DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Using machine learning for resource provisioning to run workflow applications in IaaS Cloud

WILLIAM VÅGE

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Using machine learning for resource provisioning to run workflow applications in IaaS Cloud

WILLIAM VÅGE

Master in Computer Science
Date: December 3, 2019
Supervisor: Hamid Reza Faragardi
Examiner: Elena Troubitsyna
School of Electrical Engineering and Computer Science
Host company: KTH
Swedish title: Att använda maskininlärning till resursförmedling för att köra arbetsflödesapplikationer i molnet

Abstract

The rapid advancements of cloud computing has made it possible to execute large computations such as scientific workflow applications faster than ever before. Executing workflow applications in cloud computing consists of choosing instances (resource provisioning) and then scheduling (resource scheduling) the tasks to execute on the chosen instances. Due to the fact that finding the fastest execution time (makespan) of a scientific workflow within a specified budget is a NP-hard problem, it is common to use heuristics or metaheuristics to solve the problem.

This thesis investigates the possibility of using machine learning as an alternative way of finding resource provisioning solutions for the problem of scientific workflow execution in the cloud. To investigate this, it is evaluated if a trained machine learning model can predict provisioning instances with solution quality close to that of a state-of-the-art algorithm (PACSA) but in a significantly shorter time. The machine learning models are trained for the scientific workflows Cybershake and Montage using workflow properties as features and solution instances given by the PACSA algorithm as labels. The predicted provisioning instances are scheduled utilizing an independent HEFT scheduler to get a makespan.

It is concluded from the project that it is possible to train a machine learning model to achieve solution quality close to what the PACSA algorithm reports in a significantly shorter computation time and that the best performing models in the thesis were the Decision Tree Regressor (DTR) and the Support Vector Regressor (SVR). This is shown by the fact that the DTR and the SVR on average are able to be only 4.97 % (Cybershake) and 2.43 % (Montage) slower than the PACSA algorithm in terms of makespan while imposing only on average 0.64 % (Cybershake) and 0.82 % (Montage) budget violations. For large workflows (1000 tasks), the models showed an average execution time of 0.0165 seconds for Cybershake and 0.0205 seconds for Montage compared to the PACSA algorithm's execution times of 57.138 seconds for Cybershake and 44.215 seconds for Montage. It was also found that the models are able to come up with a better makespan than the PACSA algorithm for some problem instances and solve some problem instances that the PACSA algorithm failed to solve. Surprisingly, the ML models are able to even outperform PACSA in 11.5 % of the cases for the Cybershake workflow and 19.5 % of the cases for the Montage workflow.

Sammanfattning

De snabba framstegen inom molntjänster har gjort det möjligt att genomföra stora beräkningar som exempelvis vetenskapliga arbetsflödesapplikationer snabbare än någonsin. Att köra arbetsflödesapplikationer i molnet består av att välja instanser(resursförmedling) och sedan schemalägga(resursschemaläggning) de deluppgifter inom arbetsflödet som ska utföras på de valda instanserna. Att hitta den snabbaste exekveringstiden (makespan) för ett vetenskapligt arbetsflöde inom en specificerad budget är ett NP-svårt problem och det är vanligt att använda heuristik eller metahuristik för att lösa problemet.

Detta examensarbete undersöker möjligheten att använda maskininlärning som ett alternativt sätt att hitta resursförsörjningslösningar för exekvering av vetenskapliga arbetsflöden i molnet. För att undersöka detta utvärderas om en tränad maskininlärningsmodell kan förutsäga resursförmedlingslösningar med lösningskvalitet nära den för en state-of-the-art algoritm (PACSA) fast med en betydligt kortare beräkningstid. Maskininlärningsmodellerna tränas för de vetenskapliga arbetsflödena Cybershake och Montage med hjälp av arbetsflödesegenskaper som features och lösningsinstanser givna av PACSA-algoritmen som labels. De förutspådda instanserna är schemalagda med en oberoende HEFT-schemaläggare för att få en makespan.

Av projektet dras slutsatsen att det går att träna en maskininlärningsmodell så att den uppnår lösningskvalitet nära vad PACSA-algoritmen rapporterar fast med en betydligt kortare beräkningstid och att de bästa modellerna utvärderade i examensarbetet var en Decision Tree Regressionsmodell och en Support Vector Regressionsmodell. Slutsatsen påvisas av det faktum att DTR och SVR i genomsnitt lyckas vara bara 4.97 % (Cybershake) och 2.43 % (Montage) långsammare än PACSA-algoritmen gällande makespan samtidigt som de bara inför i genomsnitt 0,64 % (Cybershake) och 0,82 % (Montage) budgetöverträdelser. För stora arbetsflöden (1000 uppgifter) visade modellerna en genomsnittlig exekveringstid på 0,0165 sekunder för Cybershake och 0,0205 sekunder för Montage jämfört med PACSA-algoritmens exekveringstid på 57,138 sekunder för Cybershake och 44,215 sekunder för Montage. Det har också visat sig att modellerna lyckas få en bättre makespan än PACSA-algoritmen för vissa problemfall och lyckas lösa vissa problemfall som PACSA-algoritmen inte lyckades lösa. Ett förvånande resultat var att ML-modellerna till och med överträffade PACSA i 11,5 % av fallen för Cybershake-arbetsflödet och 19,5 % av fallen för Montage-arbetsflödet.

Contents

Chapter 1

Introduction

The advancements of cloud computing has made it possible to execute large computations faster than before. As a result it is now possible to execute large scientific workflow applications distributed in the cloud. A workflow is an application split into parts that consist of a series of computational tasks that are connected by control-flow and data dependencies. However, by using the computing power now available through the cloud users will be charged according to how much computing power they have used. A cloud customer often has a fixed budget or a threshold to not violate but still wants to execute computations or workflows as fast as possible within that budget.

Executing workflow applications in cloud computing consists of scheduling tasks on chosen available instances. The problem can be divided into two parts: 1) Selecting the instances to use for computation (resource provisioning), 2) Scheduling the tasks on the chosen instances (resource scheduling). Instances vary in computing power, cost and characteristics, some are better for raw computation, others for memory, storage or graphical processing.

Finding the fastest execution time (makespan) of a workflow within a specified budget is a NP-hard problem [1], which makes it unfeasible to solve through an exhaustive search for a large-scale size of the problem. It is common to use heuristics or meta-heuristic algorithms to address the problem.

1.1 Problem statement

The goal of this thesis is to investigate the use of machine learning for solving the resource provisioning problem for scientific workflow execution in the cloud. Further, this thesis investigates if machine learning could be used as a faster way of doing resource provisioning while still maintaining solution quality. To accomplish this, this thesis aims to find out if a machine learning model can achieve similar performance as a state-of-the-art algorithm while doing so in a significantly shorter time. This will be investigated by comparing the performance of the machine learning models to the PACSA algorithm (presented in the Theory Chapter) in terms of makespan and execution time for the two scientific workflows Cybershake and Montage. The Cybershake workflow stems from a research project where the goal is to develop a physics-based understanding of earthquakes and use it to be able to reduce hazards from earthquakes in the region of Southern California [2]. The Montage workflow is used for doing the calculations to generate custom mosaics of the sky using sample images as input [3]. Montage is an open source toolkit for the purpose of generating custom mosaics of the sky [3].

1.2 Purpose

The main purpose is to investigate machine learning as a method for resource provisioning for executing scientific workflows in the cloud. Being able to quicker get provisioning and scheduling solutions would enable scientists running large workflows to waste less time trying to find a way to schedule their workflows within the available budget. While the current state-of-the-art algorithms are too time consuming, a quicker way of doing resource provisioning could also enable scientists to use the method as a dynamic (online) resource provisioning algorithm which can work in less than a second. Another purpose could be to inspire researchers to further explore the use of machine learning applied to the research areas of resource provisioning and resource scheduling for workflow execution in the cloud.

1.3 Problem definition

The thesis assignment entails gathering data for training supervised machine learning models using a state-of-the-art algorithm for resource provisioning in cloud computing. The algorithm, PACSA, is developed by my thesis supervisor and colleagues. The algorithm aims at solving both the resource provisioning and task scheduling problem for scientific workflows in cloud computing. A workflow is commonly represented as a Directed Acyclic Graph (DAG) modeling the workflow and it's properties. Given a problem instance consisting of a workflow and a budget, PACSA provides a solution instance consisting of both resource provisioning and scheduling. The training data will consist of provisioning solutions that PACSA provides (labels) and the input (features) will be DAG's modeling the workflow and its properties and a budget. The machine learning goal is to be able to predict which and how many of each of the instance types that should be provisioned for a scientific workflow. The training and evaluation are done on a fixed set of Amazon EC2 instances which are described in more detail in the Method Chapter.

It will be investigated if a machine learning model can be trained to predict resource provisioning instances that perform as good as the PACSA algorithm in terms of solution quality but finding the solution in significantly less execution time. Good solution quality can be defined as solutions that:

- are not violating the budget constraint and are feasible.
- Have a short makespan(workflow completion time) that is close to the optimum. Optimal in this case could be considered close to what PACSA algorithm reports for the same problem instance or better.

To be able to determine if a predicted provisioning has a short makespan, an independent scheduler will be employed to the predicted provisioning to give it a makespan as only a predicted provisioning doesn't contain a makespan. The scheduling of the machine learning predicted resources to provision will be scheduled by an implementation of the well-known HEFT algorithm [4] which is explained in more detail in the Method Chapter.

A solution to the resource provisioning problem predicted by a machine learning model should contain a vector of variables(targets) where each target is a number representing how many of each instance are used for the resource provisioning. Below in figure 1.1, two illustratory examples of what input and output to the problem could look like are shown.

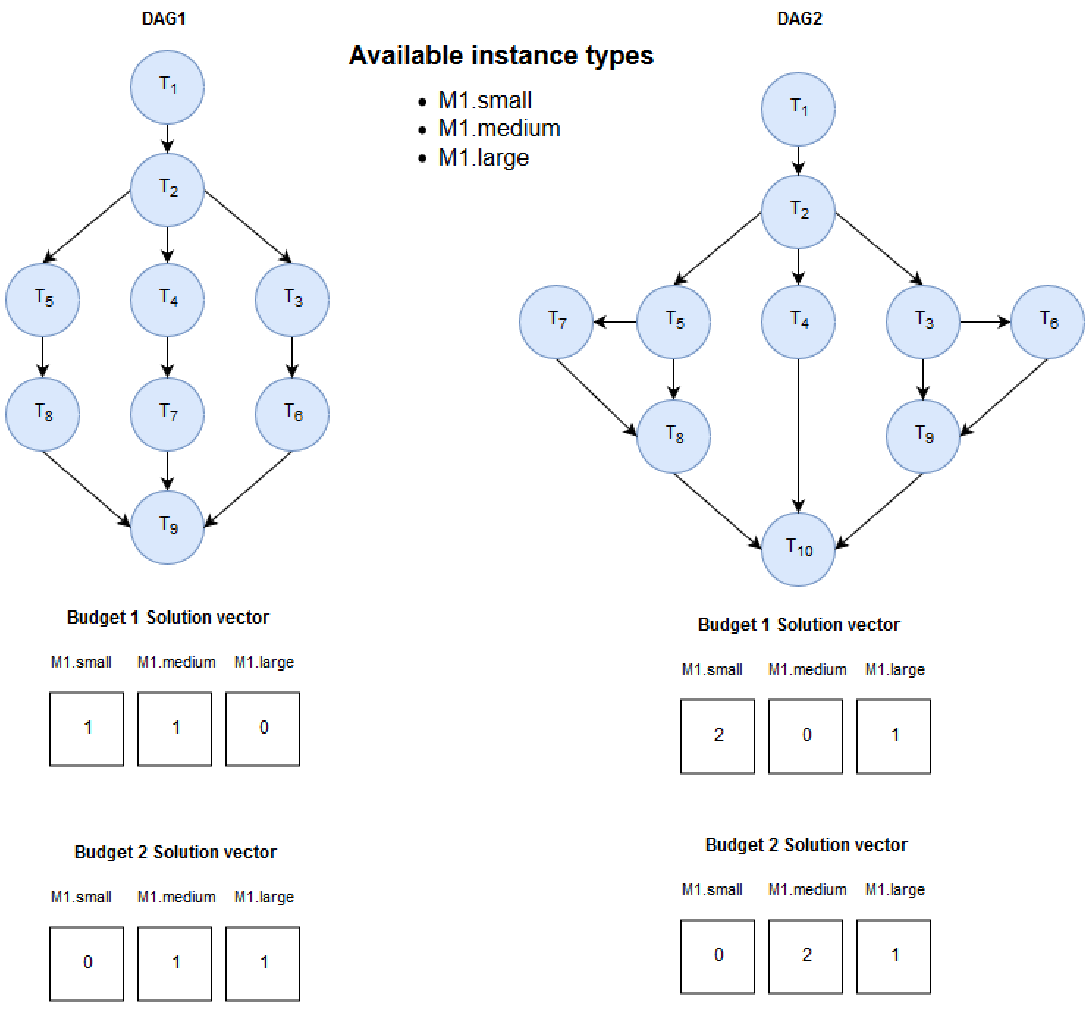

Figure 1.1: Illustration of two input DAG's with corresponding solution vectors for two different budgets each.

In figure 1.1, two example input DAGs are presented along with two different solution vectors for each DAG. The two vectors are two different solution instances illustrating that with a different budget the better way to provisioning resources may change.

The target values in the training data are the resource provisioning solutions that the PACSA algorithm suggested which are solutions of good quality. However, as a consequence of the nature of the problem and the definition of good solution quality, a provisioning solution instance which is not the same as the target value can still be a valid solution, or could even be a good quality solution. For example, let's assume Solution 1 is the target in the training data generated by PACSA for a given problem instance. Solution 1 includes two m1small instances, one m1medium, and three m1large instances. Solution 2 is a predicted provisioning solution generated by a machine learning algorithm which includes two m1medium and three m1large instances. Although Solution 2 is not the same with Solution 1, it could be of good solution quality if it is not violating the budget constraint and the makespan reported for the solution is close to the makespan of Solution 1.

1.4 Research Question

The thesis will aim to answer the following question:

RQ1: Can a machine learning model be trained to predict solutions of good quality that is close to what a state-of-the-art algorithm (PACSA) reports for the resource provisioning problem for scientific workflows in cloud computing but in a significantly shorter computation time?

1.5 Research Objective

The research objective is to investigate machine learning as an alternative way of finding resource provisioning solutions for the problem of scientific workflow execution in the cloud. Furthermore, the objective is to investigate and evaluate if a trained machine learning model can give solutions with as good solution quality as the PACSA algorithm but in a significantly shorter time. The goal is to quicker be able to decide (using machine learning) on the resources to provision so that it can be fed on to a resource scheduling algorithm. The aim is to then be able to accomplish resource provisioning(with machine learning) paired with a scheduling algorithm to faster find good scheduling solutions for scientific workflows while still maintaining a good makespan for the workflow subject to the given budget.

1.6 Research methodology

Figure 1.2 illustrates an overall overview of the research process used in this thesis work. The research starts with three phases of the thesis project concerns finding and reading research related to the resource provisioning and scheduling for scientific workflow execution in the cloud as well as reading literature concerning machine learning methods. It will also entail defining research questions and goals of the thesis project.

Afterwards there will be an implementation phase followed by an evaluation phase. The implementation phase will entail doing the necessary prerequisites for a machine learning project such as pre-processing, feature selection and feature engineering. Then there will be implementation of machine learning models followed by an evaluation step for each of the models. The implementation and evaluation phase will be connected and it is possible that after doing some evaluation there will be a need for returning to study theory and related works more.

Finally there will be two phases of drawing conclusions, discussing the findings and finishing the report.

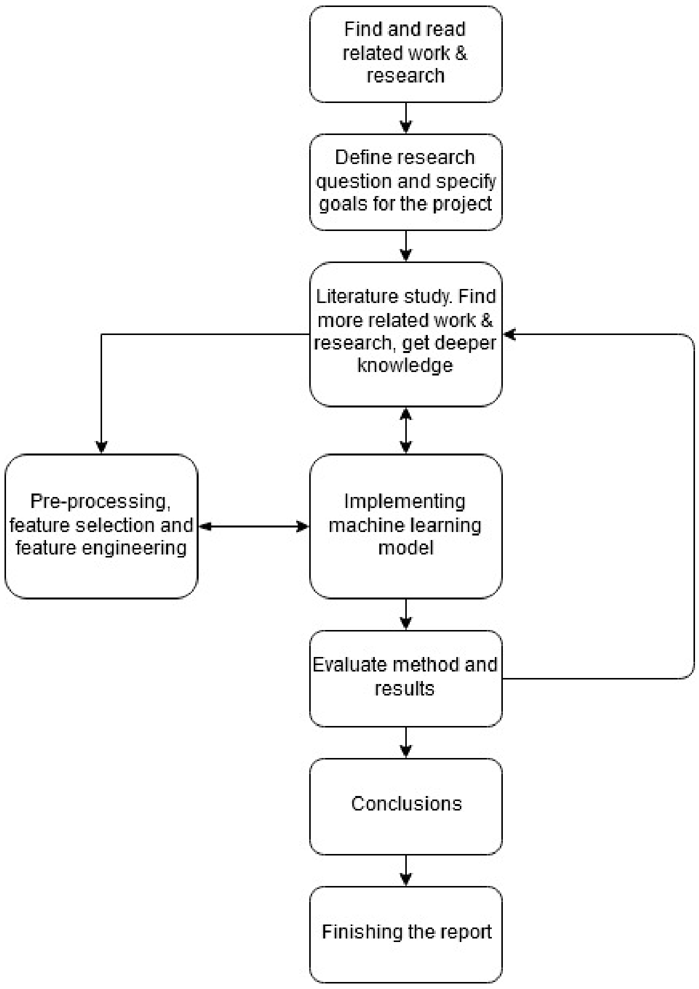
Find and read related work & research
Define research question and specify goals for the project
Literature study. Find more related work & research, get deeper knowledge
Pre-processing, feature selection and feature engineering
Implementing machine learning model
Evaluate method and results
Conclusions
Finishing the report

Figure 1.2: Research Methodology

1.7 Research delimitations

The degree project aims to investigate the learning ability of machine learning models trained on data consisting of solution instances from the PACSA algorithm for the resource provisioning problem in cloud computing.

The instance types considered for provisioning in this thesis are limited to Amazon's m1.small, m1.medium, m3.medium, m1.large, m3.large, m1.xlarge, m3.xlarge and m3.2xlarge. This limitation is due to that the algorithm (PACSA) used to generate data is using these instance types and the machine learning goal is to learn from that algorithm.

The scientific workflows used in this thesis to investigate the machine learning models learnability are the Cybershake workflow and the Montage workflow. Originally the plan was to use five types of workflow but due to the time constraints of the thesis and the time it takes to generate training data it was limited to two workflows.

An algorithm developed by my thesis supervisor and colleagues will be used in this thesis for generating training data and for comparing against. It solves the resource scheduling problem which resource provisioning is a part of. The algorithm will, in in its solution instances, provide both provisioning and scheduling solutions. This thesis will however focus on the resource provisioning part of the problem and exploring the possibility of solving it through machine learning. In other words, this thesis does not aim to investigate the use of machine learning for the resource scheduling step. However, for evaluation purposes a scheduling algorithm will be used on the predicted provisioning to get a makespan.

1.8 Thesis outline

In Chapter 2, background and relevant theory are presented. In Chapter 3, related work is presented. Chapter 4 presents the method used for conducting the thesis work. In Chapter 5 the results are presented and illustrated. In Chapter 6 there is a discussion and reflections on the work. Finally, in Chapter 7 the conclusions drawn from the thesis are presented as well as a future work section.

Chapter 2

Background & Theory

This chapter presents background and relevant theory for this project.

2.1 Infrastructure as a Service cloud

In the Infrastructure as a Service (IaaS) cloud architecture, the cloud provider manages a large set of computing resources, such as processing capacity and storage. In other words, it is the delivery of hardware such as server, storage and network as well as associated software (operating system and file system) as a service [5][6]. In the IaaS cloud, the cloud provider does very little management other than keeping the data centers up and running. The cloud consumer can deploy, configure and manage the software services like they would if they owned the hardware themselves [6].

The other types of architectural layers of cloud computing, as described in "A Break in the Clouds: Towards a Cloud Definition", are Platform as a Service (PaaS) and Software as a Service (SaaS). Platform as a Service is when the cloud provider provides the hardware and an additional certain amount of application software. Software as a Service means that the customer is offered a hosted set of software where you pay for what you use. It can be seen as an alternative to running applications locally [5].

2.2 Resource Management

Resource management for execution of workloads in the cloud consists of the two stages of Resource Provisioning and Resource Scheduling [7].

This thesis will mainly focus on the Resource Provisioning stage which is the first stage of of the process but the solution to the Resource Provisioning problem will serve as input to the Resource Scheduling problem.

2.2.1 Resource Provisioning

Resource provisioning is the task of selecting computational resources required for a computation. It is the task of selecting suitable virtual instances available from the cloud provider. Resource Provisioning have a different meaning for cloud providers where it means being able to provision enough resources for all their customers dynamically [8]. However, when Resource Provisioning is mentioned in this thesis it means the selection of computational resources by the cloud customer.

When wanting to execute a workflow in the cloud, the cloud customers themselves decide which resources and how many of the each type that they need for the execution of their workload.

Amazon EC2 is a well known cloud provider and EC2 leverages computing resources through a virtualization technology where virtual machines (VMs or EC2 instances) can be provisioned [9]. Before the provisioning of the resources for application workload, resource requirements must be specified. It contains characteristics of the resources, things like CPU architecture, disk space, region, memory and operating system [9].

Amazon provides four ways to pay for EC2 instances, On-Demand, Reserved Instances, Spot Instances and Dedicated Hosts [10]. On-Demand means that the customer pays for the computing capacity used per hour or per second depending on what instances are used [10]. The customer can increase or decrease the compute capacity depending on how much their application need [10]. Per second pricing is however only available on EC2 Linux instances [11].

Spot instances allow users to request spare EC2 computing capacity for up to 90 percent off compared to the On-Demand price [10]. Reserved Instances are when the customer reserves specific instances to ensure that they are available when the customer needs them. A Dedicated Host is a physical EC2 server dedicated for the customers use [10].

This thesis will focus on EC2 instances acquired with the On-Demand pricing with pay per hour.

2.2.2 Resource Scheduling

Resource Scheduling is the task of scheduling the workload(s) to given provisioned resources in the cloud [7]. When working with Scientific Workflows it means mapping the nodes (computations) to provisioned resources while still respecting the data dependencies/control flow [9]. The objective is finding the optimal way of scheduling different and possibly dependent tasks across provisioned instances [9]. The two primary metrics for the customer to consider are execution costs and execution time [9].

The idea of the thesis is to investigate machine learning as a way of doing quicker Resource Provisioning which will then be the input to the Resource Scheduling problem.

2.3 Scientific Workflows

Juve and Deelman [12] describes a scientific workflow as a program split into tasks that consist of a series of computational tasks that are connected by control-flow and data dependencies. Through the use of Workflows, several different computational processes can be combined into an organized whole. The size of a workflow can range from just a few tasks to millions of them. Large workflows with many tasks can be distributed to several computers in order to complete the work as fast as possible. Many different types of scientific problems can be expressed as workflows and thus they serve as a commonly used method to model computations within many science areas. Workflows can be used to combine components developed by different scientists at different times for different domains. Some examples of areas where scientific workflows are used are data analysis, simulation and image processing. Workflows often involve distributed computing on clusters, grids and the cloud. The provisioning model where the user can self-provision and self-schedule that is available through the cloud (such as IaaS clouds) are ideal for running workflow applications [12].

A workflow can be represented as a Directed Acyclic Graph(DAG) where nodes represent tasks and egdes between nodes represent data or control dependencies [13] [14]. This means that no child node (task) can be executed before all it's parents tasks have been completed. A Directed Acyclic Graph is exactly what it sounds like, a graph where the connections between nodes have a direction and there can be no cycles in the graph. Scientific workflows will occasionally be referred to as only workflows in this thesis.

An example of a small workflow is represented as a Directed Acyclic Graph is shown below in figure 1.

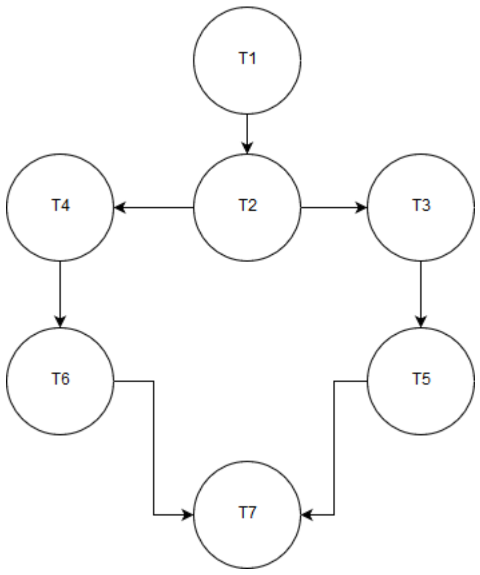

Figure 2.1: Example of a small Directed Acyclic Graph where nodes represent tasks and egdes represent data or control dependencies.

The workflows that are considered in this thesis are the Cybershake workflow and the Montage workflow both of awhich are presented below.

2.3.1 The Cybershake workflow

The Cybershake project and the Cybershake workflow is described in "SCEC CyberShake workflows—automating probabilistic seismic hazard analysis calculations" by Maechling et al. [2]. The project is operated by The Southern California Earthquake Center (SCEC) which is a community of more than 400 scientists from over 54 research organizations that conduct geophysical research. The research purpose is to develop a physics-based understanding of the processes of earthquakes and by that be able to reduce the hazards from earthquakes in the region of Southern California. The Cybershake Project goal is to utilize earthquake wave-propagation simulations to produce ground motion estimates that are to be used in PSHA hazard curves. The steps in the Cybershake workflow include preparation, simulation, postprocessing and analysis. The computational pathway can be divided into two main phases. An earthqake wave-propagation simulation phase and a postprocessing phase. The Cybershake scientific workflow is the model of the computational pathway [2].

Juve et al. [15] describes the Cybershake workflow and the different jobs it contains are in more detail in “Characterization of scientific workflows” [15]. Cybershake is divided into 5 types of jobs. It starts with the ExtractSGT jobs which are data partitioning jobs. Then the extracted data are used in the SeisomogramSynthesis jobs which generates Synthethic seismograms for each rupture variation. The SesimogramSynthesis job is the most computationally complex job and it amounts to more than 97 percent of the runtime. Peak intensity values are calculated by the PeakValueCalcOkaya jobs for each synthetic seismogram. The resulting synthethic seismograms and peak intensities are collected and compressed by the ZipSeismograms and ZipPeakSA jobs to then be stored in storage. The ZipSeismograms and ZipPeakSA jobs can be seen as simple data aggregation jobs [15].

An example of a Cybershake workflow is shown below.

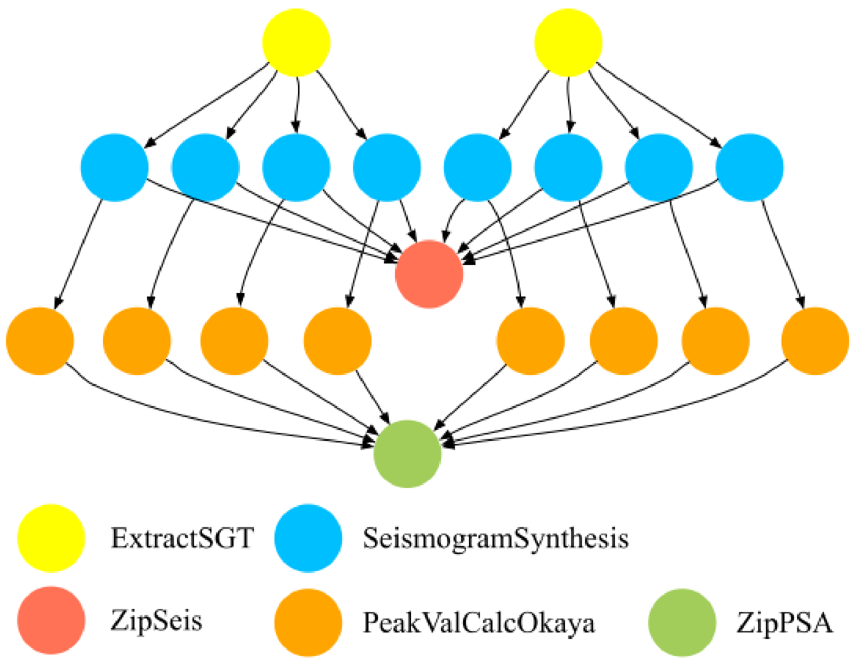

Figure 2.2: Example of a Cybershake workflow [15].

2.3.2 The Montage workflow

Montage was created by the NASA/IPAC Science Archive as an open source toolkit used to generate custom mosaics of the sky with the use of input images in the Flexible Image Transport (FITS) file format and the Montage application can be represented as a workflow [3].

The Montage astronomy workflow's size is dependent on the number of images that are used when constructing the chosen mosaic of the sky [15]. Montage's structure changes to contain the number of inputs if they are increased, which also means an increase of computational jobs [15].

Juve et al. [15] describes the Montage workflow and it's jobs are in detail. In the beginning of the workflow, the mProjectPP jobs re-projects input images. At the next level of the workflow, mDiffFit jobs compute a difference for each pair of coinciding images. Then the mConcatFit job fits the different images using a least squares algorithm. After that, the mBgModel job computes a correction to be applied to each image to get a pleasant global fit. This correction is applied to each image at the next level by the mBackground jobs. Then the mImgTbl job aggregates metadata from the images and a table is created that may be used by other jobs. The mAdd job combines all the re-projected images to generate the final mosaic in FITS format. The mAdd job also generates an area image that can be used for later computation. The mShrink job then reduces the size of the FITS image by averaging blocks of pixels and finally the image is converted to JPEG by the last job mJPEG. The two most computationally intensive jobs in Montage are the mAdd and the mConcatFit jobs where the mAdd is the most intensive [15].

An example of a Montage workflow is shown below.

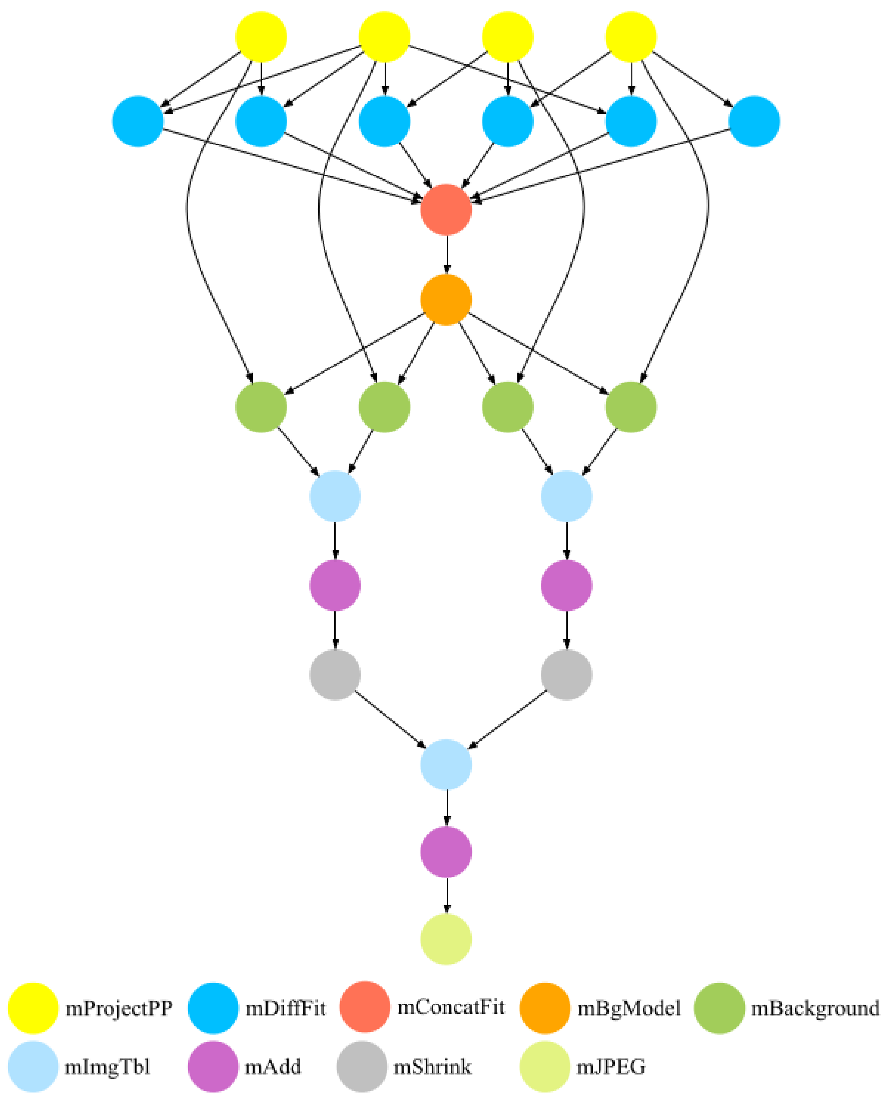

Figure 2.3: Example of a Montage workflow [15].

2.4 PACSA algorithm

The PACSA algorithm is an algorithm for solving the scientific workflow resource scheduling problem in the IaaS cloud environment. The algorithm produces resource scheduling (and provisioning) solutions and in this thesis it is used for gathering machine learning data. The algorithm was developed by my thesis supervisor and colleagues and can be found on GitHub in the optimization-framework repository [16] which includes several algorithms dedicated to finding resource scheduling solutions for scientific workflows. PACSA stands for Parallel Ant Colony Simulated Annealing and it is based on Ant Colony Optimization and Simulated Annealing.

Ant Colony Optimization (ACO) is described as a swarm intelligence population-based metaheuristic [17]. ACO is inspired by the social behaviour of ant colonies which utilizes concepts such as distributed collaboration, self-organization and adaptation in order to solve optimization problems. Parallel Ant Colony Optimization means running the computation on several parallel processing elements. However, not only the speed is improved by splitting up the search, but running it parallel also introduce a new exploration pattern that can be useful to improve the solution quality compared to a sequential implementation [17].

The PACSA algorithm also uses Simulated Annealing for optimization which is based on similarity between solving large optimization problems and simulations of the annealing of solids [18]. According to the paper "Simulated annealing: A tool for operational research", Simulated Annealing is inspired by the physical process of annealing which is the process of finding low energy states of a solid by melting a substance such as glass and then lowering the temperature slowly. While lowering, a long time is spent close to the freezing point of the solid. Within a liquid, particles are arranged at random but the ground state of solids corresponds to the minimum energy setting and has a particular structure. If the cooling process is done too fast, the solid will not reach the ground state. It will instead be frozen into a locally optimal structure such as a glass with defects. In this analogy, the different states of the solid correspond to possible solutions to a optimization problem and the energy of the system correspond to a function to minimize [19].

A paper presenting the PACSA algorithm has been written by my thesis supervisor however it is not yet released online.

2.5 Machine Learning

According to Géron [20], Machine Learning is the science of programming a computer to learn from data. Machine Learning can be used to make a computer learn from data to make complex decisions without being explicitly told what to do [20]. Machine Learning can be divided into two main groups, supervised and unsupervised learning. In supervised learning, the training data that are fed to the algorithm includes the solutions (called labels) to the problem so that it can use them to learn from example. A typical supervised learning task is classification where the objective is to predict a class for an input instance. For example classifying emails as spam or not spam. Another form of supervised machine learning is Regression, where the objective is to predict a target numeric value given a set of features. An example could be predicting the price of a house given features such as number of rooms, square metres, closeness to ocean, etc. In unsupervised learning the training data is unlabeled and the system tries to learn without examples or teaching. An example of unsupervised learning is clustering where the model tries to group data into smaller groups depending on the features [20].

In this thesis focus lies on Regression and specifically Multi-Output Regression because of the nature of the problem, which is predicting how many instances types and which to provision for a workflow with a specified budget.

2.6 Multi-Output Regression

The aim of multi-output regression is to predict multiple numerical output variables at the same time. Multi-output regression is also known as multi-target, multi-response or multi-variate regression in the literature [21]. According to Borchani et al. [21], multi-output regression methods can be divided into two categories, 1) problem transformation methods and 2) algorithm adaptation methods [21].

The problem transformation methods are based on transforming the multi-output regression problem into smaller single-target problems, building a model for each output and in the end combining them all [21]. A drawback of problem transformation methods are that the relationships among the outputs are not captured because the outputs are predicted independently [21]. This may affect the quality of predictions for problems where the output variables are correlated [21].

The algorithm adaptation methods are based on the idea of predicting all the outputs using a single model [21]. The algorithm adaptation methods are able to capture the dependencies and relationships between the output variables [21]. Algorithm adaptation methods ensures better predictive performance if the output variables are correlated [21]. Another advantage with algorithm adaptation methods are that it is easier to interpret [21].

An example of a regression algorithm that does not natively support multi-output regression is the Support Vector Regressor. To accomplish multi-output regression you could use the problem transformation method to combine several single-target Support Vector Regression models into one model capable of multi-output regression [21]. An example of a regression algorithm that natively supports multi-output regression is the Decision Tree [21].

2.7 Machine Learning Algorithms

This section presents the machine learning algorithms used in this project.

2.7.1 Decision Tree Regression

Decision Trees, originally introduced in 1984 by Breiman et al. [22], is a supervised machine learning algorithm that is capable of performing both classification and regression tasks, as well as multi-output tasks [20]. The Decision Tree makes predictions by answering questions about a new instance in a tree-like structure of conditions where the decisions depend on the features of the instance. It starts at the top node, answering the first question about the instances features. The algorithm then descends in the tree until reaching a leaf node where a class is predicted. An advantage of Decision Trees are that they are intuitive and their decisions can be easily understood [20].

There are several training algorithms used to train Decision Trees [23]. In this thesis, the *scikit learn* implementation of Decision Trees [23] is used where the CART algorithm is used for training. The CART algorithm first splits the training set into two subsets using one of the features i and a threshold value t_i. CART searches for the pair of i and t_i that produces the purest subset by weighing their sizes [20]. When the algorithm has split the subset in two, it continues using the same procedure to split the subset of the subset until it reaches a leaf node. The algorithm can also be stopped by specifying a maximum depth or if it can't find a split that will increase information gain or if a minimum information gain is specified and the minimum isn't reached for a split [20].

The main difference when a Decision Tree is used for regression is that it is predicting a value instead of a class in each leaf node [20]. The predictions made by regression trees are simply the average target value of the training instances that are associated to the leaf node that is reached for the prediction [20]. It is the prediction with the lowest Mean Squared Error (MSE) over the instances associated with the leaf node [20].

In this thesis, decision trees are used for multi-output regression and as mentioned earlier, decision trees can be used for multi-output problems. Multi-output regression trees (MRT) can be built following the same steps as in CART with the only difference being that the purity measure of a node is the sum of the squared error over the multi-variate response [21]. Each split tries to minimize the sum of the squared error over all n outputs. In the end, a prediction when reaching a leaf is made by the multi-variate average of its instances, the number of instances at that leaf and its feature values [21]. Multi-output regression trees are better than building a independent tree for each target for two reasons. Multi-output regression trees better identify dependencies between target variables and they are usually smaller than the size you would get when building independent trees [21] [23]. In this thesis Decision Trees are used for multi-output regression using the algorithm adaptation method (predicting all outputs with a single model) [21].

2.7.2 Random Forest Regression

A Random Forest (RF) is an ensemble of Decision Trees and was introduced by Breiman [24] just like the Decision Tree. However, the Random Forest technique came 17 years later in 2001 [24]. Ensemble learning is the technique of combining multiple learning algorithms to increase overall performance [20]. The Random Forest algorithm make us of more randomness when growing it's trees. RF searches for the best feature in a random subset of features instead of searching for the best feature over the whole feature set [20]. This randomness results in more diverse trees that trade higher bias for a lower variance [20]. For prediction, the average of all Decision Trees in the Random Forest are used as the final prediction result [25].

Random Forests are built from Decision Trees and can thereby be used for multi-output regression in the same way as the Decision Tree can.

2.7.3 Support Vector Regression

The Support Vector Machine is a supervised machine learning classifier and the idea behind it was originally constructed by Vladimir Vapnik [26] in Russia in the sixties [27]. The SVM stem from statistical learning theory which is the theory of characterizing properties of learning machines to give them the ability to generalize well on unseen data [27]. The basis of the SVM is to construct linear decision boundaries that try to separate the data into two different classes [28]. To find the decision boundaries, SVM's make use of support vectors which are the data points right on or inside the margin that is used to build the decision boundary [28]. This is illustrated in figure 2.4.

The SVM can however, using a Kernel function, transform input data into higher dimensions [28] [27]. This is useful because data that isn't linearly separable may be separable in a higher dimension. Commonly used kernel functions are the d-th degree polynomial kernel, the sigmoid kernel and the Radial basis function (RBF) kernel [28].

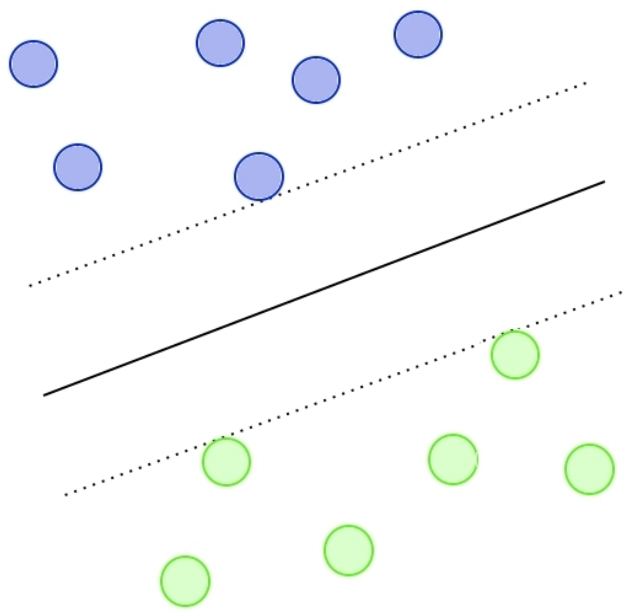

Figure 2.4: Example of a Support Vector Machine, illustrating support vectors on the margin of a hyperplane separating two classes.

The SVM algorithm is versatile and it can handle both linear and nonlinear regression as well [20]. The main goal of the Support Vector Regressor (SVR) is to find a function that has at most ϵ deviation from the labeled targets for all the data points in the training data. It tries to minimize the error while at the same time being as flat as possible. [27]. When doing regression, instead of trying to find a hyperplane that separates two classes as in SVM, the SVR tries to fit as many data points as possible within the margins [20]. The width between the two margins, called the street, is controlled by ϵ [20]. Via the kernel function, the SVR maps a data set X into a high dimensional feature space F and computes a linear regression in F [29].

The general formulation of SVR takes the form

$$f(x) = w \cdot x + b \tag{2.1}$$

where $w \cdot x$ is the dot product of w and x and b is the bias. Flatness means that a *w* as small as possible is wanted [27]. This is ensured by trying to minimize the norm $||w||^2$. Sometimes you may want to allow for some errors and for this slack variables can be introduced. Now this can be written as a convex optimization problem

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}||w^2|| + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ &\text{subject to} \begin{cases} y_i - w \cdot x_i - b \leq \epsilon + \xi_i \\ w \cdot x_i + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \quad \geq 0 \end{cases} \end{aligned} \tag{2.2}$$

The C constant determines a trade-off between flatness of f and the amount to which deviations larger than already defined by ϵ is allowed [27]. However, the complete mathematical and theoretical background for the SVR is extensive and it is difficult to give a precise description of it. The reader is invited to read in the literature to get a deeper insight in [27] and [26].

How a prediction is made by the SVR is described by Smola and Schölkopf. First the input is mapped into feature space by a map ϕ. After that the dot products are computed with the help of training patterns under the map ϕ. This operation corresponds to evaluating kernel functions. Then the dot products are added up and this plus the bias term b yields the final prediction output [27].

The Support Vector Regressor doesn't natively support multi-output regression [21] and in this thesis the problem transformation (making one predictor for each target) method is used to deal with this.

2.7.4 Linear Regression

Based on regression analysis, Linear Regression is a statistical technique investigating the relationship between variables [30]. In LR, the model assumes that the target has a linear relationship with its predictors [25] and generally a linear model predicts by computing a weighted sum of its input features plus the intercept [20]. LR is defined as:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n + E \tag{2.3}$$

When the target is dependent on multiple features (like in this case) it is called multiple linear regression. When it is dependent only on one feature it's called simple linear regression.

The most popular method of estimating the weights is the least squares [25]. In Equation 2.3 the weights are being represented as θ. In Least Squares, the coefficients $\theta = (\theta_0, \theta_1, ..., \theta_n)^T$ are used to minimize the residual sum of squares:

$$RSS(\theta) = \sum_{i=1}^{N} (y_i - f(x_i))^2 \tag{2.4}$$

LR can also handle multi-output regression problems [25]. If we have multiple outputs $Y_1, Y_2, ..., Y_K$ to predict from multiple features $X_0, X_1, X_2, ..., X_p$ then a linear model can be assumed for each output:

$$Y_K = \theta_0 k + \sum_{j=1}^{p} X_j \theta_{jk} + E_k \tag{2.5}$$

$$= f_k(X) + E_k \tag{2.6}$$

The least squares loss function can be generalized to include the errors for multiple targets and the multiple targets least square estimates doesn't effect each other [25]. In this thesis the least squares loss function will be used for LR.

2.7.5 K-Nearest Neighbor Regression

The K-Nearest Neighbor (KNN) methods use the observations closest to the prediction input to the model [25]. KNN is defined as follows:

$$Y(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i \tag{2.7}$$

In equation 2.7, $N_k(x)$ is the neighborhood of x. In other words, the k closest data points (nearest neigbors) to x_i in the training data [25]. The KNN is a simple model and for prediction, it just averages the targets of the closest k neighbors [20]. For classification it means choosing the class that most k neighbors has and for regression it means averaging the target value of the k nearest neighbors [20]. KNN is an instance-based model, which means it doesn't learn from the training data [31]. Instead, in KNN's case it looks up the k nearest neighbors from memory and use their mean for regression/classification [31].

For multi-output regression, the KNN uses the same method as for single-output regression. Instead of just looking at one target it takes the average of all targets for prediction of a new instance [20].

2.7.6 Multilayer Perceptron Regression

The Multilayer Perceptron (MLP) is one of several different architectures of Artificial Neural Networks (ANN) which is an approach to machine learning that can be used for supervised learning [32]. ANNs stem from computational models of biological learning. In other words, models of how we think learning could be happening in our brains [32]. The first artificial neuron was introduced by McCulloch and Pitts in 1943 as an attempt to model brain function [32]. The Perceptron, introduced by Rosenblatt in 1958 became the first model that could learn weights that defined classes given input from each class to train on [32].

The overall goal of MLPs (also called feedforward neural networks) are to approximate a function that maps an input to a category (classification) or a real-valued number (regression) [32]. They are called feedforward networks because the information in the network only flow in a forward direction. Information flows into the function from x (input) through intermediate computations used to define the function and in the end output the prediction y. There are no feedback connections in which outputs connects back to itself. If there are feedback connections the networks are instead called recurrent neural networks. They are called networks because they usually have many different functions combined in a network type structure. A network can, for example, consist of three f_1, f_2, f_3 functions connected in a chain forming $f(x) = f_3(f_2(f_1(x)))$. In such a case, f_1 is the first layer, f_2 the second layer and so on. The amount of layers of the chain is the depth of the neural network and the final layer of a network is called the output layer. The behaviour of the layers between the input layer and the output layer is not specified by the training data because the training data only specifies what should be the result of the output layer. The learning algorithm for a neural network decides how to use the middle layers to produce the desired output [32].

The MLP uses the backpropagation algorithm for training. For each of the training instances, the backpropagation algorithm feeds the instance to the network and computes the output of every neuron in each consecutive layer and this process is called the forward pass. When arriving at the output layer, it measures the network's output error and how much each neuron in the last hidden layer contributed to the error of each output neuron. This process called the reverse pass measures the error gradient across all the connection weights in the network by propagating the error gradient backwards in the network. [20]. This back-propagation efficiently measures the error gradient over all the connection weights in the network. The final step of the back-propagation algorithm is a Gradient Descent step on all weights in the network slightly tweaking the connection weights to reduce the error [20].

In this thesis focus is on multi-output regression and neural networks can handle multi-output problems in a seamless fashion having an output layer consisting of several neurons [25] [33].

2.8 Cross-Validation

Cross-Validation is a widely used method for estimating the prediction error [28]. K-fold Cross-Validation is a version of Cross-Validation that can be used when there isn't enough data to put aside into a validation set. K-fold Cross-Validation uses part of the available data to fit the model and another part of the data to test it. In K-fold Cross-validation, the data is split into **K** equally sized subsets (folds). If for example K = 10, one out of the ten subsets are used as a validation set and the remaining nine are used for training for that fold and the prediction error is recorded. Then the process is repeated ten times so that all subsets gets to be the validation set once. After all the folds have acted as the validation set, the prediction errors from all folds are combined [25].

2.9 Hyperparameter Optimization

Hyperparameter Optimization is the search for finding the optimal parameter setting for a machine learning model to perform as good as possible for a given task [34]. Hyperparameters are the different algorithm-specific parameters available for configuring machine learning algorithms and the parameters can be of different types such as continuous, integer-valued or categorical [34]. There are several different approaches to performing hyperparameter optimization such as gradient search, population-based search and racing algorithms [34]. However, the standard for hyperparameter optimization in machine learning has been grid search which is a method of exploring all possible hyperparameter configurations within a range specified by the user [34].

According to Bergstra and Bengio [35] though, random search is more efficient than grid search when conducting hyperparameter optimization. In the study random search was evaluated against grid search for several machine learning algorithms and several datasets [35]. One conclusion made is that random search is more efficient because not all hyperparameters are equally important [35].

2.10 Mean Absolute Error

The Mean Absolute Error calculation involves summing the absolute values of the errors (the error is calculated as the observed value minus the predicted value) to get the total error and then dividing the total error by the number of samples [36]. The Mean Absolute Error over n samples is defined as

$$MAE(y_p, y_t) = \frac{1}{n}\sum_{i=0}^{n} \mid y_{p_i} - y_{t_i} \mid \tag{2.8}$$

where y_{t_i} is the true value and y_{p_i} is the predicted value [20].

The motivation for using Mean Absolute Error is that it is a common evaluation metric for regression and that it is preferred over the Mean Squared Error which is also one the most common regression evaluation metrics [36].

For the purpose of this thesis the Mean Absolute Error is not an effective evaluation metric due to the fact that the labeled targets are not the only solutions that are considered valid solutions. However, MAE is measured anyway as a pointer to how far away from the labeled targets the regressors are predicting. MAE can for example be used to see if a model starts to predict provisioning solutions that deviate greatly from the labeled targets.

Chapter 3

Related Work

The thesis topic is within the fields of resource provisioning and resource scheduling for scientific workflow execution in the cloud and the use of machine learning within these fields. The resource provisioning problem is a part of the resource scheduling problem and research often presents solutions to both of them combined as a solution to the resource scheduling problem. The previous and related works found during the literature study can be categorized into three categories in relation to the topic of this thesis.

- Current methods of scheduling scientific workflows in the cloud
- Machine Learning used within the area of scientific workflow execution in the cloud
- Machine Learning methods used for Resource Provisioning for cloud providers

3.1 Current methods of scheduling scientific workflows in the cloud

The current research into solving the resource scheduling problem when executing scientific workflows in IaaS clouds can be divided into three main categories [37]. There are methods that use Heuristic Scheduling, Metaheuristic Scheduling and Hybrid Scheduling (combining metaheuristic and heuristic) [37]. The difference between a Heuristic and Metaheuristic method is that a heuristic method is problem specific while a metaheuristic method is a problem-independent technique that can be applied to a variety of problems. A hybrid is when you combine both of them.

When solving the scientific workflow scheduling problem, there can be different goals and objectives. Objectives such as Budget, Deadline, Reliability, Availability, Makespan minimization, Security, Supporting Service Level Agreement and Load Balancing [37].

In this thesis the focus is on provisioning resources for minimization of makespan under a budget constraint. Therefore the related works presented below are focused on addressing minimization of makespan under a budget constraint.

Faragardi et al. [38] proposes an algorithm for scheduling scientific workflows in the cloud that includes both resource provisioning and scheduling. The algorithm named GRP-HEFT (Greedy Resource Provisioning + modified HEFT) is used for minimizing completion time (makespan) of a given workflow subject to a budget constraint for the hourly-based cost model of IaaS clouds. The main difference between regular HEFT and modified-HEFT is that once the execution time goes beyond a full hour, modified-HEFT is able to consider budget violation and takes measures to avoid it. GRP-HEFT is compared against state-of-the-art which includes MOACS, PSO and GA and in the experimental results it outperforms the aforementioned by 11.69%, 19.77% and 13.64% [38].

Chen et al. [39] models the cloud workflow scheduling problem as a multi-objective optimization problem with the two objectives of optimizing cost and execution time. In their paper, a novel multi-objective ant colony system (MOACS) based on co-evolutionary multiple populations for multiple objectives is proposed. Instead of giving only one solution, MOACS can generate a set of scheduling solutions making it so that users can choose a suitable scheduling for their problem instance. Experimental simulations with MOACS are done on five types of scientific workflows (Cybershake, Epigenomics, Inspiral, Montage and Sipht) and are considering the properties of Amazon EC2 cloud platform. The results of the experiments show that MOACS performs better than both some current state-of-the-art multi-objective optimization techniques as well as the constrained optimization techniques [39].

Wang et al. [40] proposes a new algorithm for scheduling budget constrained workflows while minimizing makespan and it is based on the metaheuristic algorithm called Particle Swarm Optimization (PSO). The proposed algorithm is tested on instances from Amazon EC2 and on the scientific workflows LIGO Inspiral and Epigenomics. The performance is compared to the performance of a modified MOLS (Multi-objective list scheduling) algorithm. Results show that when executing small-sized, medium-sized and large-sized workflows the algorithm achieves good results. However, PSO has high time complexity and for handling extra large-sized workflows PSO is too slow and for those MOLS achieve better results [40].

3.2 Machine Learning used within the area of scientific workflow execution in the cloud

Pham, Durillo, and Fahringer [41] uses machine learning in a two-step approach to predict the execution time of workflow tasks in the cloud. In the first stage, a prediction of the runtime parameters are done with the workflow data and VM type as input using a regression method. The second stage uses the predicted runtime parameters from the first stage together with the workflow input data and the VM information as input to a final regression method to predict the execution time. The study was evaluated with the real-world workflows Montage, Wien2k, POV-Ray and Blender. The study achieved the best accuracy using the Random Forest algorithm but also investigated the use of Linear Regression, Multi-Layer Perceptron and Regression Trees [41].

Hilman, Rodriguez, and Buyya [42] in their study investigate predicting task Runtime in Scientific Workflows using Online Incremental Learning. Using both a special type of Recurrent Neural Networks(RNN) called Long Short-Term Memory networks (LSTM) and one type of implementation for K-Nearest Neighbour called IBk algorithm. They report that the research shows promising results and outperforms state-of-the-art in task runtime prediction. The results improve performance up to 29.89 percent compared to state-of-the-art [42].

3.3 Machine Learning methods for Resource Provisioning for cloud providers

There are several works where machine learning is used for the purpose of dynamic resource provisioning from the cloud provider's perspective. These works aim to be able to make automatic scaling decisions by evaluating the future resource usage in real time. These works are only relevant in showing that machine learning has been utilized before to predict resource provisioning but in a different context.

In a paper by Bankole and Ajila [43], machine learning is used for resource provisioning in cloud computing from the cloud providers point of view. Regression techniques are used to predict CPU Utilization and by that future resource usage by the cloud customers. The aim is to predict future resource usage to be ready to handle the customers demands. Support Vector Regression, a Neural Network and Linear Regression were tested as machine learning models and the Support Vector Regressor showed superior performance [43].

Biswas et al. [44] presents a framework and associated algorithms that perform proactive auto-scaling. The framework uses a broker-based system that estimates the number of resources to be used in the system by predicting characteristics of future consumer requests. They investigated Linear Regression and Support Vector Regression and SVR performed slightly better but took 10-15 times longer for calculation [44].

3.4 Motivation for research

During the literature study, no research using machine learning for predicting resources to provision for executing scientific workflows in the cloud were found which is a good motivation for this research.

Chapter 4

Method

This chapter presents the methodology, implementation and evaluation strategy used for this project.

4.1 Environment

Python [45] was the primary language used for implementation, although Java [46] was used for the generation of training data. Python was chosen as the primary language due to the variety of helpful libraries available for machine learning tasks. Libraries used in the project were *scikit-learn* [47], *pandas* [48] and *numpy* [49]. The *scikit-learn* library was used for implementation of the machine learning algorithms, *pandas* was used for data manipulation and *numpy* was used for array manipulation and mathematical operations. Jupyter Notebook [50] was also used in the project. Jupyter Notebook is a interactive notebook capable of running code in cells, showing the output of each cell directly below and storing it as part of the document. Jupyter Notebook was running Python and used mainly for visualizing data and results of running machine learning tasks and evaluating models.

4.2 Dataset generation

The PACSA algorithm was used for generation of datasets. It was used for finding a solution to the resource scheduling problem for scientific workflows in cloud computing. The algorithm takes a problem instance consisting of a workflow and a budget as the input. The algorithm provides a solution instance with the provisioned resources, which tasks(in the workflow) should be run on which instances and in what order the tasks should be run. However for the purpose of this project only the provisioned resources are needed and extracted as labels/targets.

The data generation phase for this project was split into two steps. The first step was to generate workflows with different number of tasks. In this thesis two types of scientific workflows were used, the Cybershake workflow and the Montage workflow. There is a generator tool made by the Pegasus project [51] that takes the type of the workflow and number of tasks of the workflow as input and then generates the workflow as a Directed Acyclic Graph (DAG) represented in XML format. The second step consisted of gathering data and labels by running the PACSA algorithm with the generated workflow DAGs and a specified budget. The result of this step was data consisting of features and labels. The labels are the solution instance generated by the algorithm, specifically which and how many of each instance type provisioned for the workflow. To extract, generate and output the data (features and labels) in a suitable format for later importing into the machine learning process some code and additional helper methods were written in Java in a local branch of the *optimization-framework* [16].

To generate a dataset, data generation was done for a workflow with different number of tasks and different budgets. This is to be able to predict solutions to different sizes of the workflow and with different budgets. The number of tasks used in the workflows were from 100 up to 1000 with steps of 50. The budgets used for each workflow(with different number of tasks) were 0.06, 0.08, 0.10, ... , 20.00. In other words, from 0.06 up to 20.00 with a step size of 0.02. The budget starts at 0.06 because it is the price of the cheapest instance type, *m1.small*. This generation was done both for the Cybershake and the Montage workflow. The number of tasks and budgets used were chosen due to the time constraints of the thesis and if there was more time, there would be no reason not to use more budget values and workflows sizes for training the models. Below in figure 4.1 is a rough illustration of the data generation process for the Cybershake workflow, however this process was repeated for the Montage workflow.

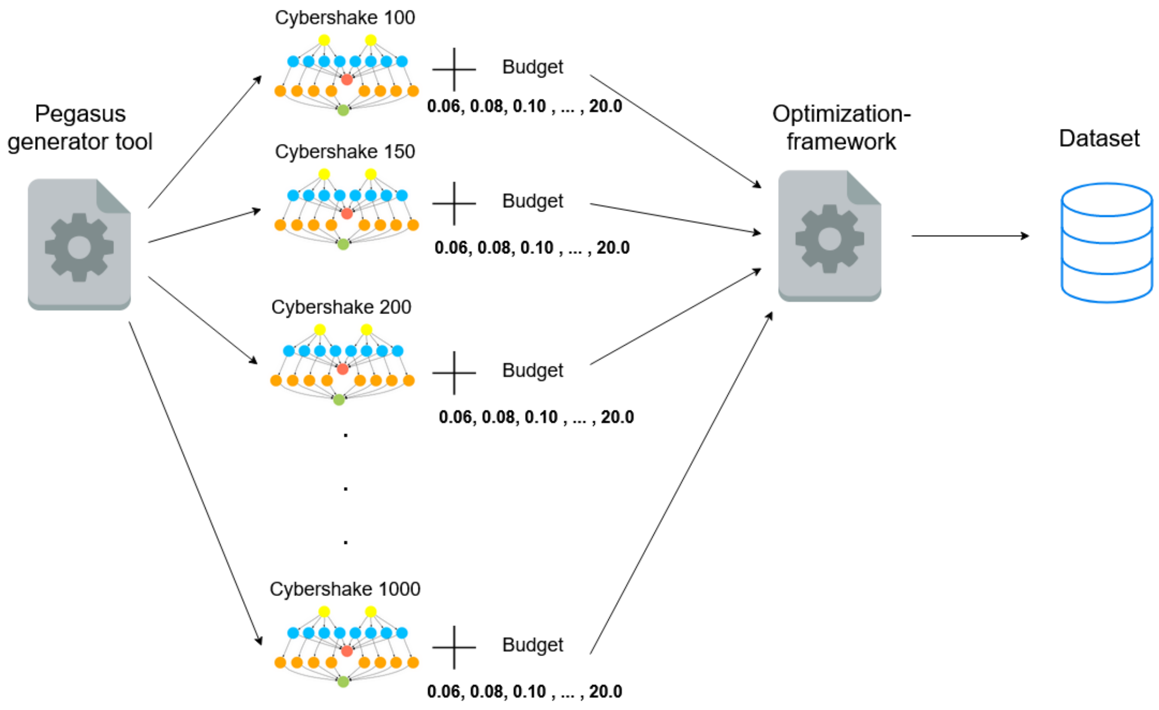

Figure 4.1: Illustration of the data generation process.

4.3 Datasets

The data were divided into two separate datasets, one dataset for the Cybershake workflow and one for the Montage workflow. The datasets contain the same labels/targets however some features differ between the two datasets. The instance types used as labels are the eight instance types presented below in Table 4.1. The reason for using this set of instance types is that it is the configuration setup used in several recent papers such as [52] and [38]. The same configuration setup is also used in the optimization-framework. The instance types can be found at the Amazon website [53]. The labels represent how many of each of those instance types are used.

Table 4.1: Instance types

Instance type	Cost
m1.small	0.06 $
m1.medium	0.12 $
m3.medium	0.113 $
m1.large	0.24 $
m3.large	0.225 $
m1.xlarge	0.48 $
m3.xlarge	0.45 $
m3.2xlarge	0.9 $

The difference in features between the Cybershake and Montage datasets are the features which are the count of how many of each task type the workflow contain. For example, in the Cybershake workflow there is a feature NrOfSeismogramSynthesis (number of SeismogramSynthesis jobs in the workflow) which is a task type in Cybershake. Similarly, in the Montage workflow there is for example a feature NrOfmConcatFit (number of mConcatFit jobs in the workflow) which is a task type in Montage. Other than that, the datasets contain the same features. A list with descriptions of all features and labels in the data are presented in Appendix A.1. Both the Cybershake and Montage datasets each contain 18962 generated samples with different workflow sizes (number of tasks) and budgets.

4.4 Feature Engineering

Feature engineering is the process of selecting the most relevant features, creating and constructing features for model fitting and prediction [20]. In this project some features were constructed that were not present in the data generated using the PACSA algorithm. Features constructed were:

- Features representing how many of each instance type that were affordable under the current budget constraint
- Features representing how many of each task type that are present in the current workflow instance

4.5 Execution

In this section the machine learning implementation and evaluation as well as the motivation behind it are presented.

4.5.1 Motivation & Approach

Due to the definition of good solution quality for this project, a predicted provisioning solution for a problem instance that does not correspond to the target/label value for that instance can still be valid and of good solution quality. This makes standard supervised regression evaluation metrics less effective because they measure how far a prediction is from the target. However in the case of this project, the labeled targets are not the only correct solution and aren't necessarily the best solutions either.

As a result of the lack of previous work investigating the use of machine learning for scientific workflow execution in the cloud, the time constraints of the thesis and the fact that the commonly used metrics for measuring the performance of a model are ineffective for the purpose of this project, it was necessary to make a decision. It was decided that as a first phase start with implementing, testing and evaluating several well-known machine learning algorithms and then follow it by a second phase with focus on the two algorithms performing the best for the task and trying to optimize the performance of these for the purpose of evaluating and comparing them to the PACSA algorithm in terms of both makespan and execution time to be able to answer the research question.

The algorithms chosen for implementation in the first phase were the Decision Tree, Random Forest, Multilayer Perceptron, Linear Regression, K-Nearest Neighbor and Support Vector Regression. One motivation for using these algorithms is that they are commonly used and have a history of performing for supervised learning [54] [55] [20]. The algorithms were also chosen for their capability of handling multi-output regression problems [28] [21]. Another motivation is that (as described in the related works section) several of these algorithms have been used for resource provisioning before but for predicting future cloud resource needs from the cloud providers point of view.

To evaluate and compare different models performance against each other in terms of good solution quality for this project, two evaluation metrics were constructed. These two evaluation metrics, Fitness value and Budget violations percentage are presented in more detail in the Evaluation section 4.6.

The implementation and evaluation of machine learning models were divided into two main phases presented below.

4.5.2 First phase

First both datasets of Cybershake and Montage were divided into a training set with 80 percent of the data and a test set with 20 percent of the data. The 80-20 split is a common ratio to use in machine learning projects [20].

Then implementations were made using the *scikit learn* machine learning library. All algorithms chosen except Support Vector Regression natively support Multi-Output Regression within *scikit learn* [56]. These algorithms utilizes the Algorithm Adaptation Method for Multi-Output Regression. For the Support Vector implementation, the Problem Transformation Method for Multi-Output Regression is used. With this approach, several single target regressors are combined using the MultiOutputRegressor class in *scikit learn* [57] that fits one regressor per target to accomplish Multi-Output Regression.

In the first phase, the machine learning models were trained using the standard settings as defined by *scikit learn* for each model. The models were then evaluated for both workflows using 10-fold Cross-validation with shuffling on the training set. The models were also evaluated for both workflows on the test set. The evaluation metrics used for this step to compare model performance were Fitness value (discussed in Section 4.6.1), Budget violations percentage (discussed in Section 4.6.2) and Mean Absolute Error.

4.5.3 Second phase

In the second phase, the two best performing models with regards to Fitness value were chosen for further investigation and for conducting hyperparameter optimization using Random Search to improve performance. Hyperparameter optimization was done to find the optimal parameter setting for the two models and it was conducted with RandomSearchCV (which utilizes Cross-validation for hyperparameter tuning) [58] on the training set with the Fitness value as the evaluation metric chosen for optimization. Random Search was chosen over Grid Search for two reasons. Random Search is reported as more efficient than grid search [35]. The second reason was a long computation time computing the Fitness value evaluation metric. Random Search was used for conducting Hyperparameter Optimization and the chosen number of iterations were 60. Random Search was conducted with 60 iterations for both models and for both the Cybershake and the Montage workflow. According to Zheng [59], 60 iterations is a good number for Random Search to find a close-to-optimal solution with high probability.

After finding the optimal parameters, the models were trained with the optimal parameters found and were measured in terms of makespan and execution time in comparison to the PACSA algorithm in order to answer the research question. The makespan that the models achieved on the test set were compared to what the PACSA algorithm achieved for the test set. The amount of budget violations were also taken into consideration and accounted for. The execution time in comparison to the PACSA algorithm is measured by the time it takes to come up with a solution for different sizes of the workflows. Execution time was measured for small workflows (100 tasks) and large workflows (1000 tasks) and it was measured for both the Cybershake and the Montage workflow.

4.6 Evaluation

In this section the evaluation methods used in this project are presented. The Fitness value and Budget violations percentage metrics were constructed specifically for the purpose of this project and are used to compare the performance of the machine learning models. The primary metric is Fitness value which takes both makespan and budget violations into consideration. Makespan and Execution time are used in order to compare the two best performing models to the PACSA algorithm to then be able to answer the research question. Execution time in this case means the time it takes for predicting a solution instance given a problem instance.

4.6.1 Fitness value

The Fitness value is designed to be able to compare machine learning models to each other both in terms of makespan and budget violations. The objective is to minimize the makespan while still respecting the budget constraint. The function for calculating the Fitness value is defined as:

$$\text{Fitness(m)} = \min\left(\sum_{i=1}^{n} m_i\right) \tag{4.1}$$

where m is the vector of all makespans associated to the predicted provisioning for the test set, m_i is the makespan of the ith prediction and n is the number of predictions made for the test set.

In addition to this, the makespan are penalized and multiplied by α for the instances where the predicted provisioning violates the budget constraint. The makespan is also penalized and multiplied by α if the model predicts that the problem instance cannot be solved due to low budget when in fact the budget is high enough for it to be solved. When that happens, the mean makespan of all the makespans is used and multiplied by α.

$$m_i = \begin{cases} m_i * \alpha, & \text{if prediction violates the budget} \\ mean(m) * \alpha, & \text{if predicted unsolvable when it is solvable} \\ m_i, & \text{otherwise} \end{cases} \tag{4.2}$$

For the calculations of Fitness value in this project, the number 1000 was chosen as α. The choice of 1000 as the α penalty parameter was to make any arbitrary infeasible solution's Fitness value to always be worse than the worst possible feasible solution. This was to make sure that feasible solutions are always prioritized because it is of big importance that the model does not suggest provisioning solutions that violates the budget constraint.

4.6.2 Budget violations percentage

The Budget violations percentage is a measure of the percentage of budget violations made by a model in a series of predictions. A budget violation can be a violation of the budget constraint, meaning provisioning instances where the total cost of the instances is over the budget. A budget violation can also be that the model predicts that a problem instance is unsolvable within the current budget when it is indeed solvable. In other words, Budget violations percentage is the total amount of budget violations made over the series of predictions divided by the total amount of predictions in the series.

4.6.3 Makespan

For this project, makespan is how long it takes to run the scientific workflow distributed on a set of provisioned instances, in other words the time it takes from the first instance is started until the last instance is closed and the workflow is done. The thesis objective is to be able to make good predictions for resource provisioning, however a predicted provisioning does not include a makespan. To be able to compare two different resource provisioning solutions for a workflow in terms of makespan first the workflow has to be scheduled on top of the provisioned instance set to get a makespan. Therefore in order to evaluate the machine learning predicted provisioning in terms of makespan, a scheduler is employed to schedule the tasks on the provisioned resources. For scheduling, an implementation of the Heterogeneous Earliest Finish Time (HEFT) algorithm that is available in *optimization framework* [16] is used. This HEFT implementation is done by my thesis supervisor and colleagues and is presented in [38]. The Heterogeneous Earliest Finish Time algorithm is a scheduling algorithm for a bounded number of heterogeneous processors with the goal of minimizing execution time [4]. HEFT has two major phases. First, a task priority phase for finding the priority of all tasks for the job. Second, a processor selection phase for placing the tasks in the order of their priority and scheduling each task on its best processor [4].

4.6.4 Execution time

The execution time is measured as how long it takes to compute a complete resource scheduling solution including both resource provisioning and task scheduling given a problem instance. For this project, execution time means how long it takes to predict a machine learning provisioning, schedule it using HEFT and then receive a complete solution containing which resources to provision and how to schedule them on the provisioned instances. Execution time is measured by letting a model run on a problem instance representing different sizes of a scientific workflow and measure the time it takes to come up with a solution.

Chapter 5

Results

In this chapter, the results of this project are presented. As the implementation of this project consists of two main phases, the results are divided into two sections presenting the results of the associated phases. The first phase contains the results of evaluating several machine learning algorithms and their learnability for the purpose of this project. In the second phase, hyperparameter optimization was performed on the two algorithms that performed the best in the first phase. Then the two algorithms were trained with their optimal hyperparameter setting and compared against the PACSA algorithm in order to answer the research question.

5.1 First phase results

In the first phase, the machine learning algorithms Decision Tree, Random Forest, Support Vector, Linear Regression, K-Nearest Neighbor and Multilayer Perceptron were trained using their standard settings defined by *scikit learn* [47]. The algorithms were then evaluated using the constructed evaluation metrics Fitness value and Budget violations percentage. Mean Absolute Error was recorded as well to see how far from the targets the models predicted. The algorithms were evaluated using 10-fold Cross-validation. The results presented are the average performance for Fitness value, Budget violations percentage and Mean Absolute Error over the 10 folds. The results for the first phase are divided into two sections, one presenting the results for the Cybershake workflow and one presenting the results for the Montage workflow.

5.1.1 Cybershake results

In the Cybershake evaluation there were quite large differences in the performance of the machine learning models with regards to the Fitness value and the Budget violation percentage. The Multilayer Perceptron, Nearest Neighbor and Linear Regression models performed significantly worse than the Random Forest, Decision Tree and Support Vector models. The best performing model was the Decision Tree with an average Fitness value of 11511 and Budget violation percentage of 1.054 percent and the second best performing model was the Support Vector with an average Fitness value of 15441 and Budget violation percentage of 1.516 percent. The results of the evaluation for the Cybershake workflow can be seen below in 5.1.

Table 5.1: Cybershake evaluation results.

Model	**Average Fitness value**	**Budget violations**	**Mean Absolute Error**
Linear Regression	105 797	24.365 %	0.19019
Decision Tree	11 511	1.054 %	0.17294
Random Forest	22 696	5.596 %	0.15665
Support Vector	15 441	1.516 %	0.21080
Nearest Neighbor	120 309	24.602 %	0.22563
Multilayer Perceptron	109 218	28.169 %	0.19250

5.1.2 Montage results

Similar to the results of the Cybershake evaluation, there were quite large differences in performance of the machine learning models. However, the models showed better overall performance on the Montage workflow, both in regards to Fitness value and Budget violations. Like for the Cybershake workflow, Multilayer Perceptron, Linear Regression and Nearest Neighbors were the three worst performing models and Random Forest, Decision Tree and Support Vector the three best performing. However, the Support Vector model did perform notably worse for the Montage workflow. The best performing model was the Decision Tree with an average Fitness value of 6221 and a Budget violation percentage of 1.905 percent and the second best performing model (in contrast to Cybershake) was the Random Forest with an average Fitness value of 9368 and a Budget violation percentage of 6.032 percent. The results of the evaluation for the Montage workflow can be seen below in 5.2.

Table 5.2: Montage evaluation results.

Model	Average Fitness value	Budget violations	Mean Absolute Error
Linear Regression	37 462	11.035 %	0.22698
Decision Tree	6 221	1.905 %	0.16585
Random Forest	9 368	6.032 %	0.15204
Support Vector	19 674	6.256 %	0.22872
Nearest Neighbor	95 414	30.166 %	0.28031
Multilayer Perceptron	30 542	10.462 %	0.23508

5.1.3 Conclusion of first phase

Looking at the results from the evaluation of the first phase for both the Cybershake and Montage workflow, the Decision Tree model performed the best for both workflows and was chosen for further investigation in the second phase. The second model chosen for further investigation was the Support Vector.

The motivation is that it performed second best on the Cybershake workflow (significantly better than Random Forest) and almost as good as the Random Forest model (which was the second best) for the Montage workflow.

Noteworthy is that for both workflows, it is not the models with the lowest Mean Absolute Error who perform the best. This means that it is not the models that predicts the closest to the labels in the training data that perform the best in terms of makespan and budget violation.

5.2 Second phase results

In the second phase, the two best performing models from the first phase were optimized with hyperparameter optimization to then be trained with the optimal hyperparameters in order to evaluate the models performance and compare it to the PACSA algorithm to answer the research question. The second phase results are divided into three sections for hyperparameter optimization, makespan evaluation and execution time evaluation. The goal is to answer the research question with the evaluation of makespan and execution time. Makespan performance is evaluated in comparison to the PACSA algorithm for the test set while taking budget violations into consideration. Execution time is evaluated in comparison to the PACSA algorithm by measuring the time to find solutions for small workflows (100 tasks) and large workflows (1000 tasks) for both the Cybershake workflow and the Montage workflow.

5.2.1 Hyperparameter Optimization

Hyperparameter Optimization was done with Random search with 60 iterations for both models (Decision Tree & Support Vector) and for both workflows (Cybershake & Montage). The hyperparameter optimization chapter is divided into two sections, one for each model.

Decision Tree

The parameters chosen for optimization of the Decision Tree model was *max_depth*, *max_features*, *min_samples_split*, and *min_samples_leaf*. The same parameters were used for both workflows. The *max_depth* parameters controls the maximum depth of the Decision Tree, which by default is *None* which means it can grow as big as it wants [20]. Parameters *min_samples_split* and *min_samples_leaf* controls how many samples are needed for a split or for a leaf. The *max_features* parameter controls how many features to consider for splitting at each node [20]. Random Search found that the optimal hyperparameters were the same for both workflows. The hyperparameter values used and the optimal parameter values found by the Random Search for Decision Tree can be seen below in table 5.3.

Table 5.3: Hyperparameters for Decision Tree

Hyperparameter	Possible values	Found optimal (both workflows)
min_samples_leaf	1, 2, 4, 8	1
min_samples_split	2, 5, 10, 20	2
max_features	'sqrt', None	None
max_depth	None, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30	None

Support Vector

The parameters chosen for optimization of the Support Vector Regression model was *kernel*, *epsilon*, *C* and *degree*. The *kernel* parameter controls which kernel function the SVR is using. The *degree* parameter is only used when the *kernel* parameter is set to 'poly' (polynomial kernel) and it is used to control the polynomial degree for the kernel function. As mentioned in the theory chapter, the width between the two margins (the street) is controlled by the *epsilon* parameter [20] and the *C* parameter is the penalty parameter for the error term. Random Search found that the optimal hyperparameters were the same for both workflows. The hyperparameter values used and the optimal parameter values found by the Random Search for the Support Vector Regressor can be seen below in table 5.3.

Table 5.4: Hyperparameters for Support Vector

Hyperparameter	Possible values	Found optimal (both workflows)
kernel	'rbf', 'linear', 'poly'	'linear'
epsilon	0.0, 0.01, 0.1, 0.5, 1	0.0
C	0.1, 0.5, 1, 10, 50, 100	1
degree	2, 3	3

5.2.2 Makespan Comparison

The makespan is measured to be able to answer the research question which involves investigating if an machine learning model can give solutions of similar quality in terms of makespan as an state-of-the-art algorithm (PACSA) to the resource provisioning for scientific workflow problem in cloud computing. The resource provisioning solutions provided by the machine learning model is scheduled by utilizing an independent HEFT scheduler to get a makespan to be compared to the makespan of solutions provided by the PACSA algorithm. The comparison is done on the test set which contains 3792 problem instances for both workflows and is presented in table 5.5 (Cybershake) and table 5.6 (Montage). To do a fair performance comparison against the PACSA algorithm in terms of makespan, the solution instances where the machine learning model violates the budget are not considered. Instead there is a column showing how many percent of the solutions predicted by the model violates the budget constraint. Metrics presented in the tables are *Budget violations*, *Average percentage budget violated by*, *Defensive budget violations*, *Average PACSA makespan / ML makespan*, *Identical solutions instances*, *Worse solutions instances* and *Better solutions instances*, *Average improvement* and *Instances solved PACSA failed to solve*. The *budget violations* is the percentage of the solutions predicted by the machine learning model that is violating the budget constraint on the test set. The *Average percentage budget violated by* is the average amount of how much that the budget is violated by. The *Defensive budget violations* metric measures how many instances that the model predicted as unsolvable when it was in fact solvable under the current budget constraint. The metric *Average PACSA makespan / ML makespan* represents, in percent and in average, how well the machine learning model performed in terms of makespan in comparison to the PACSA algorithm on the solution instances that didn't violate the budget constraint. There are three metrics *Identical solutions instances*, *Worse solutions instances* and *Better solutions instances* that measures in percent how many of the solution instances predicted for the test set got an identical, worse or better makespan than the PACSA algorithm. The *Average improvement* measures in percent and in average how big of an improvement the better solutions are. Finally, the metric *Instances solved that PACSA failed to solve* (which is self-explanatory) measures in percent how many problem instances that the models managed to solve which the PACSA algorithm couldn't solve. The makespan comparison results are divided into two sections, one for each workflow.

Cybershake makespan comparison results

For the Cybershake workflow, the Decision Tree Regressor shows better average performance in terms of makespan, being able to predict solutions that were scheduled and gave a makespan that was 1.80 percent slower than the PACSA algorithm while the Support Vector Regressor was 8.14 percent slower. In the calculation of average makespan, the PACSA makespans differ because the instances where the budgets were violated by the models were not considered and could be different instances. In other words, the PACSA makespan for the Decision Tree Regressor is lower in this case because the budget was violated for instances which required a high makespan, making the average for both PACSA and ML makespan lower. In terms of budget violation, the Support Vector Regressor performed significantly better having only 0.079 percent budget violations while the Decision Tree Regressor had 1.212 percent budget violations. The Decision Tree Regressor also manages to predict a higher amount provisioning solutions (12.81 %) that gives a better makespan compared to the PACSA algorithm than the Support Vector Regressor (10.30 %). However, the better solutions instances that the Support Vector predicts have a significantly higher Average improvement with 4.75 % compared to the 0.88 % that the Decision Tree Regressor performs. The Support Vector Regressor is also able to solve a substantially higher amount (87) of instances that PACSA failed to solve than the Decision Tree Regressor (3). These solved problem instances can be seen in Appendix B. The results of the Cybershake makespan comparisons can be seen below in table 5.5.

Table 5.5: Cybershake makespan comparison results

	Decision Tree Regressor	**Support Vector Regressor**
Budget violations	1.212 %	0.079 %
Average amount budget violated by	0.0178 $	0 $
Defensive budget violations	0.079 %	0.079 %
Average $\frac{\text{ML makespan}}{\text{PACSA makespan}}$	488.28 / 479.62 = 1.80 %	530.21 / 490.27 = 8.14 %
Better solution instances	12.81 %	10.30 %
Identical solution instances	10.79 %	6.38 %
Worse solution instances	76.4 %	83.32 %
Average improvement	0.88 %	4.75 %
Instances solved PACSA failed to solve	0.00079 % (3 instances)	0.02294 % (87 instances)

Montage makespan comparison results

The results of the makespan comparions for the Montage workflow shows that the Decision Tree Regressor performs the best in terms of makespan, being able to predict provisioning solutions that after scheduling gives a makespan that is on average 0.12 percent slower than the makespan provided by the PACSA algorithm. The Support Vector Regressor was able to predict for a makespan 4.75 percent slower than the PACSA algorithm, however it performed better in terms of not violating the budget constraint with only 0.132 percent of predicted solutions violating the budget constraint compared to 1.503 percent for the Decision Tree Regressor. In the calculation of average makespan, the PACSA makespans differ a little because the instances where the budgets were violated by the models were not considered and could be different instances. Similar to what the evaluation of Cybershake showed, the

Decision Tree Regressor manages to produce a higher amount of provisioning solutions (20.53 %) that get a better makespan compared to the PACSA algorithm than the Support Vector Regressor (19.06 %). The Support Vector Regressor performs better in terms of Average improvement of the better solution instances with 2.61 % compared to the 1.80 % that the Decision Tree Regressor showed. The Support Vector Regressor is also able to solve 37 instances that the PACSA algorithm failed to solve while the Decision Tree Regressor wasn't able to solve any instance that the PACSA algorithm failed to. These solved problem instances can be seen in Appendix B. The results of the Montage makespan comparisons can be seen below in table 5.6.

Table 5.6: Montage makespan comparison results

	Decision Tree Regressor	**Support Vector Regressor**
Budget violations	1.503 %	0.132 %
Average amount budget violated by	0.0194 $	0 $
Defensive budget violations	0 %	0.132 %
Average $\frac{\text{ML makespan}}{\text{PACSA makespan}}$	323.83 / 323.44 = 0.12 %	339.14 / 323.74 = 4.75 %
Better solution instances	20.53 %	19.06 %
Identical solution instances	46.37 %	32.06 %
Worse solution instances	33.1 %	48.88 %
Average improvement	1.80 %	2.61 %
Instances solved PACSA failed to solve	0 %	0.00975 % (37 instances)

5.2.3 Execution time comparison

The execution time is evaluated to answer the research question and in particular if a machine learning model can find a resource scheduling solution in a significantly shorter computation time than the PACSA algorithm. The execution time is measured as the time it takes to find a solution to the resource scheduling problem given a workflow as problem instance. For this project, it is measured how long it takes for the machine learning models to come up with a solution and the HEFT scheduler to find a scheduling for it. That is compared to how long it takes for the PACSA algorithm to come up with a solution to the same workflow. The execution time is measured for both workflows for large sizes (1000 tasks/nodes) and small sizes (100 tasks/nodes). The PACSA algorithm and the two machine learning models are evaluated for the budgets available in the training set for both workflows and for the small and the large size and then averaged. In other words, PACSA and the models are run on Cybershake 100, 1000 and Montage 100, 1000 for all the budgets from 0.06 to 20.00 with a step size of 0.02 and the execution time is then averaged.

Execution time comparison results

For the smaller sizes of the workflows, the machine learning models takes about the same time to come up with a solution. On average, both models take 0.005 seconds to come up with a solution both for the Cybershake and the Montage workflow while the PACSA algorithm takes approximately 2.3 seconds for the Cybershake workflow and 1.7 seconds for the Montage workflow. For the larger sizes of the workflows, the average execution time of the machine learning models ranges from 0.016 seconds to 0.021 seconds while for the PACSA algorithm the average execution time for Cybershake is 57.138 seconds and 44.215 seconds for Montage. The execution time comparion results are presented below in table 5.7.

Table 5.7: Execution time comparisons results

	PACSA	**Decision Tree Regressor**	**Support Vector Regressor**
Cybershake 100	2.321 sec	0.005 sec	0.005 sec
Cybershake 1000	57.138 sec	0.017 sec	0.016 sec
Montage 100	1.687 sec	0.005 sec	0.005 sec
Montage 1000	44.215 sec	0.020 sec	0.021 sec

Chapter 6

Discussion

This chapter presents a discussion of the results obtained by the work conducted for this thesis. The discussion serves to analyze the results and to answer the research question.

The goal of the thesis was to investigate the use of machine learning for predicting resource provisioning solutions to the problem of executing scientific workflows in the cloud while still maintaining similar solution quality as a state-of-the-art algorithm. To investigate this, the implementation was divided into two phases. In the first phase, the six machine learning algorithms Decision Tree, Random Forest, Multilayer Perceptron, Linear Regression, K-Nearest Neighbor and Support Vector Regression were evaluated and measured according to the metrics of the constructed metrics Fitness value and Budget violations. The two best performing machine learning models from the first phase were then chosen for further training and evaluation. The Decision Tree Regressor and The Support Vector Regressor performed the best and were optimized with hyperparameter optimization. Then the models were trained with their optimal parameters and were used to predict resource provisioning solutions which were then scheduled by an independent HEFT scheduler to then be compared to the PACSA algorithm in terms of makespan and execution time for the workflows Cybershake and Montage in order to answer the research question.

6.1 Result analysis

6.1.1 First evaluation phase

The first phase results show that the best performing model in terms of Fitness value and budget violations is the Decision Tree Regressor (DTR) for both the Cybershake and the Montage workflow. The Support Vector Regressor (SVR) and the Random Forest models also performed quite well. These three performed significantly better than the remaining three models of Linear Regression, Multilayer Perceptron and Nearest Neighbor. The performance difference between the models were high with the DTR showing the lowest average Fitness value which was more than 10 times smaller for the Cybershake workflow and more than 15 times smaller for the Montage workflow than the highest average Fitness value showed by the Nearest Neighbor model. The difference in performance between the models are even bigger for the budget violation metric. The Decision Tree had the lowest budget violations percentage for both workflows while the Multilayer Perceptron had the highest for the Cybershake workflow with a percentage of budget violations that was more than 25 times higher and for Montage the Nearest Neighbor had the highest budget violations percentage that was more than 15 times higher than for DTR. Taking a look at the results from the first phase, the common regression metric Mean Absolute Error (MAE) doesn't quite capture the definition of good solution quality of this thesis. This is visualized by the fact that it isn't the model with the lowest error that performs the best in terms of Fitness value and budget violations. Due to the time constraints of the thesis, the decision was made to focus on optimizing the two best performing models of the first phase in the second phase. There is a possibility that if all the six machine learning models were evaluated further and optimized with hyperparameter optimization it could be found that another model were able to perform better than the DTR or the SVR model in terms of makespan and budget violations percentage. An example of how much the performance can change with optimal hyperparameters is the improvement that the SVR made with optimized parameters going from 1.516 % to 0.079 % on Cybershake and from 6.256 % to 0.132 % in terms of budget violations.

6.1.2 Second evaluation phase

The second phase yielded many interesting results. The results of comparing DTR and SVR with the PACSA algorithm in terms of makespan and execution time were interesting. Depending on what is considered the primary goal for the user looking to execute a scientific workflow using the machine learning models for resource provisioning, the argument could be made that both the models performed the best. If the user's primary goal is a scheduling solution that is as fast as possible in terms of makespan, using DTR is preferable due to it being the closest to performing as well as the PACSA algorithm on average being only 1.80 % slower on the Cybershake workflow and 0.12 % slower on the Montage workflow. If it is more important for the user that the scheduling solution never breaks the budget constraint, using the SVR is preferable due to the low budget violation percentage that were 0.079 % for the Cybershake workflow and 0.132 % for the Montage workflow.

The results for the second phase also showed that both the DTR and SVR predicted provisioning solutions scheduled with the HEFT scheduler were able to outperform the PACSA algorithm for a significant amount of problem instances. The DTR model were able to outperform the PACSA algorithm a higher amount of times for both workflows (12.81 % and 20.53 % compared to SVR's 10.30 % and 19.06 %) while the SVR model showed a higher average improvement(4.75 % and 2.61 % compared to DTR's 0.88 % and 1.80 %) on both workflows. It seems that the SVR model is more irregular/inconsistent with it's predictions in terms of makespan due to that fact that it is on average slower than the DTR model compared to the PACSA algorithm but at the same time having a significantly higher average improvement on the instances where it outperforms the PACSA algorithm. Another interesting result is that both the DTR and the SVR models were able to solve instances that the PACSA algorithm couldn't solve. The SVR model were able to solve 87 problem instances that the PACSA algorithm couldn't solve on the Cybershake workflow and 37 problem instances on the Montage workflow. However, the DTR algorithm was less successful in this regard and only managed to solve 3 problem instances (all from Cybershake) that the PACSA algorithm couldn't. The problem instances that the models were able to solve that the PACSA couldn't were all problem instances where the budget was low (between 0 and 2 $) and the number of tasks quite large (between 400 and 1000 tasks).

The results from the evaluation of execution time of finding a scheduling solution showed that both the DTR and the SVR model were able to find a solution significantly faster than the PACSA algorithm. The difference in execution time for finding a solution for both small and large workflows were negligible with only 1 ms difference between the models for the large workflows and none for the small workflows. For small workflows (100 tasks) the models showed an execution time of 0.005 seconds (for both workflows) which was more than

400 times as fast the PACSA algorithm for the Cybershake workflow (PACSA had 2.321 seconds) and more than 300 times as fast for the Montage workflow (PACSA had 1.687 seconds). For the large Cybershake workflows (1000 tasks) the models showed an execution time (0.016/0.017 seconds) that were more than 3000 times as fast as the PACSA algorithmn which had an execution time of 57.138 seconds. For the large Montage workflows (1000 tasks) the models showed an execution time (0.020/0.021 seconds) that were more than 2000 times as fast as the PACSA algorithm which showed an execution time of 44.215 seconds.

The hyperparameter optimization took 20 days to complete (5 days per workflow and per 60 iterations), however it might have been faster with a better computing setup. If you had more time to do a Random Search with more iterations (than the 60 performed in this thesis) or Grid Search over a larger search-space for the DTR and the SVR models it is possible that you could find better parameter settings than found in the thesis. Larger datasets for both workflows could potentially help in being able to train a model with even higher performance.

6.2 Critical Evaluation

This thesis is limited to investigating machine learning as an alternative resource provisioning strategy for eight instance types provided by Amazon EC2 as well as limited to considering the scientific workflows Cybershake and Montage. The models evaluated in this thesis are therefore specialized for these configurations and would be of no use for different instance types or workflows. The models are also limited to considering budgets within the ranges of 0 to 20 dollars and workflows (Cybershake & Montage) with the number of tasks being between 100 and 1000. Exposed to problem instances outside of these boundaries there are no guarantees for how the models would behave but they would probably generalize better the closer to these boundaries the problem instances are.

The comparison of the models is limited to only using the machine learning model implementations available in the *sci-kit learn* [47] machine learning library. In the first phase the evaluations are limited to only considering the six machine learning models with their standard parameters settings as defined in *sci-kit learn* [47] and as mentioned earlier it is possible that evaluating all models with more parameter settings or performing hyperparameter optimization for all may lead to different results. Doing a more extensive Random Search, Grid Search or other type of hyperparameter optimization technique could potentially find even better parameter settings and improve the performance of the DTR and SVR models in terms of makespan and budget violation percentage.

The size of the data sets (the considered budgets and sizes of the workflows) were decided completely due to the time constraints of the thesis. It took more than a month gathering the data for the data sets running the PACSA algorithm for different workflow sizes with a corresponding budget. With bigger data sets containing more workflow sizes and budget values that could be used to training the models, the models would probably be able to perform better than in this thesis. The features in the data sets were either engineered or available as properties of the workflows. Due to the time constraints and the amount of time it took to generate data, not that much time was spent in the feature engineering step and it is possible that there are features that could be engineered who could have had a more positive impact on the learning of the machine learning models.

6.3 Validity of Results

The PACSA algorithm (which is used for generating the data set and comparing the machine learning models to) is open to the public in the *optimization framework* [16] on GitHub. The workflows Cybershake and Montage used in this thesis were generated with a tool created by the Pegasus project [51] which is also open to the public. The machine learning algorithms used in this project are found in *scikit learn* [47] which is a free open source machine learning library. This helps the validity of the results because all experiments can be reproduced with the same tools as used in this thesis.

6.3.1 Construct validity

To measure performance of a resource scheduling solution, makespan which measures of the execution time of running a scientific workflow on the instances distributed in the cloud is used. The makespan is a valid measurement because a worse makespan is equivalent to worse performance which is what we want to be able to analyze. The execution time of finding a provisioning and scheduling it with the HEFT scheduler is a valid measurement since we want to know the time it takes to find a complete scheduling solution for a scientific workflow.

6.3.2 Internal validity

All experiments and evaluations were run on the exact same setup which contributes to internal validity. Execution time evaluation experiments were executed several times in order to get an average instead of doing single runs that could make results uncertain.

6.4 Sustainability and Ethical Aspects

When conducting machine learning research it is important to handle the data set with care, especially if the data set contains sensitive information. However, the data sets for this project contained no sensitive information, in fact the data set were generated through running algorithm and the data sets contained properties of scientific workflows that are publicly available online.

The thesis work could be considered working towards ecological sustainability. The end goal of the thesis investigation is to offer a reliable and faster resource scheduling mechanism for scientific workflows. Having a faster solution would reduce the amount of heavy calculations done finding a suitable resource scheduling and in the end contribute to a lower electric energy consumption. Trying to lower the makespan of execution of scientific workflows will in the end also contribute to a lower electric energy consumption by the clouds executing the workflows.

The goal of solutions that are automated (like machine learning solutions) is to save time from humans doing repetitive tasks and subsequently having an economically positive impact. The negative social impact that automated solutions can have is that it could remove jobs that were previously performed by humans. However, finding a solution to the scheduling of the execution of scientific workflows is not a full time job but rather a part of the job for some scientists. Having a quicker way of finding a scheduling solution will hopefully only help scientists in having more time to focus on other tasks.

Chapter 7

Conclusions

This thesis investigated machine learning as an alternative way of finding resource provisioning solutions for the problem of scientific workflow execution in the cloud. Furthermore, the objective consisted of investigating and evaluating if a trained machine learning model could give solutions with solution quality close to the solution quality of a state-of-the-art algorithm (PACSA) but in a significantly shorter time. Good solution quality for the purpose of this thesis were defined as solutions that are feasible, doesn't violate the budget constraint and has a makespan close to what the PACSA algorithm reports for the problem instance. The machine learning predicted provisioning solutions were scheduled with an independent scheduler based on the HEFT algorithm to get a makespan to compare with the PACSA algorithm.

The study has shown that it is possible to train a machine learning model with a data set containing solution instances from the PACSA algorithm as labels and use it to predict resource provisioning solutions of good quality. This is shown by the fact that the Decision Tree Regressor are able to be only 1.80 % (Cybershake) and 0.12 % (Montage) slower than the PACSA algorithm in terms of makespan while imposing only 1.212 % (Cybershake) and 1.503 % (Montage) budget violations. It is also shown by the Support Vector Regressor being 8.14 % (Cybershake) and 4.75 % (Montage) slower than the PACSA algorithm in terms of makespan while imposing 0.079 % (Cybershake) and 0.132 % (Montage) budget violations. Depending on what is most important for a user executing scientific workflows, to be as fast as possible in terms of makespan or to be sure that the budget constraint is not violated, the Decision Tree Regressor model or the Support Vector Regressor model is the better choice.

The study has also shown that the machine learning models can predict solutions of a good solution quality in a significantly shorter time than the PACSA algorithm. This is shown by evaluating the execution time of finding a solution to a problem instance for both small and large Cybershake and Montage workflows. The Decision Tree Regressor and the Support Vector Regressor paired with the HEFT scheduler are able to find solutions to small workflows (100 tasks) of both Cybershake and Montage in 0.005 seconds compared to PACSA's execution time of 2.321 seconds (Cybershake) and 1.687 seconds (Montage). For large workflows (1000 tasks), the models showed an execution time of between 0.016-0.017 seconds for Cybershake and between 0.020-0.021 seconds for Montage compared to the PACSA algorithm's execution times of 57.138 seconds for Cybershake and 44.215 seconds for Montage.

The study has also shown some rather unexpected cases where the models were able to solve some problem instances even better than the PACSA algorithm and also solve some problem instances that the PACSA algorithm fails to solve.

7.1 Future work

There are several ways to continue the research conducted in this thesis and several of them have already been mentioned in the discussion.There are several ways to continue the research conducted in this thesis and several of them have already been mentioned in the discussion. One way to continue could be to investigate the use of other machine learning models than the ones used in this thesis. Another way could be to investigate the models used in this work but evaluate them more deeply than the time constraints allowed for this thesis and do a more thorough search for the optimal parameters of each considered model. The research could be continued by taking more time to build a larger dataset (with more budget values and sizes of the workflows) and investigate if features improving the learnability of the problem further could be created, extracted or engineered.

Another obviously interesting way to continue the research is to investigate the machine learning models learnability applied to other types of scientific workflows. The work could also be continued by considering other instance types than the eight used in this thesis. However if you were to investigate other instance types as well, the implementation in the *optimization-framework* [16] might have to be updated. The research could also be continued by using machine learning to imitate another state-of-the-art algorithm than the PACSA algorithm to come up with provisioning solutions. Yet another way to continue the research could be to utilize another scheduler than the HEFT algorithm to schedule the machine learning predicted provisioning solutions.

Bibliography

[1] J.D. Ullman. “NP-complete scheduling problems”. In: *Journal of Computer and System Sciences* 10.3 (1975), pp. 384–393. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(75)80008-0. URL: http://www.sciencedirect.com/science/article/pii/S0022000075800080.

[2] Philip Maechling et al. “SCEC CyberShake workflows—automating probabilistic seismic hazard analysis calculations”. In: *Workflows for e-Science*. Springer, 2007, pp. 143–163.

[3] S. Bharathi et al. “Characterization of scientific workflows”. In: *2008 Third Workshop on Workflows in Support of Large-Scale Science*. Nov. 2008, pp. 1–10. DOI: 10.1109/WORKS.2008.4723958.

[4] Haluk Topcuoglu, Salim Hariri, and Min-you Wu. “Performance-effective and low-complexity task scheduling for heterogeneous computing”. In: *IEEE transactions on parallel and distributed systems* 13.3 (2002), pp. 260–274.

[5] Luis M. Vaquero et al. “A Break in the Clouds: Towards a Cloud Definition”. In: *SIGCOMM Comput. Commun. Rev.* 39.1 (Dec. 2008), pp. 50–55. ISSN: 0146-4833. DOI: 10.1145/1496091.1496100. URL: http://doi.acm.org.focus.lib.kth.se/10.1145/1496091.1496100.

[6] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. “Cloud computing: A study of infrastructure as a service (IAAS)”. In: *International Journal of engineering and information Technology* 2.1 (2010), pp. 60–63.

[7] Sukhpal Singh and Inderveer Chana. “A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges”. In: *Journal of Grid Computing* 14.2 (June 2016), pp. 217–264. ISSN: 1572-9184. DOI: 10.1007/s10723-015-9359-2. URL: https://doi.org/10.1007/s10723-015-9359-2.

[8] Sukhpal Singh and Inderveer Chana. “Cloud resource provisioning: survey, status and future research directions”. In: *Knowledge and Information Systems* 49.3 (2016), pp. 1005–1069.

[9] V. K. Bhise and A. S. Mali. “Cloud resource provisioning for Amazon EC2”. In: *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT)*. July 2013, pp. 1–7. DOI: `10.1109/ICCCNT.2013.6726565`.

[10] *Amazon EC2 pricing AWS, Amazon*. URL: `https://aws.amazon.com/ec2/pricing/`. Accessed: 2019-05-10.

[11] *Announcing Amazon EC2 per second billing AWS, Amazon*. URL: `https://aws.amazon.com/about-aws/whats-new/2017/10/announcing-amazon-ec2-per-second-billing/`. Accessed: 2019-05-10.

[12] Gideon Juve and Ewa Deelman. “Scientific Workflows in the Cloud”. In: *Grids, Clouds and Virtualization*. Ed. by Massimo Cafaro and Giovanni Aloisio. London: Springer London, 2011, pp. 71–91. ISBN: 978-0-85729-049-6. DOI: `10.1007/978-0-85729-049-6_4`. URL: `https://doi.org/10.1007/978-0-85729-049-6_4`.

[13] Rizos Sakellariou et al. “Scheduling Workflows with Budget Constraints”. In: *Integrated Research in GRID Computing: CoreGRID Integration Workshop 2005 (Selected Papers) November 28–30, Pisa, Italy*. Ed. by Sergei Gorlatch and Marco Danelutto. Boston, MA: Springer US, 2007, pp. 189–202. ISBN: 978-0-387-47658-2. DOI: `10.1007/978-0-387-47658-2_14`. URL: `https://doi.org/10.1007/978-0-387-47658-2_14`.

[14] Maciej Malawski et al. “Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds”. In: *Future Generation Computer Systems* 48 (2015). Special Section: Business and Industry Specific Cloud, pp. 1–18. ISSN: 0167-739X. DOI: `10.1016/j.future.2015.01.004`. URL: `http://www.sciencedirect.com/science/article/pii/S0167739X15000059`.

[15] Gideon Juve et al. “Characterizing and profiling scientific workflows”. In: *Future Generation Computer Systems* 29.3 (2013), pp. 682–692.

[16] Hessam Modaberi Hamid Reza Faragardi. *Optimization Framework*. `https://github.com/workflow-scheduling-on-cloud-instances/optimization-framework`. Accessed: 2019-06-11.

[17] Martın Pedemonte, Sergio Nesmachnow, and Héctor Cancela. “A survey on parallel ant colony optimization”. In: *Applied Soft Computing* 11.8 (2011), pp. 5181–5197.

[18] Peter JM Van Laarhoven and Emile HL Aarts. “Simulated annealing”. In: *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[19] R.W. Eglese. “Simulated annealing: A tool for operational research”. In: *European Journal of Operational Research* 46.3 (1990), pp. 271–281. ISSN: 0377-2217. DOI: `10.1016/0377-2217(90)90001-R`. URL: `http://www.sciencedirect.com/science/article/pii/037722179090001R`.

[20] Aurélien Géron. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems*. " O’Reilly Media, Inc.", 2017.

[21] Hanen Borchani et al. “A survey on multi-output regression”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.5 (2015), pp. 216–233.

[22] Leo Breiman et al. “Classification and regression trees”. In: (1984).

[23] *Decision Trees*. URL: `https://scikit-learn.org/stable/modules/tree.html#tree`. Accessed: 2019-08-09.

[24] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[25] ZQ John Lu. “The elements of statistical learning: data mining, inference, and prediction”. In: *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 173.3 (2010), pp. 693–694.

[26] Corinna Cortes and Vladimir Vapnik. “Support-vector networks”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[27] Alex J Smola and Bernhard Schölkopf. “A tutorial on support vector regression”. In: *Statistics and computing* 14.3 (2004), pp. 199–222.

[28] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001.

[29] K-R Müller et al. “Predicting time series with support vector machines”. In: *International Conference on Artificial Neural Networks*. Springer. 1997, pp. 999–1004.

[30] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. Vol. 821. John Wiley & Sons, 2012.

[31] Stuart Jonathan Russell. *Artificial intelligence : a modern approach*. eng. 3. ed.. Prentice Hall series in artificial intelligence. Boston: Pearson Education, 2010. ISBN: 0-13-207148-7.

[32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[33] Girish Chandrashekar and Ferat Sahin. “A survey on feature selection methods”. In: *Computers Electrical Engineering* 40.1 (2014). 40th-year commemorative issue, pp. 16–28. ISSN: 0045-7906. DOI: `10.1016/j.compeleceng.2013.11.024`. URL: `http://www.sciencedirect.com/science/article/pii/S0045790613003066`.

[34] Frank Hutter, Jörg Lücke, and Lars Schmidt-Thieme. “Beyond manual tuning of hyperparameters”. In: *KI-Künstliche Intelligenz* 29.4 (2015), pp. 329–337.

[35] James Bergstra and Yoshua Bengio. “Random search for hyper-parameter optimization”. In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[36] Cort J Willmott and Kenji Matsuura. “Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance”. In: *Climate research* 30.1 (2005), pp. 79–82.

[37] Mohammad Masdari et al. “Towards workflow scheduling in cloud computing: A comprehensive analysis”. In: *Journal of Network and Computer Applications* 66 (2016), pp. 64–82. ISSN: 1084-8045. DOI: `10.1016/j.jnca.2016.01.018`. URL: `http://www.sciencedirect.com/science/article/pii/S108480451600045X`.

[38] Hamid Reza Faragardi et al. “A Budget-Constrained Resource Provisioning Scheme for Workflow Scheduling in IaaS Cloud”. In: *IEEE Transaction on Parallel and Distributed Systems (TPDS), 2020* (Mar. 2019). DOI: `10.13140/RG.2.2.13549.84967`.

[39] Zong-Gan Chen et al. “Multiobjective cloud workflow scheduling: a multiple populations ant colony system approach”. In: *IEEE transactions on cybernetics* 49.8 (2018), pp. 2912–2926.

[40] X. Wang et al. “Scheduling Budget Constrained Cloud Workflows with Particle Swarm Optimization”. In: *2015 IEEE Conference on Collaboration and Internet Computing (CIC)*. Oct. 2015, pp. 219–226. DOI: 10.1109/CIC.2015.12.

[41] Thanh Phuong Pham, Juan J Durillo, and Thomas Fahringer. “Predicting workflow task execution time in the cloud using a two-stage machine learning approach”. In: *IEEE Transactions on Cloud Computing* (2017).

[42] M. H. Hilman, M. A. Rodriguez, and R. Buyya. “Task Runtime Prediction in Scientific Workflows Using an Online Incremental Learning Approach”. In: *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. Dec. 2018, pp. 93–102. DOI: 10.1109/UCC.2018.00018.

[43] A. A. Bankole and S. A. Ajila. “Predicting cloud resource provisioning using machine learning techniques”. In: *2013 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. May 2013, pp. 1–4. DOI: 10.1109/CCECE.2013.6567848.

[44] A. Biswas et al. “Automatic Resource Provisioning: A Machine Learning Based Proactive Approach”. In: *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*. Dec. 2014, pp. 168–173. DOI: 10.1109/CloudCom.2014.147.

[45] *Python Software Foundation. Python Language Reference*. URL: https://www.python.org/. Accessed: 2019-06-20.

[46] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[47] Fabian Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *J. Mach. Learn. Res.* 12 (Nov. 2011), pp. 2825–2830. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1953048.2078195.

[48] W McKinney. “Pandas, python data analysis library. 2015”. In: *Reference Source* (2014).

[49] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. “The NumPy array: a structure for efficient numerical computation”. In: *Computing in Science & Engineering* 13.2 (2011), p. 22.

[50] Thomas Kluyver et al. “Jupyter Notebooks – a publishing format for reproducible computational workflows”. In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by F. Loizides and B. Schmidt. IOS Press. 2016, pp. 87–90.

[51] Ewa Deelman et al. “Pegasus: a Workflow Management System for Science Automation”. In: *Future Generation Computer Systems* 46 (2015). Funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575, pp. 17–35. DOI: `10.1016/j.future.2014.10.008`. URL: `http://pegasus.isi.edu/publications/2014/2014-fgcs-deelman.pdf`.

[52] Z. Zhu et al. “Evolutionary Multi-Objective Workflow Scheduling in Cloud”. In: *IEEE Transactions on Parallel and Distributed Systems* 27.5 (May 2016), pp. 1344–1357. DOI: `10.1109/TPDS.2015.2446459`.

[53] *Instance Types - Amazon Elastic Compute Cloud*. URL: `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html#AvailableInstanceTypes`. Accessed: 2019-07-12.

[54] S. B. Kotsiantis. “Supervised Machine Learning: A Review of Classification Techniques”. In: *Proceedings of the 2007 Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2007, pp. 3–24. ISBN: 978-1-58603-780-2. URL: `http://dl.acm.org/citation.cfm?id=1566770.1566773`.

[55] Rich Caruana and Alexandru Niculescu-Mizil. “An Empirical Comparison of Supervised Learning Algorithms”. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML ’06. Pittsburgh, Pennsylvania, USA: ACM, 2006, pp. 161–168. ISBN: 1-59593-383-2. DOI: `10.1145/1143844.1143865`. URL: `http://doi.acm.org.focus.lib.kth.se/10.1145/1143844.1143865`.

[56] *Multiclass and multilabel algorithms*. URL: `https://scikit-learn.org/stable/modules/multiclass.html`. Accessed: 2019-08-01.

[57] *sklearn.multioutput.MultiOutputRegressor*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.multioutput.MultiOutputRegressor.html`. Accessed: 2019-08-01.

[58] *sklearn.model*$_selection.RandomizedSearchCV$. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html`. Accessed: 2019-08-09.

[59] Alice Zheng. *Evaluating machine learning models: a beginner's guide to key concepts and pitfalls*. O'Reilly Media, 2015.

Appendix A

Data features and labels

A.1 Features in both datasets

Table A.1: Features and labels in both datasets.

Name	Type	Description
NrOfM1SmallAffordable	Feature	The number of m1.small affordable within current budget
NrOfM1MediumAffordable	Feature	The number of m1.medium affordable within current budget
NrOfM3MediumAffordable	Feature	The number of m3.medium affordable within current budget
NrOfM1LargeAffordable	Feature	The number of m1.large affordable within current budget
NrOfM3LargeAffordable	Feature	The number of m3.large affordable within current budget
NrOfM1XLargeAffordable	Feature	The number of m1.xlarge affordable within current budget
NrOfM3XLargeAffordable	Feature	The number of m3.xlarge affordable within current budget

Continued on next page

Table A.1 – continued from previous page

Name	Type	Description
NrOfM32XLargeffordable	Feature	The number of m3.2xlarge affordable within current budget
AvgExeOnM1Small	Feature	Average execution time of all tasks on m1.small
AvgExeOnM1Medium	Feature	Average execution time of all tasks on m1.medium
AvgExeOnM3Medium	Feature	Average execution time of all tasks on m3.medium
AvgExeOnM1Large	Feature	Average execution time of all tasks on m1.large
AvgExeOnM3Large	Feature	Average execution time of all tasks on m3.large
AvgExeOnM1XLarge	Feature	Average execution time of all tasks on m1.xlarge
AvgExeOnM3XLarge	Feature	Average execution time of all tasks on m3.xlarge
AvgExeOnM32XLarge	Feature	Average execution time of all tasks on m3.2xlarge
Budget	Feature	The budget for current problem instance
NumberOfTasks	Feature	The number of tasks in the current workflow
m1.small	Label	The number of m1.small to provision
m1.medium	Label	The number of m1.medium to provision
m3.medium	Label	The number of m3.medium to provision
m1.large	Label	The number of m1.large to provision
m3.large	Label	The number of m3.large to provision
m1.xlarge	Label	The number of m1.xlarge to provision
m3.xlarge	Label	The number of m3.xlarge to provision

Continued on next page

Table A.1 – continued from previous page

Name	Type	Description
m3.2xlarge	Label	The number of m3.2xlarge to provision

A.2 Cybershake specific features

Table A.2: Features present in only Cybershake dataset.

Name	Type	Description
NrOfZipPSA	Feature	The number of ZipPSA tasks in the current cybershake workflow
NrOfZipSeis	Feature	The number of ZipSeis tasks in the current cybershake workflow
NrOfExtractSGT	Feature	The number of ExtractSGT tasks in the current cybershake workflow
NrOfSeismogramSynthesis	Feature	The number of SeismogramSynthesis tasks in the current cybershake workflow
NrOfPeakValCalc0kaya	Feature	The number of PeakValCalc0kaya tasks in the current cybershake workflow

A.3 Montage specific features

Table A.3: Features present in only Montage dataset.

Name	**Type**	**Description**
NrOfmImgTbl	Feature	The number of mImgTbl tasks in the current montage workflow
NrOfmAdd	Feature	The number of mAdd tasks in the current montage workflow
NrOfmBackground	Feature	The number of mBackground tasks in the current montage workflow
NrOfmBgModel	Feature	The number of mBgModel tasks in the current montage workflow
NrOfmConcatFit	Feature	The number of mConcatFit tasks in the current montage workflow
NrOfmDiffFit	Feature	The number of mDiffFit tasks in the current montage workflow
NrOfmJPEG	Feature	The number of mJPEG tasks in the current montage workflow
NrOfmProjectPP	Feature	The number of mProjectPP tasks in the current montage workflow
NrOfmShrink	Feature	The number of mShrink tasks in the current montage workflow

Appendix B

Problem instances solved that the PACSA algorithm didn't solve

Table B.1: Solved problem instances PACSA didn't solve

Workflow & tasks	Budget	Model(solver)	Makespan
Cybershake1000	1.42	Decision Tree	3966
Cybershake1000	1.44	Decision Tree	3966
Cybershake1000	1.46	Decision Tree	3966
Cybershake1000	1.44	Support Vector	4112
Cybershake950	1.18	Support Vector	5973
Cybershake400	0.48	Support Vector	4826
Cybershake950	1.16	Support Vector	5973
Cybershake900	1.28	Support Vector	5470
Cybershake750	0.96	Support Vector	4724
Cybershake850	0.48	Support Vector	10560
Cybershake850	0.74	Support Vector	10560
Cybershake750	1.04	Support Vector	4724
Cybershake800	0.58	Support Vector	9491
Cybershake900	1.4	Support Vector	3646
Cybershake1000	1.42	Support Vector	4112
Cybershake1000	1.34	Support Vector	6169
Cybershake450	0.48	Support Vector	5486
Cybershake600	0.5	Support Vector	7577
Cybershake750	0.78	Support Vector	9449

Continued on next page

Table B.1: Solved problem instances PACSA didn't solve

Workflow & tasks	Budget	Model(solver)	Makespan
Cybershake500	0.7	Support Vector	5957
Cybershake650	0.92	Support Vector	3943
Cybershake800	0.94	Support Vector	4745
Cybershake600	0.92	Support Vector	3788
Cybershake950	0.78	Support Vector	11947
Cybershake650	0.48	Support Vector	7887
Cybershake900	0.9	Support Vector	5470
Cybershake700	0.7	Support Vector	8528
Cybershake750	0.54	Support Vector	9449
Cybershake650	0.68	Support Vector	7887
Cybershake800	0.72	Support Vector	9491
Cybershake550	0.48	Support Vector	6488
Cybershake950	1.54	Support Vector	3982
Cybershake1000	0.66	Support Vector	12338
Cybershake900	0.58	Support Vector	10940
Cybershake850	1.0	Support Vector	5280
Cybershake700	1.06	Support Vector	4264
Cybershake500	0.56	Support Vector	5957
Cybershake400	0.54	Support Vector	4826
Cybershake900	1.08	Support Vector	5470
Cybershake1000	0.7	Support Vector	12338
Cybershake800	0.82	Support Vector	9491
Cybershake900	0.5	Support Vector	10940
Cybershake1000	1.46	Support Vector	4112
Cybershake850	1.08	Support Vector	5280
Cybershake600	0.94	Support Vector	3788
Cybershake800	0.5	Support Vector	9491
Cybershake700	0.98	Support Vector	4264
Cybershake600	0.54	Support Vector	7577
Cybershake1000	0.56	Support Vector	12338
Cybershake700	0.94	Support Vector	4264
Cybershake750	0.48	Support Vector	9449
Cybershake900	0.92	Support Vector	5470
Cybershake700	0.84	Support Vector	8528

Continued on next page

Table B.1: Solved problem instances PACSA didn't solve

Workflow & tasks	Budget	Model(solver)	Makespan
Cybershake900	0.6	Support Vector	10940
Cybershake700	0.64	Support Vector	8528
Cybershake650	0.78	Support Vector	7887
Cybershake800	0.84	Support Vector	9491
Cybershake550	0.58	Support Vector	6488
Cybershake700	0.76	Support Vector	8528
Cybershake800	0.6	Support Vector	9491
Cybershake1000	0.58	Support Vector	12338
Cybershake850	1.18	Support Vector	5280
Cybershake650	0.82	Support Vector	7887
Cybershake950	1.44	Support Vector	3982
Cybershake750	0.74	Support Vector	9449
Cybershake900	0.48	Support Vector	10940
Cybershake1000	1.0	Support Vector	6169
Cybershake950	1.52	Support Vector	3982
Cybershake500	0.6	Support Vector	5957
Cybershake400	0.5	Support Vector	4826
Cybershake950	0.54	Support Vector	11947
Cybershake650	0.62	Support Vector	7887
Cybershake750	0.88	Support Vector	9449
Cybershake700	0.86	Support Vector	8528
Cybershake850	0.82	Support Vector	10560
Cybershake950	1.22	Support Vector	5973
Cybershake950	1.48	Support Vector	3982
Cybershake750	0.56	Support Vector	9449
Cybershake700	1.08	Support Vector	4264
Cybershake1000	0.84	Support Vector	12338
Cybershake900	0.56	Support Vector	10940
Cybershake900	1.18	Support Vector	5470
Cybershake950	0.76	Support Vector	11947
Cybershake650	0.64	Support Vector	7887
Cybershake650	0.6	Support Vector	7887
Cybershake400	0.46	Support Vector	4826
Cybershake750	0.46	Support Vector	9449

Continued on next page

Table B.1: Solved problem instances PACSA didn't solve

Workflow & tasks	Budget	Model(solver)	Makespan
Cybershake900	0.86	Support Vector	10940
Cybershake400	0.6	Support Vector	4826
Cybershake600	0.62	Support Vector	7577
Montage1000	0.62	Support Vector	6052
Montage800	0.26	Support Vector	9707
Montage650	0.48	Support Vector	3947
Montage1000	0.36	Support Vector	12105
Montage950	0.76	Support Vector	3900
Montage850	0.48	Support Vector	5153
Montage800	0.44	Support Vector	9707
Montage1000	0.66	Support Vector	6052
Montage700	0.4	Support Vector	8440
Montage600	0.34	Support Vector	7285
Montage750	0.48	Support Vector	4526
Montage700	0.54	Support Vector	4220
Montage1000	0.54	Support Vector	6052
Montage800	0.48	Support Vector	4853
Montage800	0.42	Support Vector	9707
Montage700	0.26	Support Vector	8440
Montage600	0.24	Support Vector	7285
Montage900	0.28	Support Vector	10959
Montage650	0.38	Support Vector	7895
Montage400	0.3	Support Vector	4832
Montage1000	0.64	Support Vector	6052
Montage700	0.48	Support Vector	4220
Montage300	0.28	Support Vector	3610
Montage950	0.3	Support Vector	11519
Montage800	0.58	Support Vector	4853
Montage950	0.78	Support Vector	3900
Montage850	0.38	Support Vector	10307
Montage850	0.44	Support Vector	10307
Montage550	0.3	Support Vector	6635
Montage750	0.62	Support Vector	4526
Montage950	0.66	Support Vector	5759

Continued on next page

Table B.1: Solved problem instances PACSA didn't solve

Workflow & tasks	Budget	Model(solver)	Makespan
Montage500	0.36	Support Vector	6024
Montage1000	0.6	Support Vector	6052
Montage700	0.56	Support Vector	4220
Montage750	0.6	Support Vector	4526
Montage700	0.46	Support Vector	4220
Montage900	0.36	Support Vector	10959

TRITA -EECS-EX-2019:832