-art research by Kanda et al.[20]. The features based on the properties and search-space of the Genetic Algorithm are defined according to the following notation:

- S_i: $\left\{s_i^1, s_i^2, \ldots, s_i^w\right\}$, the set of candidate solutions of the Genetic Algorithm for an instance i.
- R_i: $\left\{r_i^1, r_i^2, \ldots, r_i^y\right\}$, a set of randomly generated solutions where $R_i \subset S_i$.
- N_i: $\left\{n_i^{j,1}, n_i^{j,2}, \ldots, n_i^{j,v}\right\}$, the set of neighbour solutions of a solution s_i^j with 1 mutation where the best neighbour solution $n_i^{j,best}$ is the one with the lowest (best) fitness value (makespan).

The Appendix A includes the mathematical expressions of the features.

3.2.1 Simple meta-features

The simple meta-features are calculated directly from the instance (data set). They are listed in table 3.1.

TABLE 3.1: Simple meta-features .

Meta-feature	**Description**
avgM	Average number of machines
avgT	Average time-duration of the operations
HT	Highest time-duration
LT	Lowest time-duration
varT	Time-duration variance
kurT	Time-duration kurtosis

The first meta-feature *avgM* computes the average number of machines per instance. It does this by summing up the total number of machines from all jobs listed and averaging them by the job-count. The second feature *avgT* first calculates the average amount of time per job, then sums these up and adopts the mean over the total number of jobs of the instance. *HT* and *LT* respectively seek out the highest and lowest time-duration from all jobs of the instance. *varT* and *kurT* compute the time-duration variance and kurtosis per job and sums them up. Consequently, they are averaged over all jobs of the instance.

We think it is good to include these meta-features, because simple information-based features have been used extensively in metalearning. Thus, they form the minimal basis of data-characterization for the Chip Production Scheduling Problem instances. Subsequently, we can expand this basis with landmarking features, explained in the next section.

3.2.2 Landmarker features

The landmarker features are divided into two groups: the proposed and adapted features, which are shown in the next table.

TABLE 3.2: Proposed and adapted landmarkers from the state-of-the-art .

Meta-feature	**Description**
Proposed	
avgF	Average makespan i.e. fitness value
avgR	Estimated average from random solutions
maxF	Maximum solution value of the search-space
minF	Minimum solution value of the search-space
Adapted	
EPN	Expected proportion of neighbours with a better solution
ERN	Expected ratio between fitness of best neighbour solution and the generated solution
QBO	Quality of best offspring solution
RFP	Average ratio between fitness of best offspring solution and best solution of their parent
AOP	Average number of times offspring is better than parents

These features are based on properties of the Genetic Algorithm and measure different relations between various solutions found by the algorithm. We include these features, because we think they provide an all around characterization of the Chip Production Scheduling Problem instances.

The average makespan *avgF* feature is calculated over n different landmarkers run on the instance and can be defined as follows: $\frac{1}{n}\sum_{k=1}^{n} f(r_i^x)$. The *avgR* feature is calculated by randomly sampling N solutions for an instance and calculating the average makespan of these solutions: $\mathbf{avg}(f(r_i^1), f(r_i^2), \ldots, f(r_i^N))$. Next, the *maxF* and *minF* features extract the maximum and minimum values of the search-space given by 2 iterations of the Genetic Algorithm. The expected proportion of neighbours with a better solution, *EPN*, is the proportion calculated between the makespan of a random solution and of its neighbourhood. A random solution r_i^j is picked along with its v neighbours n_i^j. The makespan of each neighbour is compared to the random solution, where we count the number of times the makespan of the neighbour is worse than that of the random solution and average the result.

The *ERN* gives the average ratio between the fitness values of m random solutions $\{r_i^1, r_i^2, \ldots, r_i^m\}$ and the corresponding fitness value of the each r_i^j's best neighbour (out of v neighbours). The computed ratios are averaged. The features *RFP*, *QBO* and *AOP* involve a set of t parent-solutions P_i: $\{(p_i^{1,1}, p_i^{1,2}), \ldots, (p_i^{t,1}, p_i^{t,2})\}$ that are selected from R_i. By applying **pmx** crossover to the parents, the offspring O_i: $\{(o_i^{1,1}, o_i^{1,2}), \ldots, (o_i^{t,1}, o_i^{t,2})\}$ are created. The quality of the best offspring solution, *QBO*, measures the average number of times that the best solution of an offspring pair is better than the best solution of the corresponding parents.
The *RFP* meta-feature instead calculates the average of the ratio between the fitness values from the best solutions of the offspring pairs and the parent pairs. Finally, the *AOP* meta-feature measures the average number of times an offspring solution is better than its parents.

3.3 Summary

We discussed the implemented metalearning solution for the Chip Production Scheduling Problem. The metalearning approach is based on meta-data, consisting of meta-features and a meta-targets for the instances. Since we address two metalearning tasks, 1) predicting the makespan of each meta-heuristic variant and 2) selecting the best meta-heuristic for a given instance, which represent regression and classification tasks we used different evaluation methods. For regression, the MSE (Mean Squared Error) and for the classification task prediction accuracy.

The implemented meta-features consist of simple features, proposed and adapted Genetic Algorithm landmarkers. We excluded some of the proposed meta-features from the experiments because of long computation time.

Experiments

We explain the experimental setup and results obtained with the metalearning framework.

4.1 Experimental Setup

The experimental setup (figure 4.1) can be divided into three phases, namely:

1. Input generation (meta-features and meta-targets) for the metalearning model
2. Training the algorithm to derive a metalearning model and running it on test instances
3. Evaluating the recommendations and the performance of the algorithm

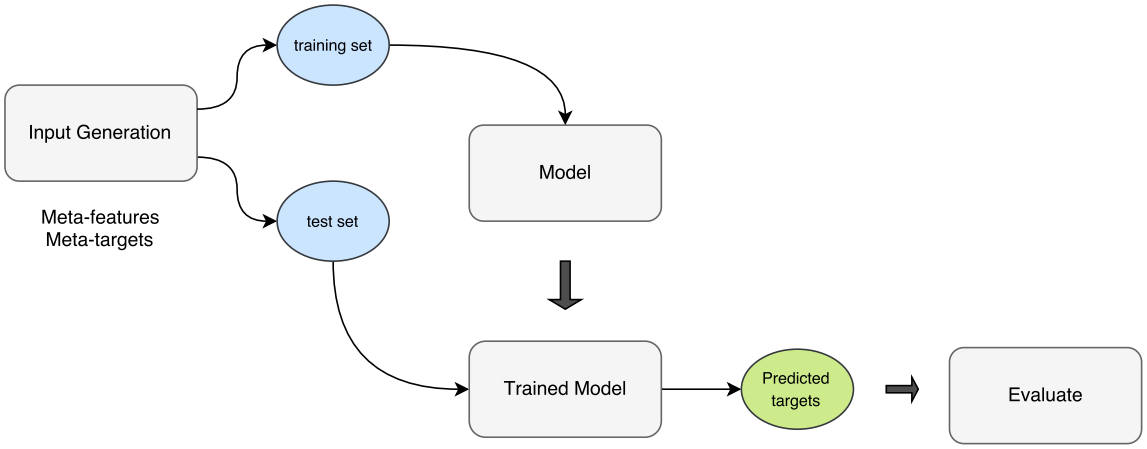

FIGURE 4.1: Experimental setup.

We used a Genetic Algorithm framework implemented in **R** [21] to generate most input from the training and test data. The simple meta-features were calculated directly from the data. The test data consists of 3 SEMATECH scheduling data sets derived from real data of several chip fabrication facilities [22]. The training data includes 41 randomly generated scheduling data sets from Taillard's Job Shop Scheduling Problem benchmarks [23]. In addition, we adjusted these to match the nonlinear test sets having unequal length. Namely, each set has a different number of jobs to process and multiple jobs can use the same machine. We extend each job from every set with a random number of machines.

Since the input generation of all features except the simple meta-features took a long time, we parallelized the scripts and distributed them over multiple cores using the **foreach** [24] and **doParellel**[25] packages. We give a brief overview of the setups used for the input generation except for the simple meta-features since these are directly computed from the data.

- We ran the avgF feature with 20 iterations, population sizes of 10 and 20, mutation probabilities $\in \{01, 0.01, 0.1\}$ and crossover probabilities $\in \{0.45, 0.5, 0.6, 0.75\}$. We chose these mutation and crossover probabilities because they are located in the range of known good parameter values for the Genetic Algorithm.

- All of the adapted features use a setup of the population size of 8 for the neighbourhood, 10 for both the random and parent population and a crossover probability of 1.

- The avgR, maxF, and minF features were derived from the GA setup of two iterations, a population size of 50 and mutation and crossover probabilities 01 and 0.6, respectively.

- As discussed in the previous chapter, the **meta-targets** are calculated with two setups. The first one, derived from the Grefenstette setup consists of 30 iterations, a population size of 30, and 0.01 and 0.95 for the mutation and crossover probabilities. The second setup, derived from the deJong setup similarly consists of 30 iterations, a population size of 50, and 01 and 0.6 for the mutation and crossover probabilities.

The choice of algorithms includes a simple Decision Tree (DT), Multilayer Perceptron (MLP), Nearest Neighbour (kNN), Random Forest (RF) and a Linear Model (LM). We used the implementation of the regression and classification variants of these algorithms in Python using the **Scikit-Learn** machine-learning package [26]. The important thing to note is that some of these are *stochastic* algorithms. This signifies that an algorithm produces different results for different runs because of randomness.

For the training run, we employed the Leave-One-Out training method. So to say, the models train on all but one instance of their training data and predict the value of the left-out instance. We used this training method due to the small size of the training data set. The training data consisted of a reduced set of meta-features: a subset of the landmarker features and all simple features. The reason for this is the long computation time. For the test run, we train the models on all training data and run them afterward the instances of the test data.

The following section reports the performance results of the training and test runs of the models

4.2 Results

4.2.1 Training Run

We designed various meta-features, so it follows that we train and test algorithms on these in separate experiments. We divided the training run into three different experiments. The first experiment trains with the simple meta-features, the second with the subset of landmarkers, and the third with the former two combined. The results per experiment are displayed in the following tables.

The results of the regression and classification tasks are presented in the next tables for all experiments. Both tasks were trained on 25 instances of the data. We normalized the meta-features by subtracting the mean of each feature and dividing the result by its standard deviation.

TABLE 4.1: Performance results from the Leave-One-Out Regression Task for all experiments on training data.

Performance Measures	**ZR**	**LM**	**kNN**	**DT**	**RF**	**MLP**
Simple features						
RMSE deJong	**0.0228**	0.0144	0.0171	0.0208	0.0158	0.0632
RMSE Gref.	**0.0264**	0.0211	0.0219	0.0352	0.0252	0.0755
Landmarker features						
RMSE deJong	**0.0228**	0.0144	0.0171	0.0214	0.0181	0.0956
RMSE Gref.	**0.0264**	0.0211	0.0220	0.0340	0.0241	0.0753
Combined						
RMSE deJong	**0.0228**	0.0216	0.0176	0.0212	0.0183	0.1249
RMSE Gref.	**0.0264**	0.0289	0.0213	0.0340	0.0265	0.1710

Notes: ZR-baseline = ZeroRule, LM = Linear Model, 5NN = Nearest Neighbour , MLP = Multilayer Perceptron, DT = Decision Tree and RF: Random Forest.

TABLE 4.2: Performance results from the Leave-One-Out Classification Task for all experiments on training data.

Performance Measures	**LM**	**kNN**	**DT**	**RF**	**MLP**
Simple features					
Accuracy Reg-Class.	64%	56%	64%	68%	60%
Accuracy Classification	28%	20%	32.0%	52%	48%
Landmarker features					
Accuracy Reg-Class.	64%	56%	76%	68%	52%
Accuracy Classification	28%	20%	40%	32%	40%
Combined					
Accuracy Reg-Class.	68%	68%	60%	64%	56%
Accuracy Classification	48%	20%	32%	48%	40%
Baseline Majority Class			**56%**		

Notes: ZR = ZeroRule baseline, LM = Logistic Model,kNN = Nearest Neighbour , DT = Decision Tree and RF = Random Forest.

The following subsections explain the results for each experiment, shown by the previous tables.

4.2.1.1 Simple Features

The results for the regression task show that all models except the Multilayer Perceptron (MLP) have better predictive performance than the ZR-baseline for the deJong setup on predicting the target gain. For the Grefenstette setup, the Decision Tree and Multilayer Perceptron had the worst predictive performance. Furthermore, the Linear Model (LM) and Random Forest (RF) got the best results for both Genetic Algorithm setups, with the Nearest Neighbour model following close behind. Finally, the Decision Tree (DT) acquired mixed results with a better predictive performance for the deJong setup than that of Grefenstette. Since the algorithms, except the Linear Model, are stochastic the predicted errors differ slightly per run. We confirmed this by running the models a few times, where the results did not improve significantly.

The classification results, displayed in table 4.2 indicate that the straightforward classification approach does not work well. The accuracy of the straightforward classification is worse than the baseline Majority Class. We repeated the classification task a few times, but the results did not improve. In contrast, the accuracy results for the classification by regression approach are usually than the baseline. In this case, the Random Forest (RF) outperformed the other models with the best accuracy score. However, the repeated classification tasks show that for the classification by regression approach, the accuracy of the models stays more or less the same.

4.2.1.2 Landmarker features

Once again, the results for the regression task show that the Multilayer Perceptron (MLP) acquired the worst predictive performance of all models, as illustrated in the next table. For this experiment, the Linear Model (LM) and Nearest Neighbour (kNN) models acquired the best results for both Genetic Algorithm setups. The performance results of the Random Forest (RF) follow close behind. The Decision Tree (DT) once again acquired variable results, where it had better RMSE values for the deJong setup than that of Grefenstette.

The results of the classification for this experiment are similar to the previous one. Namely, the models fail the straightforward classification task as can be clearly seen in the results in the second row. This time, the Decision Tree (DT) acquired the best accuracy performance (76%) for the classification by regression task, followed closely by the Random Forest (68%). Incidentally, the same comments regarding the variation in the accuracy of the classification by regression task RMSE values of the regression task apply here too.

The performance results obtained by the models on the simple features and landmarkers reveal that simple features already lead to good performance results.

4.2.1.3 Simple and Landmarker features

The results for the regression task show that all models except the Multilayer Perceptron (MLP) have better predictive performance than the ZR-baseline for the deJong setup. For the Grefenstette setup, these were only the Random Forest (RF) and Nearest Neighbour (kNN). In particular, the Random Forest and Nearest Neighbour models acquired the best results for both Genetic Algorithm setups (deJong and Grefenstette). The Decision Tree (DT) and Linear Model (LM) acquired variable results. Both have better RMSE errors for the deJong setup than that of Grefenstette. Finally, the Multilayer Perceptron (MLP) had the worst performance of the entire set of models based on the poor results it acquired for both Genetic Algorithm setups.

The classification results, confirm once again the conclusion that emerged from the previous experiments: that worse results are obtained with a traditional classification approach to this problem.

The accuracy of the straightforward classification is worse than the baseline Majority Class. On the other hand, the results for the classification by regression approach are as good or better than the baseline. The Logistic Model (LM) and Nearest Neighbour (kNN) thereby outperformed the other models with the best accuracy score.

The first conclusions that can be drawn from these experiments are as follows:

- The straighforward classification approach to this problem obtained unsatisfactory results so it will not be used on the test data.

- The Random Forest algorithm consistently performed well in the regression tasks for all experiments. For the classification by regression tasks, the algorithms acquired different accuracy scores per experiment.
- There is little difference between the predictive performance of the models on the different sets of meta-features for the regression-tasks.

Given the first conclusions, we look at the predicted performance of the Random Forest for the regression task on the simple features, in more detail.

For each instance we converted the predicted gains $y_{recommend}$ from the Random Forest to their respective makespan values. We separately plotted the original makespan value *true* computed by the Genetic Algorithm and the predicted value, to which we refer to as *pred*, for both setups. Figure 4.2 presents the results:

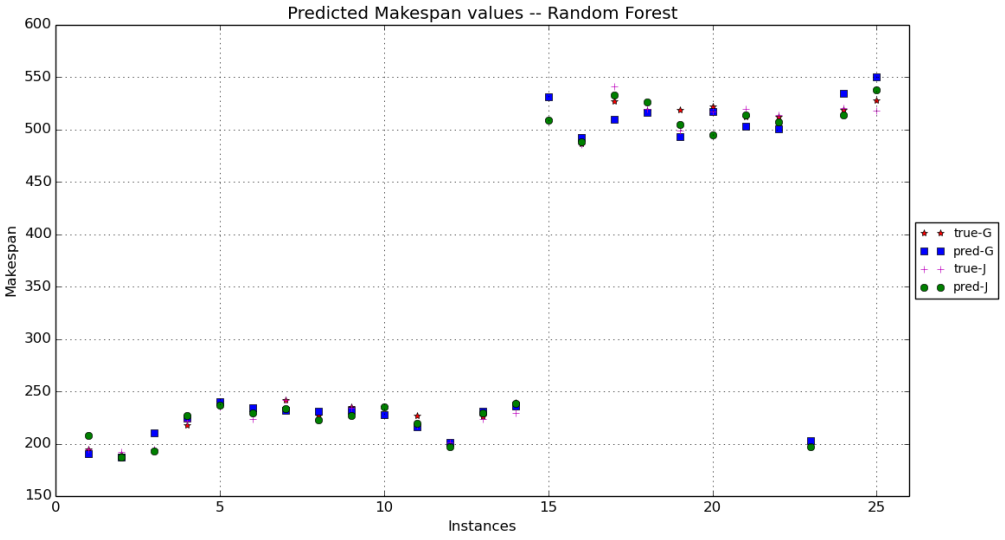

FIGURE 4.2: Makespan predictions by the Random Forest model, for both Genetic Algorithm setups.

It is clear that the predictive error of the makespan values is small, as shown by the overlapping points in the plot. Since the classification-regression error is difficult to deduce from this plot, we present those of the instances with the lowest RMSE values of the predicted gains $y_{recommend}$ for both setups:

TABLE 4.3: Classification by regression predictions of best, low and worst RMSE values from predicted gains

Instance	**Setup**	**RMSE for $y_{recommend}$**	**Pred. Class**	**True Class**
23	deJong	**0.00088**	deJong.	deJong
5	Gref.	**0.00058**	deJong	deJong.
10	deJong	**0.00611**	Gref.	deJong
12	Gref.	**0.00298**	deJong	Gref.
1	deJong	**0.05903**	Gref.	Gref.
3	Gref.	**0.07531**	deJong	Gref.

The results show that in general, most of the time the meta-model predicts the same setup as the original for data sets with low RMSE values. Furthermore, the predicted makespans converted from the gains with the correct classification prediction followed the same classification. This indicates that, given the low RMSE values, the approach selects the best Genetic Algorithm for an instance, leading to the best scheduling solution. Despite the difference in prediction accuracy, we think this model and the Nearest Neighbour are good options for the classification by regression task of the test data.

4.2.2 Test Run

Given the results of the training run, we decided on the Random Forest and Nearest Neighbour as the algorithms. We trained the algorithms on the same meta-features sets we investigated in the three experiments. Afterward, we ran the models on 3 test instances of the Chip Production Scheduling Problem described with the same meta-features as the training set. Table 4.4 displays the results, including the Zero-Rule (ZR) baseline.

TABLE 4.4: Performance results from the Regression Task for all experiments on test data .

Performance Measures	**ZR**	**RF**	**kNN**
Simple features			
RMSE deJong	0.0311	0.0290	0.0354
RMSE Gref.	0.0319	0.0339	0.0399
Landmarker features			
RMSE deJong	0.0311	0.0386	0.0360
RMSE Gref.	0.0319	0.0379	0.0357
Combined			
RMSE deJong	0.0311	0.0379	0.0356
RMSE Gref.	0.0319	0.0415	0.0457

The Random Forest algorithm has the best results on the simple set, with the lowest RMSE values for its predicted gains $y_{recommend}$, although the obtained RMSE value for the Grefenstette setup is worse than the baseline. The Nearest Neighbour model also obtained the best results on the simple meta-features set, but its RMSE values were also worse than the baseline. The remaining performance results show higher RMSE values than those of the baseline and the simple meta-feature set, for both algorithms. The classification accuracy performances for the classification by regression task were in general better. The calculated majority class baseline for the test data was 66.67%. Both models obtained classification accuracy performances of 66.67% for the combined set. The Nearest Neighbour algorithm subsequently obtained 100% and 33.33% accuracy performances for the landmarker and simple features respectively. Finally, Random Forest got an accuracy performance of 66.67% for both the simple and landmarker sets. We repeated the experiments and noticed little difference in the RMSE values and classification accuracy performances.

Again, we analyze the predicted gains which are converted to predicted makespans *pred*. Figures 4.3 and 4.4 display the predictions acquired by the models for the three instances on the simple set of meta-features of the test data.

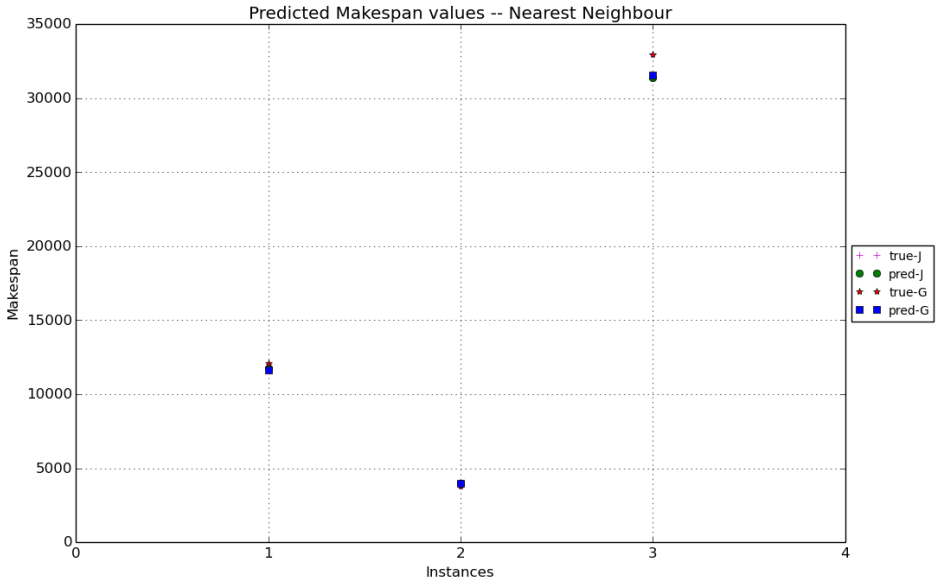

FIGURE 4.3: Makespan predictions for the test sets by the Nearest Neighbour model, for both Genetic Algorithm setups.

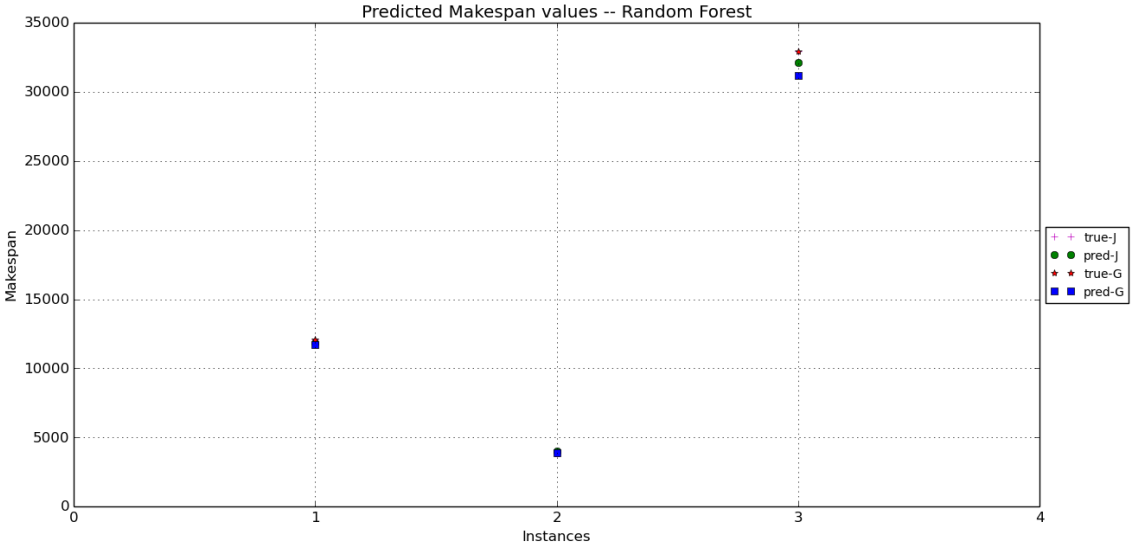

FIGURE 4.4: Makespan predictions for the test sets by the Random Forest model, for both Genetic Algorithm setups.

Here too, we observe the same results as in the makespan prediction plot of the training run which are the very small predictive errors between the original and the predictive makespan values. We subsequently look at the classification by regression errors for both models, on both setups:

TABLE 4.5: Classification by regression predictions of best,low and worse RMSE values from predicted gains.

Instance	**Setup**	**RMSE for $y_{recommend}$**	**Pred. Class**	**True Class**
Random Forest				
1	deJong	**0.01886**	Gref.	deJong.
4	Gref.	**0.01553**	Gref.	Gref.
7	deJong	**0.02492**	Gref.	Gref.
1	Gref.	**0.03238**	Gref.	deJong
4	deJong	**0.04326**	Gref.	Gref.
7	Gref.	**0.05379**	Gref.	Gref.
Nearest Neighbour				
1	deJong	**0.02701**	Gref.	deJong
1	Gref.	**0.03491**	Gref.	deJong
4	deJong	**0.03080**	Gref.	Gref.
7	Gref.	**0.04209**	deJong	Gref.
7	deJong	**0.04829**	deJong.	Gref.
4	Gref.	**0.04286**	Gref.	Gref

The models predicted the correct setup for instance 4. The Random Forest model also predicted the correct Genetic Algorithm for instance 7, regardless of the higher RMSE values. The Nearest Neighbour model furthermore predicted incorrect setups for two out of the three instances. It shows that lower (better) RMSE values do not necessarily correspond with a correct classification. By looking at the converted makespan values of the predicted gains, we discover that the wrongly predicted classes for the first data set were, in fact, better than the makespan values for the 'would-be' correctly predicted class of the deJong setup. We assume that this has to do with the regression-error. If the model, for instance, obtained perfect regression-errors of zero, its predicted makespan values would be the same as the true values. Hence, it would predict the same classification as the true classification.

4.3 Discussion

The training experiments revealed various things. Addressing the algorithm selection problem tackled in this project as a straightforward classification problem does not lead to good results. On the other hand, the regression models acquired good predictive performance (low RMSE values) on all sets of meta-features for the regression task. From the set of meta-models, the Random Forest and Nearest Neighbour algorithms were the best.

There was also little difference between the performance on the different sets. So in this case, simple meta-features would indeed provide the minimal basis of data-characterization of the instances for the meta-model to do good predictions for the Chip Production Scheduling Problem instances. We worked with a preliminary set of landmarkers but we expect that the inclusion of other meta-features of this kind, some of which have already been discussed in this project, may improve the results, as they did in other selection tasks.

Despite the small sample size of the training set, the Random Forest predicted the correct Genetic Algorithm for the instances for which it acquired the lowest RMSE values in the classification by regression task. Naturally, the model did not predict the correct setups for all instances, but this can be expected. Here too, the prediction accuracy can be improved taking the same measures that would improve the accuracy of the regression task.

For the test experiments the chosen algorithms, Random Forest and Nearest Neighbour, achieved mixed results on the regression task. The performance results obtained by both models indicate that simple meta-features provide good data-characterization, given the low RMSE values. The results of the classification by regression show that Random Forest did better than the Nearest Neighbour. Both meta-models predicted the correct setups for two out of three instances for the combined meta-feature set. For the remaining sets, the Random Forest correctly predicted the setups for two out three instances. In contrast, the Nearest Neighbour predicted correctly predicted the setup for only one instance, for the simple meta-feature set. For the landmarker set, it predicted the correct setup for two out of thee instances.

The general observation can be made that given the small amount of data we worked with, the metalearning approach to the Chip Production Scheduling Problem achieved promising results. We carefully assume that the complete set of meta-features and a greater sample size of training data will lead to better results.

Conclusions

In this thesis, we designed a metalearning approach to select one of two Genetic Algorithms to find the best scheduling solutions for real-world instances of the Chip Production Scheduling Problem, a Job Shop Scheduling Problem. The Chip Production Scheduling Problem is about finding an optimal schedule of job assignments to a set of machines, which reduces the makespan of multiple orders of different products. We modeled the Chip Production Scheduling Problem as regression and classification tasks. The classification problem was thereby approached as a straightforward classification task and a regression-classification task. With straightforward classification, the algorithms predict the targets, i.e. the algorithm that works best on the instance. While with regression-classification, the algorithms predicted the class by comparing the makespan values of the predicted regression targets for each instance, and choosing the algorithm with the lowest one as prediction.

The training data for the metalearning process were a set of artificially generated Job Shop Scheduling Problem instances and the test data the real-world instances of the Chip Production Scheduling Problem. The instances were described with simple and landmarker meta-features. We extracted statistical information from the instances to form the simple meta-features. Afterward, we adapted and modified landmarker features from the paper of J.Kanda [20] and also proposed some new ones. We computed different meta-targets for the regression and classification tasks. For the regression task, this was the gain between the makespan of the random scheduling solution, and the best one found by the Genetic Algorithm for the two Genetic Algorithms. For classification, we computed binary class labels that indicated which algorithm had lower makespan values,

thus a better scheduling solution. We used a Random Forest, Decision Tree, k-Nearest Neighbour, Multilayer Perceptron and Linear algorithms to learn the meta-model.

We trained the meta-models on each set of meta-features and on the whole set altogether, where we worked with a preliminary set of landmarkers and a small sample size of training instances. The results of the training experiment showed little performance difference between the meta-models for the regression task. We noticed that simple meta-features already obtained good performance results.

The Random Forest and Nearest Neighbour algorithms were the best of the training set with the lowest RMSE scores. This translated into small predictive errors between the original optimal makespan, and the predicted makespan. Another finding is the fact that a straightforward classification approach to predicting the best algorithm for an instance does not work, given the bad performance accuracy scores obtained by the models. Namely, they were worse than the Majority-class baseline. In contrast, the regression-classification task proved more successful. Here too, we noticed that the Random Forest algorithm was one of the more successful algorithms.

For the test experiment, we subsequently trained the Random Forest and Nearest Neighbour meta-models again on all meta-feature sets of the training data and ran them on same the meta-features of the test instances. Both obtained the best results for the simple meta-feature set with the lowest RMSE values. The remaining results indicated no improvement in predictive performance for both meta-models for the other sets of meta-features.

The classification by regression task showed that the Random Forest obtained better results than the Nearest Neighbour algorithm. For all meta-feature sets, it predicted the correct algorithm for two out of three instances. In contrast, the Nearest Neighbour algorithm predicted the correct algorithm for one instance, for the simple meta-features set. It subsequently predicted the correct algorithm for two out of thee instances for the landmarkers set.

What appears from all these findings we summarize as follows.

- The metalearning approach we designed obtains promising results for recommending i.e. predicting the best Genetic Algorithm for the instances of the Chip Production Scheduling Problem.
- The performance of the meta-models does not vary much between the different sets of meta-features. We saw that simple meta-features seem to work as well as the landmarker features for this problem, given the results we obtained.
- By tuning the parameters of the meta-models and improving the set of meta-features, a better metalearning approach can be constructed.

5.1 Future Work

There is still a lot of work to be done in improving the designed metalearning approach. For instance, we could design meta-features that exploit more properties of the instances of the Chip Production Scheduling Problem. We also suggest trying out different Genetic Algorithm operators, implementing another heuristic algorithm and tuning the parameters of the meta-models. Finally, we recommend increasing the training data by ten-fold.

Appendix

TABLE A.1: Simple meta-features.

Meta-feature	**Mathematical Expressions**
Simple	
avgM	$\frac{1}{M}(\sum_{j=1}^{M}\sum_{i=1}^{k} m_i)$
avgT	$\frac{1}{M}(\sum_{j=1}^{M}(\frac{1}{k}\sum_{i=1}^{k} t_i))$
HT	$\mathbf{max}([t_1, t_2, \ldots, t_k]^{j_1}, \ldots, [t_1, t_2, \ldots, t_k]^{j_M})$
LT	$\mathbf{min}([t_1, t_2, \ldots, t_k]^{j_1}, \ldots, [t_1, t_2, \ldots, t_k]^{j_M})$
varT	$\frac{1}{M}(\sum_{j=1}^{M} var(t_1, t_2, \ldots, t_k))$
kurT	$\frac{1}{M}(\sum_{j=1}^{M} kur(t_1, t_2, \ldots, t_k))$

TABLE A.2: Landmarker meta-features .

Meta-feature	**Mathematical Expressions**
Proposed	
avgF	$\frac{1}{n}\sum_{k=1}^{n} f(r_i^x)$
avgR	$\mathbf{avg}(f(r_i^1), f(r_i^2), \ldots, f(r_i^N))$
maxF	$\mathbf{max}(f(r_i^j))$
minF	$\mathbf{min}(f(r_i^j))$
Adapted	
EPN	$\frac{1}{v}\sum_{k=1}^{v} \mathbf{chk}(f(n_i^{k,j}) < f(r_i^j))$
ERN	$\frac{1}{m}\sum_{k=1}^{m} \frac{f(n_i^{k,best})}{f(r_i^k)}$
QBO	$\frac{1}{j}\sum_{j=1}^{t} \frac{\mathbf{chk}(f(\mathbf{min}(o_i^{j,1},o_i^{j,2})) < f(\mathbf{min}(p_i^{j,1},p_i^{j,2})))}{t}$
RFP	$\frac{1}{j}\sum_{j=1}^{t} \frac{f(\mathbf{min}(o_i^{j,1},o_i^{j,2}))}{f(\mathbf{min}(p_i^{j,1},p_i^{j,2}))}$
AOP	$\frac{1}{4t}\sum_{j=1}^{t}\sum_{k=1}^{2}\sum_{l=1}^{2} \mathbf{chk}(f(o_i^{j,k}) < f(p_i^{j,l}))$

Bibliography

[1] Yahong Zheng, Lian Lian, and Khaled Mesghouni. Comparative study of heuristics algorithms in solving flexible job shop scheduling problem with condition based maintenance. *Journal of Industrial Engineering and Management*, 7(2):518–531, 2014. ISSN 2013-0953. doi: 10.3926/jiem.1038. URL http://www.jiem.org/index.php/jiem/article/view/1038.

[2] Sönke Hartmann and Rainer Kolisch. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 127(2):394–407, 2000.

[3] Peter J. M. van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Oper. Res.*, 40(1):113–125, January 1992. ISSN 0030-364X. doi: 10.1287/opre.40.1.113. URL http://dx.doi.org/10.1287/opre.40.1.113.

[4] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, and Marco Trubian. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.

[5] Antonio Garrido, Miguel A Salido, Federico Barber, and MA López. Heuristic methods for solving job-shop scheduling problems. In *Proc. ECAI-2000 Workshop on New Results in Planning, Scheduling and Design (PuK2000)*, pages 44–49, 2000.

[6] Shijin Wang and Jianbo Yu. An effective heuristic for flexible job-shop scheduling problem with maintenance activities. *Computers & Industrial Engineering*, 59(3):436 – 447, 2010. ISSN 0360-8352. doi: http://doi.org/10.1016/j.cie.2010.05.016. URL http://www.sciencedirect.com/science/article/pii/S0360835210001439.

[7] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford, UK, 1996. ISBN 0-19-509971-0.

[8] Appendix a - genetic algorithm. URL http://www.wardsystems.com/manuals/genehunter/crossover_of_enumerated_chromosomes.htm.

[9] Appendix a - genetic algorithm. URL http://www.wardsystems.com/manuals/genehunter/mutation_of_enumerated_chromosomes.htm.

[10] Bernard Giffler and Gerald Luther Thompson. Algorithms for solving production-scheduling problems. *Operations research*, 8(4):487–503, 1960.

[11] Pavel Brazdil, Ricardo Vilalta, Christophe Giraud-Carrier, Carlos Soares, and Geoffrey I. Webb. *Metalearning*, pages 818–823. Springer US, Boston, MA, 2017. ISBN 978-1-4899-7687-1. doi: 10.1007/978-1-4899-7687-1_543. URL http://dx.doi.org/10.1007/978-1-4899-7687-1_543.

[12] John R Rice. The algorithm selection problem. *Advances in computers*, 15:65–118, 1976.

[13] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2009.

[14] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[15] Kenneth A. De Jong and William M. Spears. An analysis of the interacting roles of population size and crossover in genetic algorithms. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, PPSN I, pages 38–47, London, UK, UK, 1991. Springer-Verlag. ISBN 3-540-54148-9. URL http://dl.acm.org/citation.cfm?id=645821.670188.

[16] J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):122–128, Jan 1986. ISSN 0018-9472. doi: 10.1109/TSMC.1986.289288.

[17] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[18] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009. doi: 10.4249/scholarpedia.1883. revision #136646.

[19] Multilayer perceptron network. URL http://users.ics.aalto.fi/harri/thesis/valpola_thesis/node43.html.

[20] Jorge Kanda, Andre de Carvalho, Eduardo Hruschka, Carlos Soares, and Pavel Brazdil. Meta-learning to select the best meta-heuristic for the traveling salesman problem. *Neurocomput.*, 205(C):393–406, September 2016. ISSN 0925-2312. doi: 10.1016/j.neucom.2016.04.027. URL https://doi.org/10.1016/j.neucom.2016.04.027.

[21] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016. URL https://www.R-project.org/.

[22] Sematech datasets. URL ftp://ftp.eas.asu.edu/pub/centers/masmlab/factory_datasets.

[23] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278 – 285, 1993. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/0377-2217(93)90182-M. URL http://www.sciencedirect.com/science/article/pii/037722179390182M. Project Management anf Scheduling.

[24] Revolution Analytics and Steve Weston. *foreach: Provides Foreach Looping Construct for R*, 2015. URL https://CRAN.R-project.org/package=foreach. R package version 1.4.3.

[25] Revolution Analytics and Steve Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2015. URL https://CRAN.R-project.org/package=doParallel. R package version 1.0.10.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.