University of Stuttgart
Institute for Signal Processing and System Theory
Professor Dr.-Ing. B. Yang

Forschungsarbeit S1320

Solving the extended time-cost trade-off problem (TCTP) with activity and event variant selection, schedule compression and stochastic nonlinear models.

Lösung des erweiterten Zeit-Kosten Trade-off Problems (TCTP) mit Aktivitäts- und Ereignisvariantenauswahl, Zeitplankompression und stochastischen nichtlinearen Modellen.

Author: Malte Ebner

Date of work begin: 18.04.2019
Date of submission: 23.10.2019

Supervisor: Prof. Bin Yang

Keywords: project management, black-box optimization, DTCTP, TCTP

Choosing the activity variants of a project such that the total project duration and cost is minimized is a problem well known in literature as the discrete time-cost trade-off problem (DTCTP). This problem is extended and generalized by adding stochastic events and activity durations and allowing a continuous-valued schedule compression of activities, making it a mixed (categorical and continuous), probabilistic and nonlinear optimization problem.
This thesis has two contributions: First it shows that algorithms for solving this generalized time-cost trade-off problem perform much better than humans and heuristics do and compares the performance of Bayesian Optimization, Genetic and Actor-Critic algorithms on these kinds of problems. Second it provides a framework for modelling complex project management problems including stochastic activity durations, stochastic events and continuous-valued schedule compression which is used for providing a simulation model of three example projects based on project management business games.

Contents

Acronyms

AON activity on node (form of project management network visualization).

CEI cost efficiency index (of contractors in a discipline).

DEI duration efficiency index (of contractors in a discipline).
DTCTP discrete time-cost trade-off problem.

EI expected improvement (acquisition function for Bayesian optimization).

LCB lower confidence bound (acquisition function for Bayesian optimization).

MIS Information Systems Project Manager (name for business game).

PDF probability density function.
PERT program evaluation and research task.
PI probability of improvement (acquisition function for Bayesian optimization).

QI quality index (of contractors in a discipline).

RV random variable.

TCTP time-cost trade-off problem.
TD-error temporal difference error (in reinforcement learning).

VAE Variational Autoencoder.

1. Problem description

Projects consist of many different activities which have to be done in a certain sequence as they depend on each other. These activities often have several variants to choose from, e.g. an activity might be done by oneself, with the help of a partner or outsourced to different contractors. These variants differ in duration, cost, quality and other factors. Choosing the activity variants such that the total project duration and cost are minimized is a problem well known in literature as the discrete time-cost trade-off problem (DTCTP). In its most generalized form this problem includes stochastic, nonlinear and non-differentiable relationships and the choice of discrete reaction variants to events. Furthermore, the problem is extended by allowing real-valued schedule compression. Put together this is a high-dimensional, mixed (categorical and real), nonlinear stochastic optimization problem. While both the DTCTP and real-valued stochastic project management optimization have been covered in literature, this thesis combines both problems into one and uses different project models with more activities and/or more complex models than in previous literature.

1.1. Basic problem

A project model can be seen as a function mapping from the space of all activity variants to a tuple of the real-valued key performance indicators time and cost:

$$model : V_1 \times V_2 \times ...V_n \rightarrow \mathbb{R} \times \mathbb{R} \tag{1.1}$$

with $\mathrm{V_i}$ being the finite set of all variants of activity i, n being the number of activities and the output space representing the (time, cost) - tuple. Furthermore a loss function

$$loss\_function : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \tag{1.2}$$

dependent on the performance indicators is defined as single objective function. By combining the model and the loss function, a function mapping from the high-dimensional discrete space of activity variants to the real space, is created:

$$model\_to\_loss\_function : V_1 \times V_2 \times ...V_n \rightarrow \mathbb{R} \tag{1.3}$$

Finding the minimum of this function is the basic problem. It is generally named in literature as discrete time-cost trade-off problem (DTCTP). The trade-off comes from the fact that project activity variants with lower durations usually have higher costs and vice-versa.

Project schedules can be visualized as project network diagrams as shown in 1.1 in the activity on node (AON) form. Activity B can only be started when activity A is finished, activity C is worked on in parallel. Because of the dependency of the project activities choosing the best variant for one activity is a non-trivial task. In the example project choosing a variant of activity A with a shorter duration does not help in finishing the project earlier, as the project finishes earliest when activity C is finished, i.e. after 5 days. It would be better to choose a variant of A costing less while taking one day longer or to choose a shorter variant of C to reduce project duration.

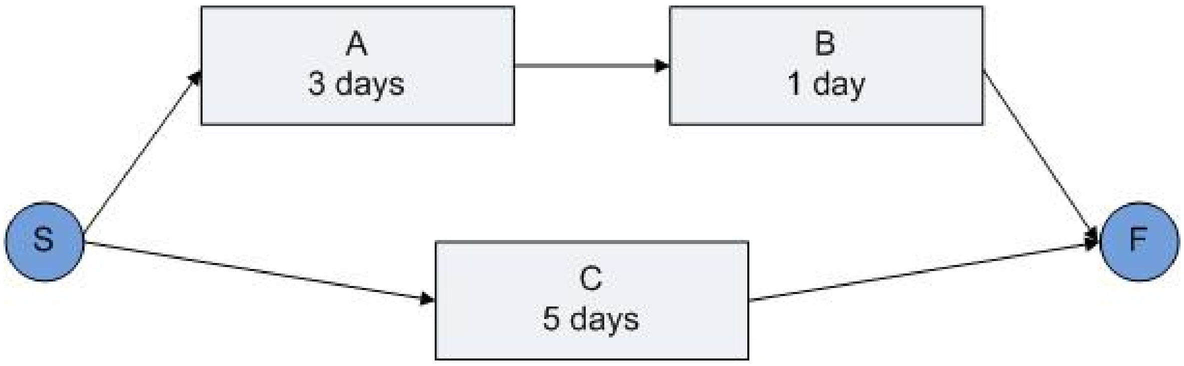


Figure 1.1.: Example project network diagram with start S, finish F and three activities

Source: [1]

A heuristic approach to solve the DTCT problem using the critical path method was proposed by Antill and Ronald in 1990 [2]. Mathematical solutions include linear programming [3], integer programming [4] and dynamic programming [5]. As the problem is NP-hard [6], these methods do not scale to projects having a lot of activities and activity variants. Hence problem formulations for bigger projects were solved mostly by genetic algorithms as in [7], [8] and [9]. Mungle even used a portfolio of many algorithms to solve the DTCT problem using several benchmarks of different sizes [10].

1.2. 1st problem extension: Nonlinear and non-differentiable model

Linear programming for solving the DTCT problem only works when all relationships in the model are piecewise linear. Most DTCT problems fulfil this condition. A good overview of the linear relationships is given in [10], page 4ff. In reality this is rarely the case which is included in some models, e.g. in the project management game by Batista et al. [11] the duration of an activity depends nonlinearly on the quality of its predecessor activities. This makes linear optimization algorithms and heuristics based on linear relationships infeasible. Furthermore, allowing any kind of project models, including non-differentiable ones, makes it unsolvable by gradient-based algorithms. Thus in its generalized form a project model is so complex that nearly only black-box optimization algorithms are applicable.

1.3. 2nd problem extension: Additional activity variant parameters

Nearly all DTCT problems assume that activity variants differ only in cost and duration even though Atkinson has shown in 1999 that there are many other success criteria like quality, user satisfaction and organisational impact like team morale [12]. Extending the properties of the activity variants such that they also differ in more parameters makes the project model more realistic. As an example, Edholm and Lindström [13] have shown that ordering overtime causes short-term benefits, but decreases team morale, which decreases quality and productivity in the long-term.

1.4. 3rd problem extension: Stochastic activity duration

The project model so far assumed that durations of an activity variant are a deterministic fixed number, usually the estimated or averaged duration of it. Elmaghraby proved that this simplification can cause huge errors in estimating the real duration of the project [14]. This can be avoided by extending the project models such that the duration of each activity variant is a random variable.

A minimal example for the difference between stochastic and deterministic modelling would be a project consisting of two activities in parallel with both activity durations being independent and having an exponential distribution with an expectation of 1 month. Thus the probability density functions (pdf) of the durations are the following:

$$\begin{aligned} &pdf\_duration\_activity_1(t) = pdf\_duration\_activity_2(t) = exp(\frac{-t}{month}) \\ &duration\_total = max(duration\_activity_1, duration\_activity_2) \\ &pdf\_duration\_total(t) = 2\ exp(\frac{-t}{month}) - 2\ exp(\frac{-2\ t}{month}) \end{aligned} \tag{1.4}$$

Hence the expected duration of both activities is 1 month and the expected total duration 1.5 months. On the other hand, when assuming that both activities would need deterministically 1 month, the total project duration would also be deterministically 1 month, which is far shorter than the true expected duration.

In 1959 Malcolm et al. proposed the PERT (Program Evaluation Research Task) approach to handle the stochastic nature of task durations in project planning. [15] They found that giving a good estimate of a task's duration distribution is very difficult, thus experts estimating a duration should rather give three estimates: the minimum (t_{min}), most likely (t_{ml}) and maximum (t_{max}) duration. The average duration t_m is a weighted average of them. The probability density function of the PERT distribution is shown in 1.2. Out of the three duration estimates, which are intuitive parameters, the abstract parameters for a scaled beta-distribution can be calculated [16] (page 235).

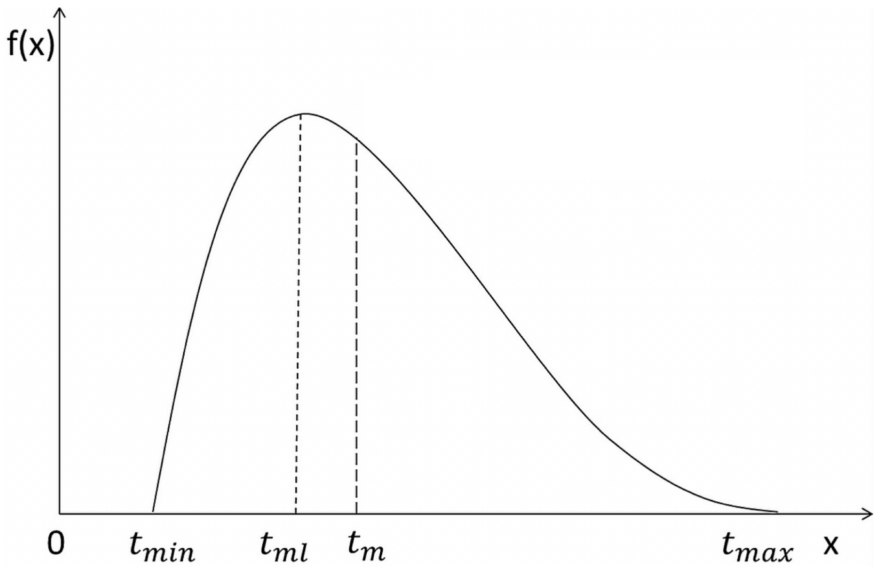


Figure 1.2.: PERT probability density function of project duration

Source: [17], page 5

Stochastic activity durations have only been used in a few optimization models so far with Azaron et al. [18] modelling activity durations as Erlang distributions and Haque et al. [19] using piecewise linear probability density functions.

1.5. 4th problem extension: Unexpected events

Often unexpected events happen during the course of a project, e.g. a team member resigns, a supplier declares bankruptcy or there is a technical defect. The probability of occurrence and the effect of these events may depend on the choice of activity variants. Often these events have different categorical variants of reactions to it, extending the problem by choosing a reaction variant to these events:

$$model\_with\_events : V_1 \times V_2 \times ...V_n \times E_1 \times E_2 \times ...E_m \to \mathbb{R} \tag{1.5}$$

E_i denotes the space of all reaction variants to an event, m is the number of events. Again V_i is the space of variants for one activity and n the number of activities. The output space is the real-valued loss.

1.6. 5th problem extension: Continuous-valued schedule compression of activity durations

The duration of an activity can be reduced by allocating more resources to the activity, for example by letting more people work on the project or ordering overtime. This process is called schedule compression. As the cost of reducing the duration depends strongly on the project domain and the individual activity, there is no rule of thumb how to model this time-cost trade-off relationship for all kind of project domains and actions.

One domain-specific model is the COCOMO II model. It was developed by fitting a regression model to empirical data of many software development projects [20]. The relationship between schedule compression and cost according to it is shown in Table 1.1. As its relationship between cost and time only applies to software projects it should be rather seen as a place holder for domain-specific schedule compression trade-off functions than a general model.

duration relative to nominal duration	75%	85%	100%	130%	160%
cost multiplier	1.43	1.14	1	1	1

Table 1.1.: COCOMO II schedule compression cost multipliers

In order to allow a continuous-valued schedule compression, a continuous-valued function has to be fit to these data points, which is done in two steps: Firstly the cost multiplier for durations >= 100% is set to one. Secondly a scaled exponential function is fit to the three data points having a relative duration <= 100%. Put together this yields to the function in equation 1.6 fitting exactly to all data points in Table 1.1.

$$cost(duration_{\mathrm{rel}}) = \begin{cases} 560.7 * exp(-9.43 * duration_{\mathrm{rel}}) + 0.955 & duration_{\mathrm{rel}} < 1 \\ 1 & duration_{\mathrm{rel}} >= 1 \end{cases} \tag{1.6}$$

As schedule compression factors greater than one increase the duration of an activity without any cost benefit and schedule compression factors smaller than 0.75 are unachievable in most real-world cases, all schedule compression factors are set to be in the interval [0.75; 1].

After adding the real-valued schedule compression factor for each activity to the input space of the model, it has the following form:

$$model\_with\_events\_and\_schedule\_compression : V_1 \times V_2 ... V_n \times E_1 \times E_2 ... E_m \times [0.75; 1]^{\mathrm{n}} \rightarrow \mathbb{R} \tag{1.7}$$

Again $\mathrm{V_i}$ is the space of variants for activity i, n the number of activities, $\mathrm{E_j}$ denotes the space of all reaction variants to event j and m is the number of events. $[0.75; 1]^{\mathrm{n}}$ are the schedule compression factors for each of the n activities and the output space is the real-valued loss.

Both Azaron et al. [18] and Haque et al. [19] used genetic algorithms to find the optimal resource allocation and schedule compression of project activities. Their models have the limitation of only having very few activities (up to 10) and not including any choice between different activity variants, thus only being a continuous optimization problem.

1.7. Generalized project management optimization problem

By defining the space of all activity variants, event reaction variants and schedule compression factors as action space all project models can be described by equation 1.8.

$$\begin{aligned} &model\_generalized : A \rightarrow \mathbb{R} \\ &with\ A \subseteq V_1 \times V_2 ... V_n \times E_1 \times E_2 ... E_m \times [0.75; 1]^{\mathrm{n}} \end{aligned} \tag{1.8}$$

Here A denotes the action space consisting of all categorical and continuous variables and the output space is the real valued loss. As the models must not necessarily contain events and schedule compression, A is only a subset and not equal to the full space used in 1.7.

1.8. Overview of literature on project management optimization problems

The project management optimization problems in current literature can be compared by the aspects of project management they incorporate into their problem definition. Following aspects can be identified:

1. choice between discrete activity variants:
   Project models may assume that there is a finite set of different variants for each activity, from which one has to be chosen.
2. nonlinearities:
   The model may include nonlinear relationships.
3. activity variant parameters additional to time and cost:
   Most project models assume that activity variants only differ in time and cost, while others include additional parameters like quality.
4. stochastic activity durations:
   Most project models assume activity durations to be deterministic while others model them as stochastic.
5. unexpected events:
   The project model may include unexpected events and variants to choose from as reaction to the events.
6. continuous-valued schedule compression:
   Models may allow to reduce the duration of an activity by increasing resource allocation and cost.
7. resource and staff allocation:
   There may be a set of resources or staff with special skills and each activity needs a special combination of them to be worked on. Thus this assignment problem is part of the project management problem.

A non-exhaustive overview which literature covers which aspects is given in Table 1.2. It shows that the aspects of project management problems covered in this thesis have not yet been covered before, especially not in combination.

	DTCTP			this
papers	[2] [3] [4] [5] [9] and more	[18] [19]	[8]	thesis
choice between discrete activity variants	x			x
nonlinearities		x		x
activity variant parameters beneath time & cost				x
stochastic activity durations		x		x
unexpected events				x
continuous-valued schedule compression		x		x
resource and staff allocation			x	

Table 1.2.: Literature overview concerning aspects covered

2. Project Models

The particular project models used for representing the general project optimization problem are all business simulation games used for teaching project management. They all provide a project network plan in the activity on node (AON) form showing all activities and their dependencies in a graph. As they all have one start and end activity, the total project time equals the time difference between the startpoint of the first activity and the endpoint of the last one. The total project cost is the sum of the cost of all activities plus any costs that cannot be assigned to a specific activity. The purpose of all models is to provide the mapping from the action space to the loss as denoted in equation 1.8.

2.1. Refinery project model

In 2015, Moraes and Batista developed a project management simulation game in which players had to choose from different contractors for the construction of a refinery [11]. The project consists of 25 activities as shown in figure 2.1.

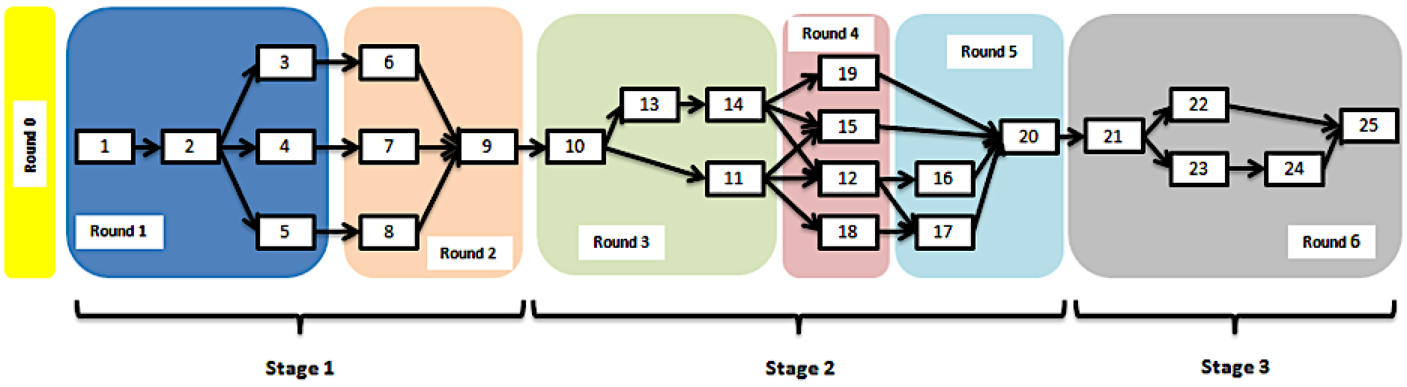


Figure 2.1.: Refinery project network plan

Source: [11], page 99

Each activity has its predecessors, a base cost per day, base duration and discipline of contractors needed for working on it. The table of all activities with these parameters can be found in [11], page 97. The disciplines needed for the activities are: civil, processes, electrical, mechanics, tubing, instruction and automation, safety, multidisciplinary, processes/tubing and processes/civil. Eleven different contractors are available in total, but each of them only has competencies in some disciplines, thus the contractors available for each activity differ. Each contractor-discipline combination has three parameters: a cost efficiency index (CEI), duration efficiency index (DEI) and quality index (QI). They determine an activity's final duration, influence and cost. The exact functions for calculating those are given in equation 2.1. The quality of an activity depends on the average quality of its predecessor activities, causing bad quality on early activities to negatively affect cost and duration of later activities.

$$\begin{aligned} quality &= \frac{3 * QI + mean(predecessors\_qualities)}{4} \\ duration &= \frac{base\_duration}{DEI * quality^2} \\ cost &= \frac{base\_cost\_per\_day * duration}{CEI} \end{aligned} \tag{2.1}$$

The refinery project model was played by 10 student groups and one control group, which tried to choose the contractors such that the total project duration and cost was minimized. The loss to minimize was defined as the product of duration and cost. The performance of the 11 groups is shown in Table 2.1.

	loss [days * million R$]	Duration [days]	cost [million R$]
best	930.76	367.00	2.54
average	1790.66	579.18	3.04
standard deviation	452.31	100.26	0.29

Table 2.1.: Human performance in the refinery project

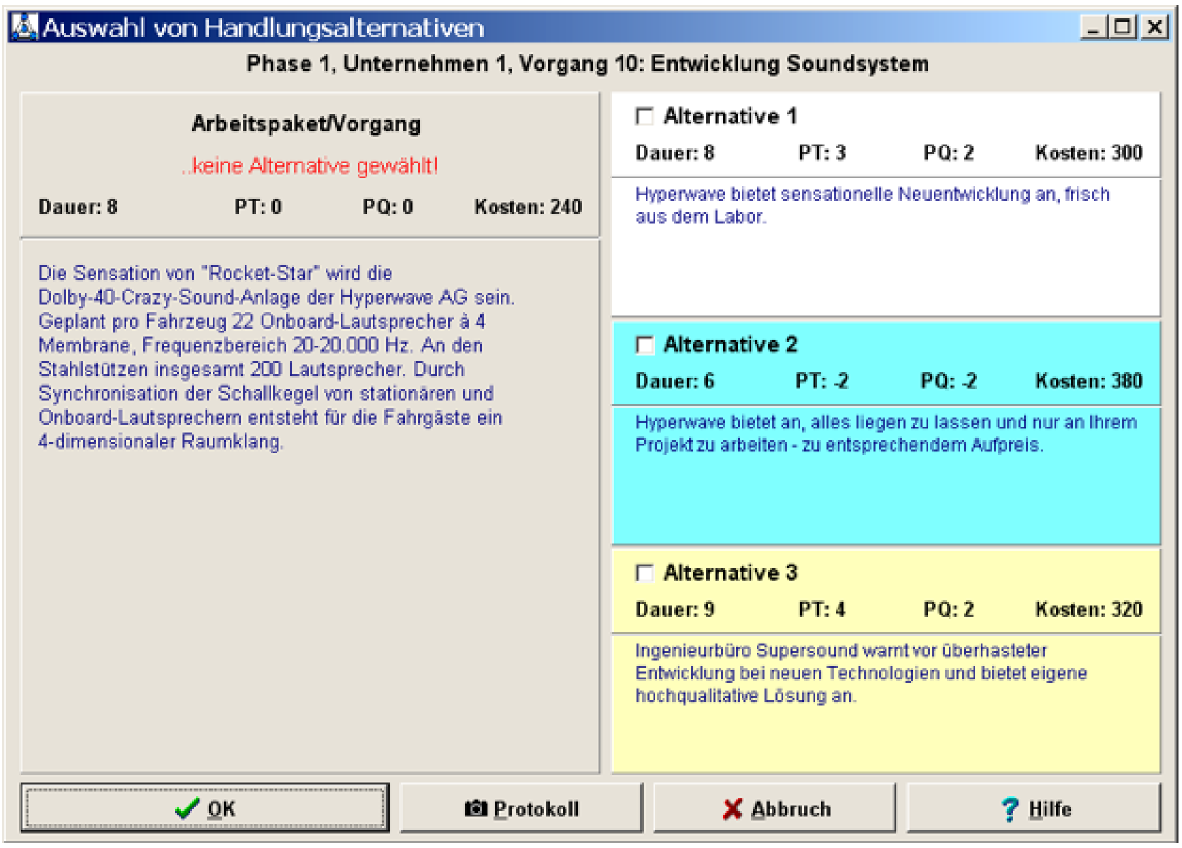
As the human groups only played the game once, they had two disadvantages compared to algorithmic solvers:

- No exact knowledge of the contractors' properties:
  The cost efficiency, duration efficiency and quality index of the contractors were not provided to the students as exact number, but only in the categorical form of 'good' , 'average' and 'weak'.
- No exact knowledge of the simulation model:
  The functions how an activity's final quality, duration and cost are calculated were not provided to the students.

The optimization algorithms do not have that knowledge explicitly either, but they can gain it implicitly by playing the project simulation games many times.

2.2. TopSim roller coaster project model

The roller coaster project simulation game was developed by the company Tata Interactive Systems under the brand TopSim [21]. It consists of 46 activities with 2 to 4 variants each. These variants may be different contractors for one activity or different options how to perform an activity in-house. They differ in cost, duration and points for technology and quality. An example for one activity with its four different variants and their parameters is shown in 2.2.


Auswahl von Handlungsalternativen

Phase 1, Unternehmen 1, Vorgang 10: Entwicklung Soundsystem

Arbeitspaket/Vorgang

keine Alternative gewählt!

Dauer: 8 PT: 0 PQ: 0 Kosten: 240

Die Sensation von "Rocket-Star" wird die Dolby-40-Crazy-Sound-Anlage der Hyperwave AG sein. Geplant pro Fahrzeug 22 Onboard-Lautsprecher à 4 Membrane, Frequenzbereich 20-20.000 Hz. An den Stahlstützen insgesamt 200 Lautsprecher. Durch Synchronisation der Schallkegel von stationären und Onboard-Lautsprechern entsteht für die Fahrgäste ein 4-dimensionaler Raumklang.

Alternative 1

Dauer: 8 PT: 3 PQ: 2 Kosten: 300

Hyperwave bietet sensationelle Neuentwicklung an, frisch aus dem Labor.

Alternative 2

Dauer: 6 PT: -2 PQ: -2 Kosten: 380

Hyperwave bietet an, alles liegen zu lassen und nur an Ihrem Projekt zu arbeiten - zu entsprechendem Aufpreis.

Alternative 3

Dauer: 9 PT: 4 PQ: 2 Kosten: 320

Ingenieurbüro Supersound warnt vor überhasteter Entwicklung bei neuen Technologien und bietet eigene hochqualitative Lösung an.

Figure 2.2.: TopSim roller coaster project example activity

Source: [21], page 96

The points for technology and quality do not influence cost and duration, but rather cause a financial reward or punishment in the end according to a tabula [21], page 15. Finishing the project earlier or later also causes a financial bonus or malus. The profit is calculated with equation 2.2. The loss to minimize is the negative profit. The financial boni dependent on the total project duration, technology index and quality index are shown in figures 2.3, 2.4 and 2.5. All curves are nonlinear around the default values of 65 weeks for the duration respectively 0 for the technology and quality index and become linear towards the extrema.

$$profit = revenue - projectCost + bonus(duration) + bonus(technology) + bonus(quality) \tag{2.2}$$

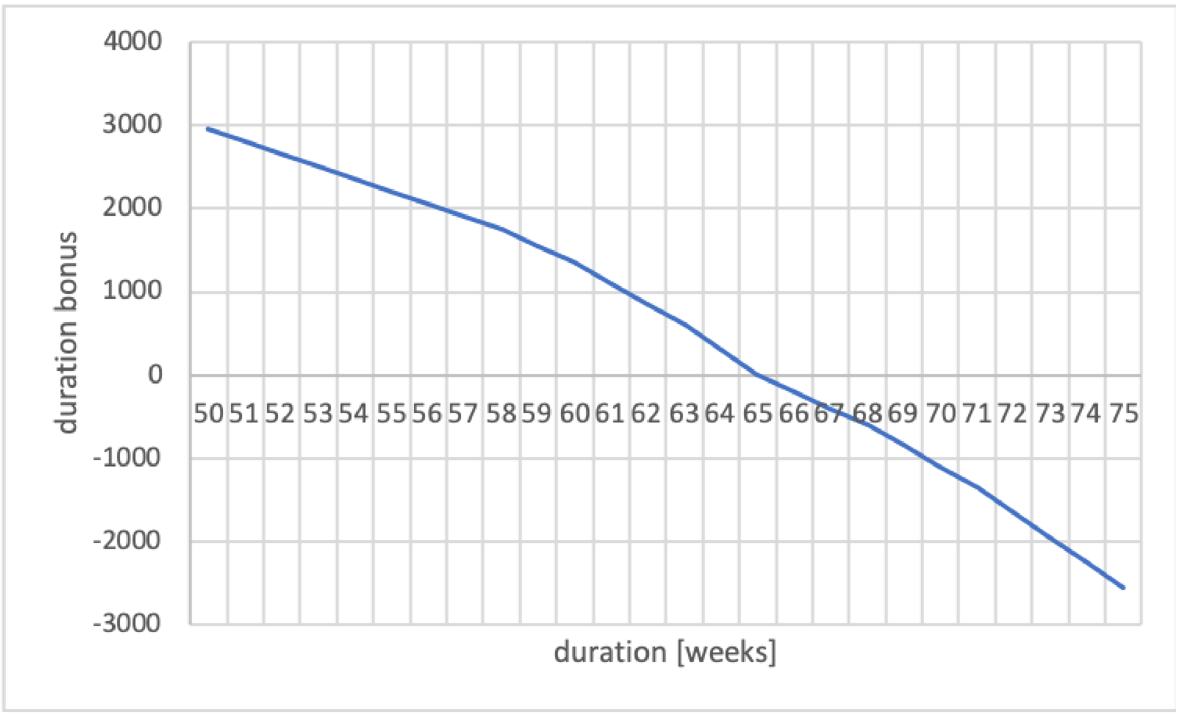


Figure 2.3.: TopSim roller coaster project bonus dependent on duration

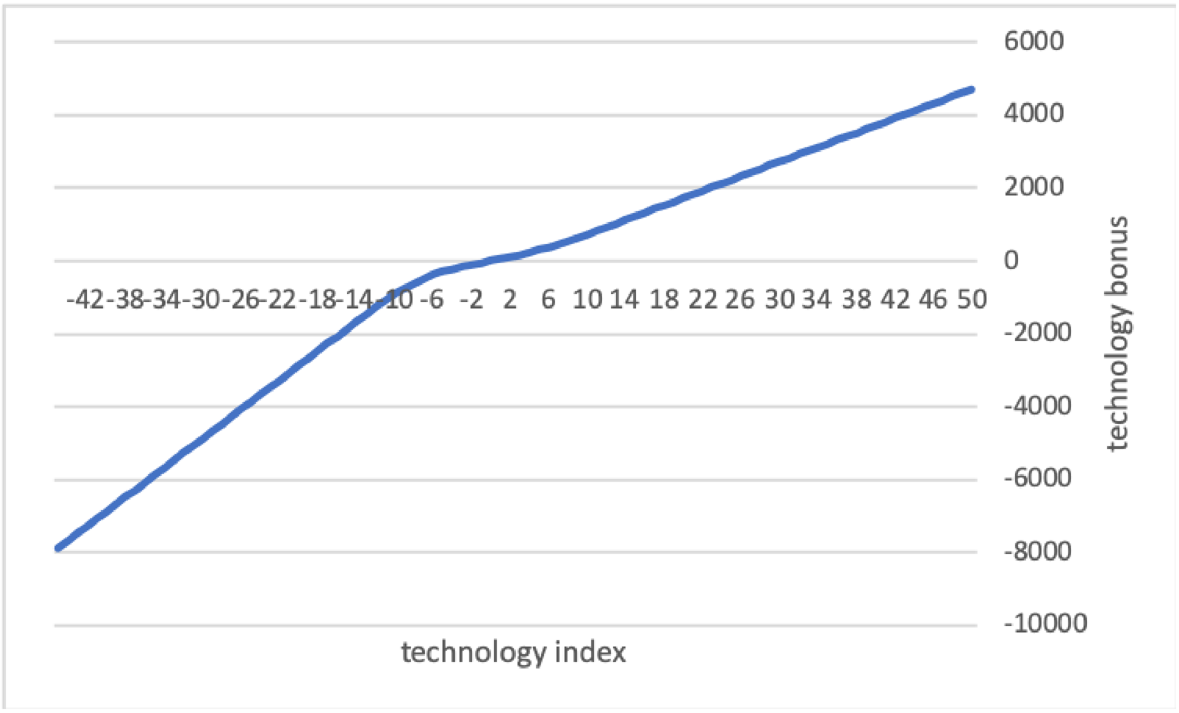


Figure 2.4.: TopSim roller coaster project bonus dependent on technology index

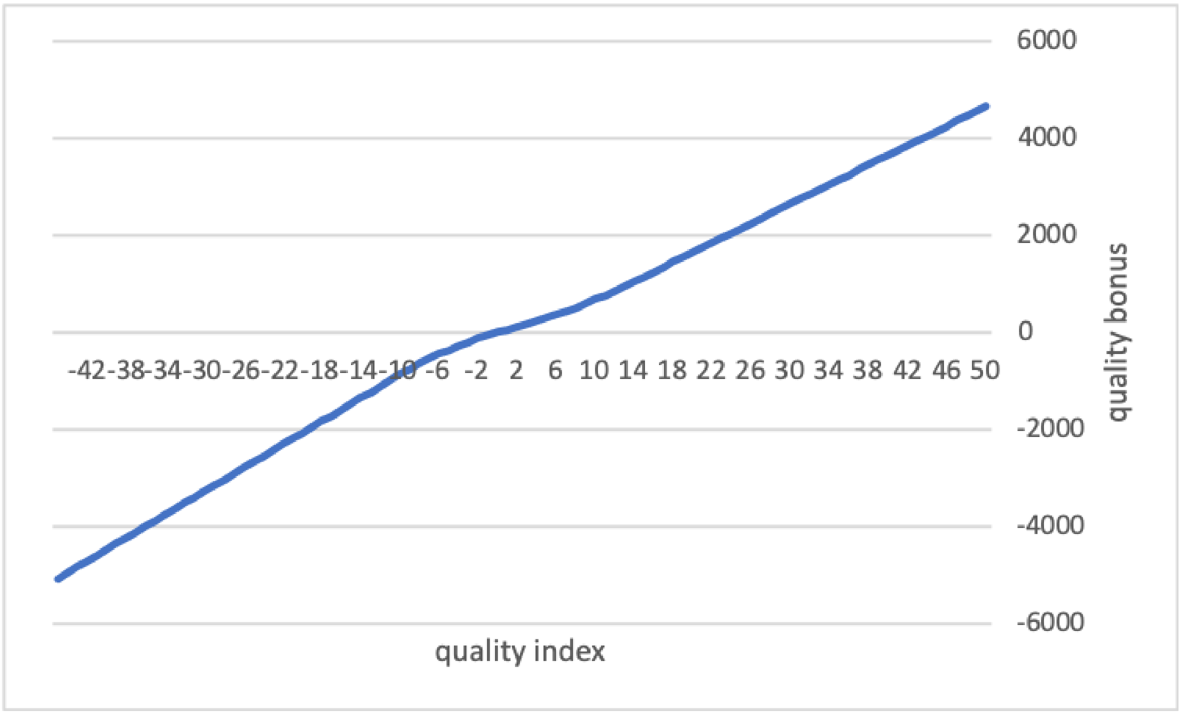


Figure 2.5.: TopSim roller coaster project bonus dependent on quality index

In the model by TopSim there are many events that can occur during the course of the project, mainly dependent on the choice of variants. As these events and their conditions for occurrence are neither publicly available nor were provided by the company due to confidentiality reasons, they could not be implemented. Instead a probabilistic model was introduced in another way: Each activity variant has a so-called 'risk factor' which determines the parameters of the project duration. The distribution is a PERT distribution as already described in 1.4.

$$duration \sim base\_duration * PERT_{\mathrm{RV}}(t_{\mathrm{min}} = \frac{1}{1 + risk}, t_{\mathrm{ml}} = 1, t_{\mathrm{max}} = 1 + risk) \tag{2.3}$$

According to the project handbook [21], variant 0 always had a low risk, variant 1 a medium one, variant 2 the highest risk and variant 3 the lowest, independent of the activity to which the variant belongs. Thus the risk factors were set to 0.1, 0.2, 0.4 and 0.05 respectively.

2.3. MIS project manager project model

The MIS project manager is a project management simulation game developed by Andrew Martin in 2000 for educational purposes [22]. It has a computer-based version allowing players to learn the eventualities that happen in a software management project and how the players respond to them effects the further project steps. It consists of 18 activities from the feasibility study through design, programming and testing to live running as shown in figure 2.6.

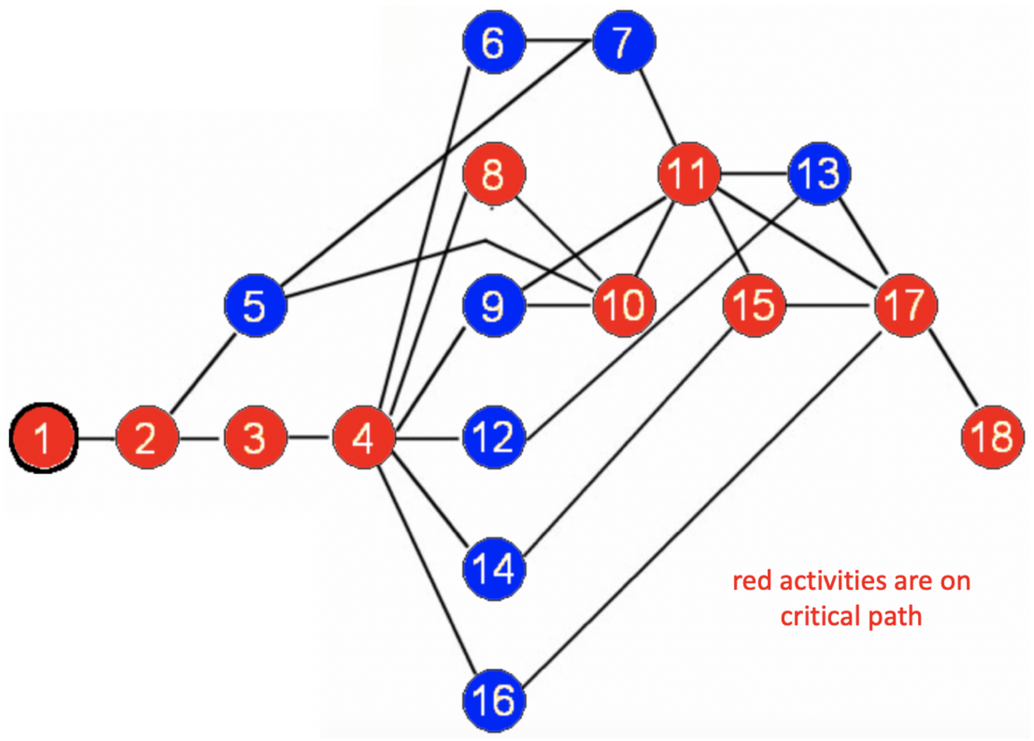
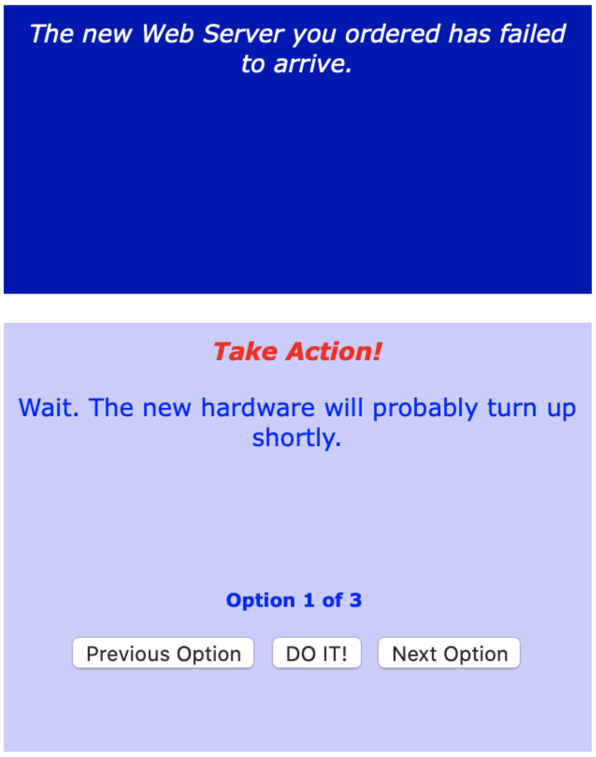


Figure 2.6.: MIS project network plan

Source: [23]

Different from the other two project simulation games, players do not choose between different variants of activities. They rather choose between different reaction variants to events that pop up during the course of the game. These events pop up if certain conditions are met, such as a specific time point, a low security score, a specific reaction to a former event or a combination of multiple of them. In total there are 179 events, each having up to 4 reaction variants. An example event with one of its three reaction variants is shown in 2.7.



The new Web Server you ordered has failed to arrive.

Take Action!

Wait. The new hardware will probably turn up shortly.

Option 1 of 3

Figure 2.7.: MIS project example event

Source: [23]

Besides cost and time there are also the performance indicators of quality, user acceptance, team morale and security. The overall score is defined as the average of all 6 performance indicators, the loss is the negative overall score. The author of the model called activities tasks instead, so these are equivalent in the context of the model.

The conditions which determine if an event pops up are the following :

- TaskTime(taskIndex) </=/> value:
  This condition is fulfilled if the progress on a certain activity is smaller/equal/greater than the specified value.
- TaskOvertime(taskIndex) </=/> value:
  This condition is fulfilled if the time spent on an activity longer than the base duration is smaller/equal/greater than the specified value.
- BoughtResource(resourceIndex):
  This condition is fulfilled if the resource specified by the index has been acquired before.
- ResourceAvailable(resourceIndex):
  This condition is fulfilled if the resource specified by the index has not been acquired before.
- RandomPercent() < value:
  This condition is fulfilled if a random integer from the range of 0...99 is smaller than the specified value.

- ScoreXXX < value:
  XXX may stand for cost, time, quality, acceptance, morale or security. This condition is fulfilled if the current score in XXX is smaller than the specified value.
- Time </=/> value:
  This condition is fulfilled if the total time spent on the project so far is smaller/equal/greater than the specified value.
- TimeLag = value:
  This is not a condition in the strict sense but rather specifies the time between the scheduling of the event by another event and the actual occurrence. As an example: If an event schedules another event at time t using the *action_Event(eventIndex)*-function and the scheduled event has a timeLag of tl, then it is set to have the occurCondition Time == t+tl.

The actions which may happen if an event pops up and an event option is chosen are the following:

- action_Event(eventIndex):
  This action schedules another event specified by the event index to occur after a delay defined by the timeLag-attribute of the other event.
- action_ProjectDelay(noDays):
  This action delays the whole project by the specified number of days. During this time no progress on the activities is made, but events may still pop up.
- action_TaskQuality(taskID,value):
  This action increases the quality of the activity specified by the taskID by the specified value.
- action_TaskCost(taskID,value):
  This action increases the cost of the activity specified by the taskID by the specified value
- action_ScoreXXX(value):
  Here XXX may stand for acceptance, security or morale. This action increases the corresponding score by the specified value, which may be negative.
- action_ProjectCost(value):
  This action increases the indirect project cost which cannot be allocated to any activity by the specified value.
- action_TaskDelay(taskID,noDays):
  This action increases the time needed for finishing the activity specified by the taskID by the specified number of days.
- action_CancelEvent(eventID):
  This action cancels the scheduling of an event.
- action_Budget(value):
  This action increases the total budget available for the project by the specified value.
- action_CurrTaskQual(value):
  This action is just like *action_TaskQuality(taskID,value)*, but it does not apply the quality change to the activity specified by the taskID, but rather to the activity currently worked on.

- action_BoughtResource(resourceID):
  This action acquires the resource specified by the resourceID and adds it to the available resources.

While the scores for user acceptance, security and team morale are manipulated by the events directly, the scores for cost, duration and quality are determined by the functions in 2.4.

$$\begin{aligned}
totalCost &= sum(task\ costs) + projectCost \\
scoreCost &= 50 + \frac{budget - totalCost}{30} \\
scoreTime &= 50 + \frac{330 - totalDuration}{5} \\
scoreQuality &= 50 + mean(task\ qualities) * 10
\end{aligned} \tag{2.4}$$

The MIS project game was played by 40 student groups. Their performance is shown in Table 2.2. Again humans had a disadvantage compared to algorithmic solvers, as they could not gain knowledge of the events and their outcomes implicitly by playing multiple times.

	Overall	Cost	Time	Quality	User Acceptance	Morale	Security
best	62	51	56	75	75	66	65
average	58.9	49.9	50.7	68.8	65.6	58.3	59.8
standard deviation	1.9	0.6	1.8	4.5	5.1	3.6	3

Table 2.2.: Human performance in the MIS project

2.4. Overview of project models

Table 2.3 gives an overview of the three project models. It shows that the roller coaster and MIS model have much more options than the refinery model making it harder to find an optimum. Furthermore the MIS model needs much longer to evaluate as every event occurrence condition must be checked every time step. This makes it harder to find an optimum for algorithms needing many evaluations, while sample-efficient algorithms are nearly unaffected.

model	refinery	roller coaster	MIS
number of activities	25	46	18
variants per activity	3-6	2-4	1
activity variant parameters (additional to duration, cost)	quality	technology, quality, risk	quality
number of events	0	0	179
variants per event			1-4
total number of discrete options	1.78E+18	1.39E+27	3.71E+30
number of variables	25	46	197
number of variables with schedule compression	50	92	215
evaluation time [ms]	0.55	0.16	79.3
performance parameters (additional to duration, cost)	quality	technology, quality	quality, user acceptance, team morale, security

Table 2.3.: Comparison of optimization problems for project models

3. Optimization algorithms

The purpose of all optimization algorithms is to find the minimum of the project model function, i.e. the minimum of function 1.8. The algorithms just get the function as black-box function with the high-dimensional, mixed (categorical and continuous) input space called action space. The number of function evaluations is unlimited and each function evaluation only takes less than one to a few hundred milliseconds, depending on the project model and computer used.

3.1. Criteria for optimization algorithms

Besides being able to solve a black-box optimization problem there are other criteria determining how well an optimization algorithm solves these particular problems:

- 1st criterion: good scaling with dimensionality of input
  As the number of input variables ranges from 25 to 215, the algorithm should scale well with the dimensionality. This makes algorithms with a high complexity as a function of dimensionality computationally very expensive.
- 2nd criterion: good scaling with number of samples
  Because of the huge input space many samples are needed and the algorithms should not become slower after acquiring many samples.
- 3rd criterion: global optimum search
  Often a change of multiple input variables simultaneously is needed to reduce the loss. Thus there are many local minima and the algorithm should be able to overcome them.

3.2. Genetic algorithm

Genetic algorithms are one dialect of evolutionary algorithms. They iteratively find a solution for non-convex black-box optimization problems by using bio-inspired representations like genes and chromosomes and concepts like mutation, recombination and survivor selection according to a fitness function. The typical pseudocode of a genetic algorithm varies a lot between different applications. In this thesis the pseudocode shown in Algorithm 1 is used, which is a slightly adapted version of the one in [24], page 26.

The functions used in the pseudocode do the following:

- generateInitialPopulation(desiredPopulationSize,inputSpace):
  This function generates the population, which is a list of individuals. Each individual is an action being element of the input space and can thus be the input to the objective

Algorithm 1 Pseudocode for a genetic algorithm

```
$pop \leftarrow generateInitialPopulation(desiredPopulationSize, inputSpace)$
while $endConditionNotMet$ do
    $fitnesses \leftarrow fitnessFunction(pop)$
    $breedingPool \leftarrow selectParentsForCrossover(pop, fitnesses)$
    $offspring \leftarrow crossover(breedingPool)$
    $pop \leftarrow mutate(offspring)$
end while
return $fittestIndividualAndFitness(pop)$
```

function. The number of individuals being part of the population is specified by the desiredPopulationSize.

- endCondition:
  The end condition is usually a certain number of iterations, after which the algorithm stops.
- fitnessFunction(pop):
  This function applies the objective function called fitness function from equation 1.8 on every individual in the population and returns a list of fitnesses, one for each individual.
- selectParentsForCrossover(pop,fitnesses):
  This function chooses a subset of the total population having the highest fitnesses (i.e. lowest losses). There are many strategies how to sample the subset: Most implementations have an elite population, which is the subset of the best individuals. The other individuals in the breeding pool may be sampled randomly from the next best group of individuals or a higher probability is assigned to individuals with a higher fitness. The number of individuals being part of the elite population and normal breeding pool as well as the sampling strategy are hyperparameters.
- crossover(breedingPool):
  This function applies crossover on the breeding pool: It chooses randomly two individuals from the breeding pool, crosses them and then adds the offspring to the offspring-list. Again there might be variants of this strategy, e.g. it might be sampled more often from the elite population and/or the elite population becomes part of the offspring unaltered. These strategies and their respective parameters are more hyperparameters.
- mutate(offspring):
  This function mutates every individual in the offspring by randomly changing variables of it. This could be a change of value for categorical variables or the addition of a value sampled from a zero-mean uniform or Gaussian distribution for continuous variables. Some variants of this step apply no mutation or only mutations with a lower mutation probability to the elite population. The use of these variants, the mutation probability and the kind of mutation are hyperparameters.
- fittestIndividualAndFitness(pop):
  This function returns both the highest fitness and the individual having it.

As this genetic algorithm's complexity is linear in dimensionality of the input and number of samples/iterations, it fulfils the first two criteria in 3.1 very well. Global optimum search can only happen randomly and is not specifically fostered, thus the third criterion is not fulfilled well.

3.3. Bayesian Optimization

Bayesian Optimization is a prominent optimizer for global optimization for gradient-free black-box functions. In the following it is assumed that the problem is a minimization problem. Maximization problems can be translated to minimization problems by multiplicating the objective function with -1 and vice-versa.
Bayesian optimization models a probability distribution M over the estimated black-box function given a set of data points D. The conditional distribution given the data, P(M|D), is calculated via the Bayes formula in equation 3.1.

$$\begin{aligned} &P(M|D) = P(D|M) * P(M)/P(D) \propto P(D|M) * P(M) \\ &with\ M = model\ of\ black\text{-}box\ function \\ &and\ D = data\ acquired\ so\ far = \{(x_1, f(x_1))...(x_{t-1}, f(x_{t-1}))\} \end{aligned} \tag{3.1}$$

The model M is usually a Gaussian distribution over functions, which is parametrized by a mean function $\mu(x)$ and standard deviation function $\sigma(x)$. The exact formulas how to calculate them out of the Data D are described in [25], page 6 ff. To decide which point to evaluate next, an acquisition function is needed which is maximal at a point with both a good estimated value and high estimation uncertainty, allowing a balance between exploitation and exploration of the input space. Commonly used acquisition functions are the following, which are taken from [25], page 11ff:

1. probability of improvement (PI):
   This acquisition function returns the probability at every point that evaluating it by the black-box function, will yield a better point than the currently best point x_{best}:

   $$\begin{aligned} PI(x) &= P(f(x) < f(x_{\text{best}})) \\ &= ncdf(\frac{f(x_{\text{best}}) - \mu(x)}{\sigma(x)}) \end{aligned} \tag{3.2}$$

   with ncdf being the normal cumulative density function.

2. expected improvement (EI):
   By defining the improvement $I(x) = max(0, f(x_{\text{best}}) - f(x))$, the expected improvement can be calculated with following function:

   $$\begin{aligned} &EI(x) = \mathbb{E}(max(0, f(x_{\text{best}}) - f(x))) \\ &= \begin{cases} (f(x_{\text{best}}) - \mu(x)) * ncdf(Z) + \sigma(x) * npdf(Z) & \sigma(x) > 0 \\ 0 \end{cases} \\ &with\ Z = \frac{f(x_{\text{best}}) - \mu(x)}{\sigma(x)} \end{aligned} \tag{3.3}$$

   Again ncdf and npdf denote the normal cumulative respectively probability density function.

3. lower confidence bound (LCB):
   The lower confidence bound can be calculated easily out of the model given the scalar hyperparameter $\kappa \in \mathbb{R}^+$:
   $$LCB(x) = \mu(x) - \kappa * \sigma(x) \tag{3.4}$$
   When using it the point having the lowest LCB is chosen next.

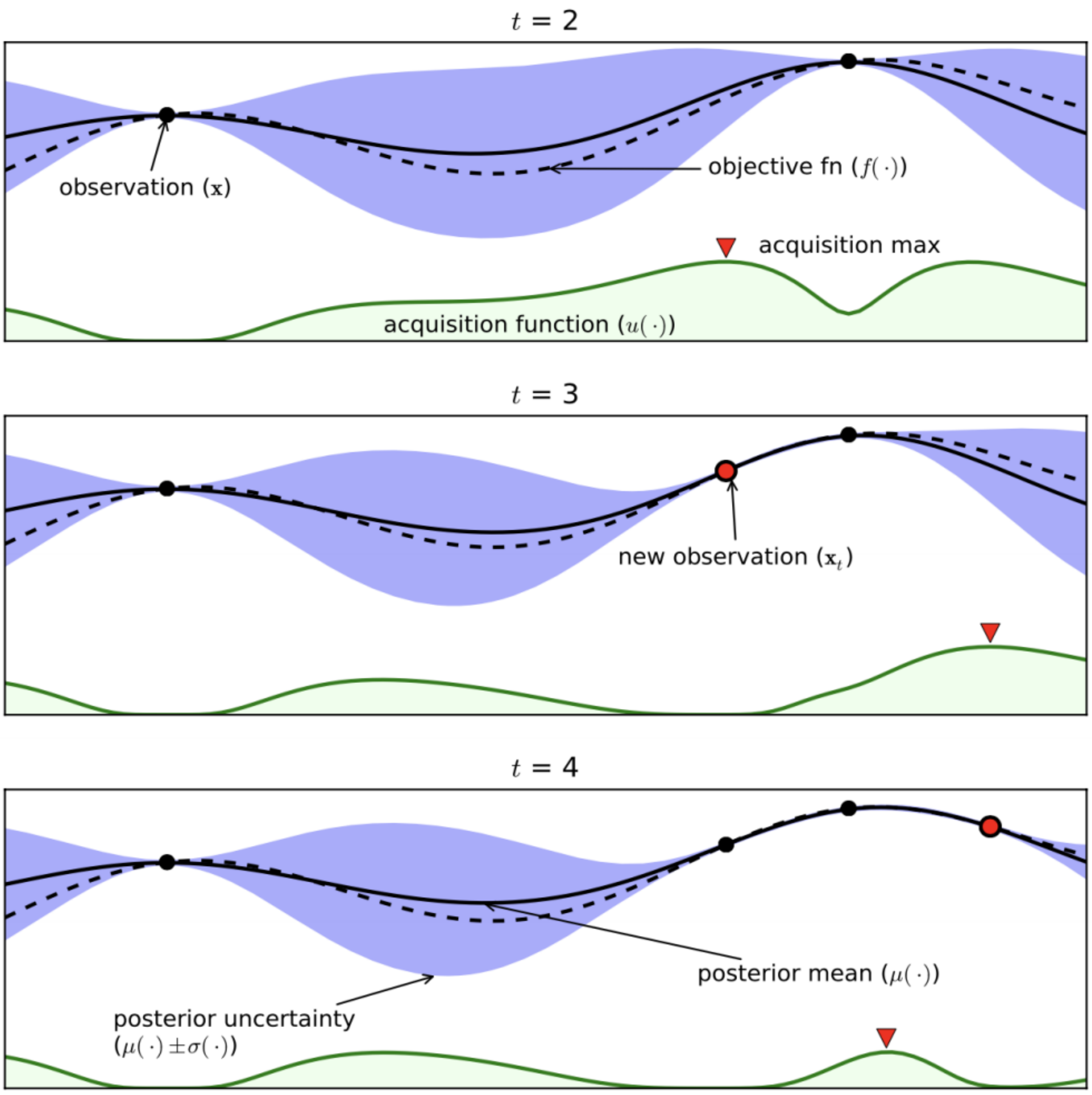


Figure 3.1.: Example for steps of Bayesian optimization

Source: [25]

An example how the model and acquisition functions are used is given in Figure 3.1. The first picture shows the model on a problem with one continuous-valued variable and already 2 observations. The black curve is the mean $\mu(x)$ of the estimated curve, the violet curves show the upper and lower confidence bounds given by the standard deviation $\sigma(x)$. In the example, the goal is to maximize, thus the green acquisition function is largest near the largest point sampled so far. At the maximum of the acquisition function the next point (depicted in red) is sampled, causing the model to update, as shown in the next picture. These steps are repeated till an end criterion is reached, which is also shown in the pseudocode in Algorithm 2.

Algorithm 2 Pseudocode for a Bayesian Optimization algorithm

```
$initialObservations \leftarrow [actionSpace.sample()\ for\ i\ in\ range(numberInitialData)]$
$D \leftarrow set([(x, f(x))\ for\ x\ in\ initialObservations])$
**while** $endConditionNotMet$ **do**
    $P(M|D) \leftarrow P(D|M) * P(M)$
    $x \leftarrow argmax_x(acquisitionFunction(x|M))$
    $f(x) \leftarrow objectiveFunction(x)$
    $D \leftarrow D \cup \{(x, f(x))\}$
**end while**
**return** $bestXandValue(D)$
```

Calculating the model via the Gaussian process needs an inversion of its covariance matrix, which has a cubic complexity w.r.t. the number of samples in D. Finding the minimum of the acquisition function also has a high complexity, usually also w.r.t. the dimensionality of the input space, but this is dependent on the kind of optimizer used. [26] As both of these operations scale with higher than linear complexity as a function of dimensionality and number of samples, the first two criteria from 3.1 are not fulfilled well.
On the other hand, compared to the other optimization algorithms, Bayesian optimization is very sample efficient, even though this advantage is mostly irrelevant here because all project models are cheap to evaluate. Furthermore Bayesian optimization actively fosters exploration through the acquisition function by evaluating points with a high estimation uncertainty, making it fulfil criterion three from 3.1 very well.

3.4. Actor-Critic algorithm

Actor-Critic methods were first proposed by Barto et. al in 1983 for solving complex reinforcement learning problems. [27] In the reinforcement learning problem there is an environment which maps the current state and the action by the agent to a new state and a reward. A probabilistic agent acts in this environment and assigns a probability to every action given a state with the so-called policy function $\Pi(a|s)$. The agent's goal is to maximize the cumulative discounted reward R by finding the optimal policy Π^* as shown in equation 3.6. The functions needed to find it are shown in equation 3.5, their relationship in Figure 3.2.

$$\begin{aligned}
&environment : S \times A \rightarrow S \times \mathbb{R} \\
&agent's\ policy\ \Pi(a|s) : S \times A \rightarrow [0;\ 1] \\
&cumulative\ reward :\ R = \sum_{t=1}^{\infty} \gamma^t * r_t \\
&with\ discount\ factor\ \gamma \in (0;\ 1) \\
&and\ r_t\ =\ reward\ at\ step\ t \\
&and\ r_t \in \mathbb{R}, a \in A, s \in S
\end{aligned} \tag{3.5}$$

$$optimal\ policy :\ \Pi^*\ = argmax_{\Pi}\ \mathbb{E}[R] \tag{3.6}$$

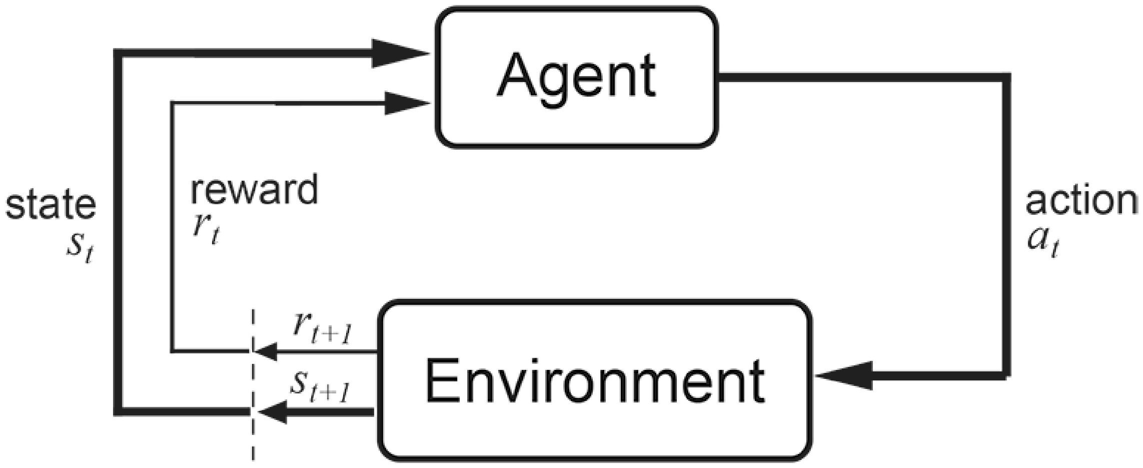


Figure 3.2.: Model of reinforcement learning

Source: [28]

Finding the optimal policy Π^* is non-trivial, as the policy is defined step-wise while the cumulative discounted reward R is defined for an epoch being a row of steps. Often rewards for actions are delayed or only a combination of specific actions in specific states yields a reward. Thus it is hard to find out if a change of the policy for one state is good, as this needs multiple epochs. This problem is called the credit assignment problem and was made popular by Minsky in 1961 [29].

3.4.1. General actor-critic algorithm

The actor-critic method tries to solve the reinforcement learning problem by splitting it into two parts: The critic models the expected reward via the parametric action-value function Q(state,action) and value function V(state). They are both dependent on the current state and policy, the Q-function is additionally dependent on the action taken in the current state.

$$\begin{aligned} Q(s,a) &= \mathbb{E}[R|s,a,\Pi] \\ V(s) &= \mathbb{E}[R|s,\Pi] \end{aligned} \tag{3.7}$$

Combining equations 3.6 and 3.7 yields to equations 3.8. Subtracting V(s) is possible, as the policy is applied on a current state, thus subtracting the constant w.r.t. the action does not change the argmax.

$$\begin{aligned} \Pi^* &= argmax_{\Pi}\ \mathbb{E}[R] \\ &= argmax_{\Pi}\ Q(s,a) \\ &= argmax_{\Pi}\ (Q(s,a) - V(s)) \end{aligned} \tag{3.8}$$

The actor is the second component of the actor-critic algorithm. It models the parametric policy function from which the action is sampled from and which is updated to maximize the expected reward. In this thesis the advantage actor-critic algorithm is used. Its policy increases the probability of an action, if this action given the state is better than the other actions in the state, i.e. it has a positive advantage. Mathematically the advantage A(state,action) can be defined as the difference between the action-value function Q(state,action) and the value-function V(state). The relationship between the components is depicted in Figure 3.3.
Mathematically the two components can be seen as following: The critic solves a supervised learning problem by estimating the value functions in equation 3.7 as good as possible. The actor solves the problem of finding a function maximizing an objective function, which is written down in equation 3.8.

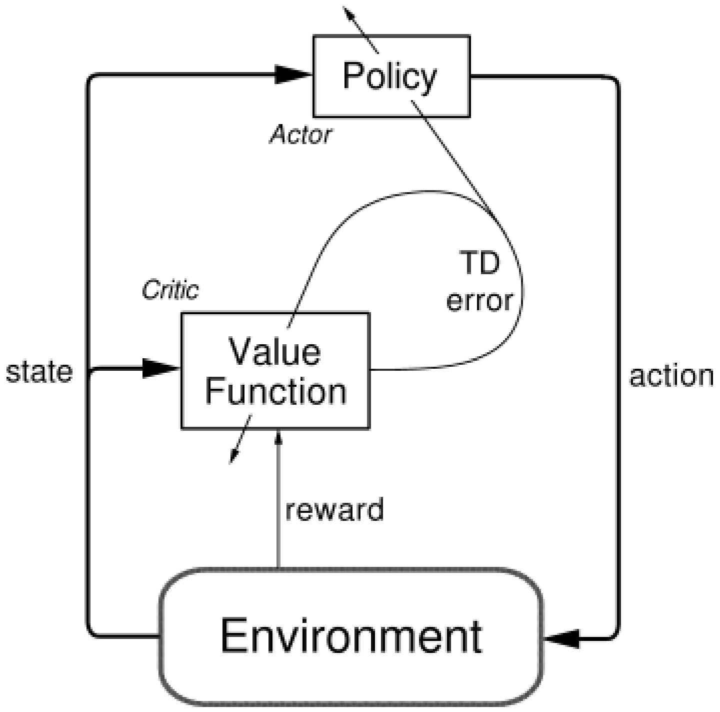


Figure 3.3.: Relationship of components of actor-critic algorithm

Source: [30]

The general pseudocode for the advantage actor-critic algorithm is shown in Algorithm 3. It differs from other pseudocodes for the actor-critic algorithm as it makes the assumption, that the V-function cannot be calculated out of the Q-function through the Bellman equation as the action space is too big. Thus both the Q-function and V-function have to be learned.

Algorithm 3 Pseudocode for the general actor-critic algorithm

```
$state \leftarrow environment.startingState$
while $endConditionNotMet$ do
    $action \leftarrow policy(state).sample()$
    $(newState, reward) \leftarrow environment.step(action)$
    $QFunction.update(action, state, reward, VFunction(newState))$
    $VFunction.update(state, reward, VFunction(newState))$
    $advantage \leftarrow QFunction(state, action) - VFunction(state)$
    $policy.update(state, action, advantage)$
end while
```

3.4.2. Adaptation to function optimization problem

While reinforcement learning problems more generally try to find a policy dependent on a state so that a long-term reward is maximized, any function optimization problem can also be treated as a special case of reinforcement learning, having only one state and each epoch being only one step long. Thus following functions get an easier form:

- policy function:
  The policy function $\Pi(s,a)$ assigning a probability to each state-action-pair now only assigns a probability to each action $\Pi(a)$.
- value function:
  Instead of assigning a value to each state v(s), the value becomes a scalar being the expected loss under the policy. This value, also called baseline, can be estimated as the output of a low pass filter applied on the stream of losses.
- action-value function:
  This function Q(s,a) assigning a value to each state-action-pair becomes a function Q(a) assigning a value to each action. It is even possible to use no Q-function at all, as the project model already gives the true cumulated reward for each action. The credit assignment problem does not exist here.

The pseudocode of the actor-critic algorithm can be adapted accordingly as shown in Algorithm 4. The relationship between the components is graphically depicted in Figure 3.4.

Algorithm 4 Pseudocode for the adapted actor-critic algorithm

```
while $endConditionNotMet$ do
    $action \leftarrow policy.sampleAction()$
    $reward \leftarrow environment.step(action)$
    $VValue.update(reward)$
    $advantage \leftarrow reward - VValue$
    $policy.update(action, advantage)$
end while
```

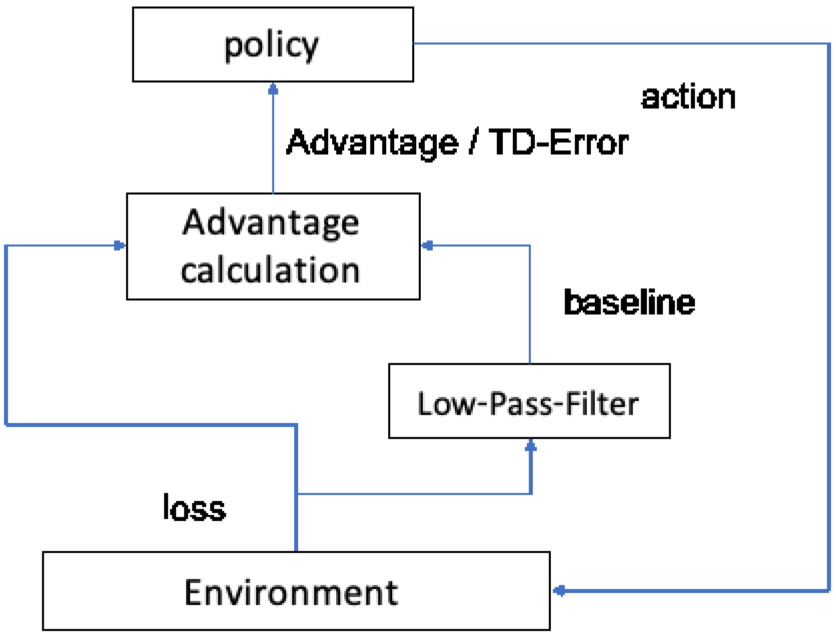


Figure 3.4.: relationship of components of adapted actor-critic algorithm

The parametrized policy function can be modelled such that sampling an action from it and updating the probability of an action is done in linear time as a function of the dimensionality of the action space. The duration of each iteration of the algorithm stays approximately the same, thus the complexity is also linear in the number of samples. Just like for the genetic algorithm, the first and second criterion of the three criteria defined in 3.1, are fulfilled very well. The third criterion of global search or exploration cannot be fulfilled as well as by Bayesian Optimization. To reduce this disadvantage following strategies were used for exploration:

1. probability distribution over variables:
   The policy does not output a fixed action, but rather a probability distribution over the action space, from which an action is sampled randomly.
2. discouraging exploitation:
   By subtracting an exploration factor from the advantage, the advantage may become negative. This reduces the probability of the currently sampled action, increasing the probability of asimilar actions. The exploration factor is decayed exponentially till 0 to allow convergence.

3.5. Heuristic Optimizer assuming independent variables

The heuristic optimizer is not an optimizer in the strict sense. Furthermore it only works for the basic project models having activity variants to choose from, but without events and schedule compression factors. It assumes that the choice of activity variants is independent from each other. Then it finds the optimum under this assumption by treating every activity as a whole project model. The best variant of this mini-project can be easily found by trying all variants. Last all these optima per activity are stacked together to the optimum for the whole project.

3.6. Re-parametrization of action space with variational autoencoder

A re-parametrization of the high-dimensional categorical and continuous action space by a variational autoencoder was used to increase the performance of the optimization algorithms.

The action space has three properties making it difficult for some optimization algorithms to find an optimum:

- high dimensionality
  The high dimensionality of the action space increases computation time a lot for algorithms with a high complexity with respect to the search space dimensionality, like the Bayesian optimization algorithm. This problem becomes obvious when applying the Bayesian optimization algorithms on the MIS project model with the huge variable space, as it causes computation time per step to increase a lot compared to the other project models.
- categorical parameters
  Categorical parameters do not allow to learn gradients empirically, making algorithms using them infeasible. Furthermore artificially small steps are not possible to make.

- many local minima
  Very often changes of many parameters simultaneously are needed to reduce the loss. As an example: If the variant chosen for an activity changes, also the schedule compression factor must change, as a certain schedule compression factor is only best for a certain activity variant. This problem became obvious when applying the optimization algorithms on a project model after adding schedule compression factors to the search space: It was expected that the loss should be reduced, as there is an additional possibility to reduce project duration. Nonetheless some algorithms even performed worse.

The ideal action space would not have the listed shortcomings, i.e. it would be a low-dimensional action space with only continuous parameters and only few closely connected local minima.

A variational autoencoder, which was first presented by Kingma and Welling in 2013 [31], allows to represent a high-dimensional space by a continuous-valued space with a defined dimensionality called the latent space. This is done by using two neural networks: the encoder mapping from the action space to a Gaussian distribution in the latent space and the decoder mapping from the latent space to the action space. They are trained with two goals: One loss is the reconstruction error between the action output of the neural network and the action fed into it, i.e. actions fed through the encoder and decoder should remain as unaltered as possible. The second loss is the Kullback-Leibler-Divergence between the distribution in the latent space and a standard Gaussian distribution with zero mean and unit variance. This loss ensures that the latent action space is smooth and samples lie closely together. Put together the variational autoencoder reduces the action space to a low-dimensional continuous and smooth space, overcoming the shortcomings listed above.

As the action space is defined such that all variables are independent from each other, it is impossible to find a lower-dimensional representation for the whole action space. Thus only a subset of the original action space consisting of good actions with low losses should be represented by the variational autoencoder. Three strategies to define this subset were identified:

- random sampling of actions, training on samples with low loss
  Samples are drawn randomly from the action space, then the samples with the lowest loss, e.g. the samples with a loss under a certain threshold, are used to train the autoencoder. This is implemented easily and computationally very cheap.
- taking samples from former optimization runs
  The optimization algorithms so far already create a lot of samples. These samples are mainly ones with a good loss, as the optimization algorithms exploit and sample at points expected to have a low loss. Thus the variational autoencoder can be trained on all samples generated in former optimization runs, ideally of all algorithms. This method has the disadvantage, that all samples must be stored and read. Furthermore it needs a lot of optimization runs to generate the samples before the autoencoder can be trained, making it computationally very expensive.
- training autoencoder while optimizing
  The autoencoder can also be trained on-the-fly while the optimization algorithm runs. While this is easy to implement and computationally quite cheap, it suffers from the small number of samples available to train the algorithm.

As the first and third strategy are implemented easily and computationally cheap, both of them were implemented and tested both alone and together.

4. Implementation

The implementation is generally split into four parts:

- Implementation of project models
  First a general project model is provided which includes the basic structure of every project including activities, activity variants, their connections, events, event options and the simulation functions. Every specific project model inherits from the general model and further specifies its components.
- Interface between project models and optimization algorithms
  Both the input to all project models and the search space of the optimization algorithms are clearly defined via action spaces and actions represented as instances of classes. This makes it easy to run every optimization algorithm on every project model.
- Interface with variational autoencoder
  The interface with the variational autoencoder may replace the original interface and re-parametrizes the action space which the optimization algorithms optimize over.
- Implementation of optimization algorithms
  The optimization algorithms are either provided by third-party libraries or implemented in a separate class. Then a script can be run, which applies the optimization algorithm on a project model.

The programming language chosen is python as it is a general purpose high-level programming language. Furthermore its library Keras [32] allows an easy implementation of the multi-output neural networks needed for the actor-critic optimization algorithm and the variational autoencoder.

4.1. Implementation of project models

Even though all project models already had an implementation in some form, they were re-implemented in python as their implementation was in form of an excel spreadsheet, program with a GUI or website, which could not be accessed both easily and with a short calculation time. The data needed for defining the model in python is provided in excel spreadsheets to make it both easily understandable and manipulable by humans.

4.1.1. Implementation of general model

The general project model consists of following classes:

- Model_general
  The attributes of this class include all activities, events, model options and the functions how to calculate the performance indicators and loss of a simulation run. Furthermore it contains the method to run a simulation given an action and return the loss. Each simulation run includes the simulation of all activities with their respective activity variants and schedule compression factors. It further includes the run of all events whose occurrence conditions are met with their respective event reaction variants.
- Activity
  The class *Activity* contains its predecessor activities and all activity variants. An activity can only be started if all predecessor activities are finished.
- Variant
  The class *Variant* contains the function to simulate the run of an activity variant during which the activity's cost, duration and possibly other parameters like quality are determined.
- Event
  The class *Event* contains its occurrence condition and event reaction options. When an event fires because its occurrence condition is true the event option chosen is run.
- EventOption
  The class *EventOption* contains the function to run when the event fires, which may have effects on the duration or cost of an activity or other parameters or cause another event to fire with a certain probability.

4.1.2. Implementation of Refinery Model

The Refinery Project Model was developed by Moraes and Batista [11]. An overview of the model was given in 2.1. The model was implemented by Moraes and Batista in form of an excel spreadsheet. The relevant data of this spreadsheet is read to instantiate the model in python. This is done in following steps:

1. Reading of activities
   All activities with their predecessor activities, base duration, base cost and discipline are listed in one sheet. They are read and stored as instances of the class *Activity_Refinery*, which inherits from the activity class of the general model.
2. Reading of suppliers
   Next the 11 suppliers which may have competencies in 10 different disciplines are read. Each competence consists of a cost efficiency index (CEI), duration efficiency index (DEI) and quality index (QI). They are saved as instances of the class *Supplier* having several competencies.
3. Generation of activity variants
   The activity variants are generated in form of instances of the class *Variant_Refinery* inheriting form the *Variant* class of the general model. Their constructor needs the corresponding supplier. The *Variant_Refinery* contains the method *simulate* to calculate the startpoint, endpoint, duration, cost, and quality of the activity.

4. Filling the activities
   Next the *Variant_Refinery* instances are saved as a list in the *Activity_Refinery* instances. Furthermore the predecessor activities, previously stored as indices, are replaced by pointers to the real predecessor activities.

5. Ordering of activities
   Last the activities are ordered such that all predecessor activities of an activity are listed before that activity. This allows to run the simulation on all activities in order without checking each step if all predecessor activities are finished yet.

4.1.3. Implementation of TopSim roller coaster model

The roller coaster project model was developed by the company Tata Interactive Systems under the brand TopSim [21]. An overview of the model was given in 2.2. Their publicly available handbook provides the information about the activities and activity variants, which was extracted and written down in an excel spreadsheet to make it easily readable by the python program. The steps for reading the data from the spreadsheet are very similar to the ones for the refinery model 4.1.2: Again the activities and activity variants are stored in model-specific classes which inherit from the general model. The simulate function of the *Variant_RollerCoaster* class may sample the activity duration from a PERT distribution as described in equation 2.3 , if the model options are set to run the probabilistic setup.
This model includes a financial bonus-malus system dependent on the project's duration, technology and quality defined by a look-up table for the functions shown in 2.3, 2.4 and 2.5. Reading the look-up table from the excel spreadsheet and calculating the final profit and loss is implemented in a separate class.

4.1.4. Implementation of MIS Model

The MIS project manager is a project management simulation game developed by Andrew Martin [22]. An overview of the model was given in 2.3. The definition of the model's data was exported from a Microsoft Access Database to an excel spreadsheet to make it easily readable and manipulable.

Reading the activities is implemented similar to the models before. This model does not have any activity variants, but at least one variant is needed to create the general model. Thus one single dummy variant is created for each activity. Andrew Martin called the activities tasks instead, thus these two are equivalent in the context of this project model. The activities, activity variants, events and event options are again specified in classes inheriting from the one of the general model.

As the occurrence conditions of the events are written down as strings and may contain brackets or multiple conditions combined, functions similar to compilers using regular expressions are needed to translate the strings to functions returning a boolean value. As an example, the condition "TaskOvertime(3) > 2" is first read as string and then split into the substrings "TaskOvertime", "3", ">" and "2" by using a regular expression. These substrings are called *attributeString*, *indexString*, *comparisonString* and *valueString* respectively. The *indexString* and *valueString* are simply cast to integers, and the attribute function can be gained with the attribute string by using the *getattr()*-function. Combined this yields to the following expression to yield the condition function: "*lambda model: getattr(model,attributeString)(index) > value*" which is an anonymous function. It takes the model as argument, calls the method of the model defined by the *attributeString* with the index as argument, then compares the return value with the specified value and returns a boolean whether the condition is fulfilled or not. A condition might even be a combination of multiple basic conditions, an example would be the condition "TaskTime(8)=37 AND ScoreMorale<55". To transform this string into a condition, the *conditionString* is first split into basic *conditionString*s, on which the steps above are applied. The final *occurCondition* is then a function returning True only when all basic *occurConditions* return True. All functions defined by the *attributeString* are implemented as methods of the project model class. An example would be the condition "TaskOvertime(3) > 2" for which the method *TaskOvertime(activityIndex)* is implemented to return the overtime of the task specified by the activity index.

The actions to run when an event option is chosen are also specified in strings. Thus again translation functions are needed compiling functions out of the string. An example action of an event option would be "TaskCost(4,3)" increasing the cost of the task with index 4 by 3. The actions are implemented as methods of the model. A complex action is the combination of many basic actions of which some are only chosen under certain conditions. An example for such a complex action with the if-condition, then-actions and else-actions each specified as one string is shown in Table 4.1. All of the strings are translated separately into their corresponding functions and then the final action function combines all functions with the if-then-else rule.

IF	"ScoreSecurity>51"
THEN	"Event(53) ProjectCost(10) ProjectDelay(3) ScoreSecurity(3)"
ELSE	"Event(55) ProjectCost(10) ProjectDelay(3) ScoreSecurity(3)"

Table 4.1.: MIS project: example for if-then-else action

4.2. Implementation of model-optimizer interface

The model-optimizer interface between the optimization algorithms and project models defining the objective function serves four purposes:

- Generation of Model
  The *generateModel* function together with the *Model_options* class allows the optimization scripts to generate a project model easily.
- Definition of *ActionSpace* and *Action*s for optimization algorithms
  Each optimization problem defines the kind of input space to optimize over in form of an action space. This includes the number and kind of categorical and continuous variables and a method to sample random actions. This enables the optimization algorithm to correctly define the actions they input to the interface.

- Definition of *ProjectActionSpace* and *ProjectAction*s
  The project models provide a project action space defining how they need their input and can expect to only receive instances of the *ProjectAction* class as input to their simulation function.
- Easy implementation of action space re-parametrizations
  The action space the optimization algorithms optimize over is not necessarily the same as the action space needed for the project model. This makes it easy to implement an interface re-parametrizing the action space and mapping from the optimization action space to the project action space.

Together this enables it to add new project models, optimization algorithms or re-parametrization interfaces without needing to manipulate other parts of the code. The relationship between the parts of the interface is shown in 4.1.

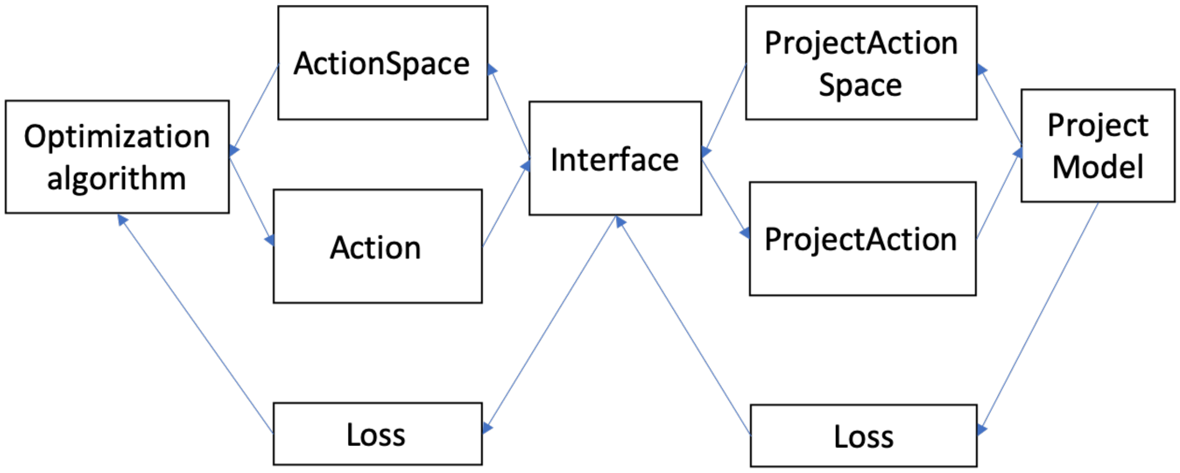


Figure 4.1.: Interface between optimization algorithms and project models

Both the class *ActionSpace* and the class *ProjectActionSpace* inherit from the class *spaces.Tuple* of the library OpenAI-Gym by Brockman et al.[33]. This library defines various spaces such as discrete, continuous and combined spaces and already provides functions to sample from them as methods. The *spaces.Tuple* class can combine a tuple of these simpler spaces. The *ProjectActionSpace* class always consists of one *spaces.multi_discrete* class for defining the space of a project's activity variants, another one for the event reaction variants and possibly a *spaces.Box* class for continuous and bounded variables like the schedule compression factors.

Both the class *Action* and the class *ProjectAction* store the values of the action in the attribute *valuesList*. Furthermore the *Action* class contains methods to one hot encode the discrete values and to check if the values stored are within the range defined by the action space.

The class *DefaultInterface* and any other interface, as it inherits from the *DefaultInterface*, provide following methods: They inform the optimization algorithms about the action space to search in with the *getActionSpace*-method, and they have various forms of the *simulate*-method, which take one action or a batch of actions as input, translate them to project actions, run the simulation to return the performance and then return either the whole performance tuple per action or just the loss.

4.3. Implementation of interface with variational autoencoder

The interface with the re-parametrization by the variational autoencoder is implemented in two parts: The variational autoencoder itself is implemented as a stand-alone class and the interface is another class inheriting from the default interface and overwriting some of its methods.

4.3.1. Keras model of variational autoencoder

The variational autoencoder itself was implemented based on the example in the keras library [32]. The input layer, output layer and loss are generated dynamically depending on the *ProjectActionSpace* provided. Besides the encoder-decoder-architecture itself, the *VAE_Model* class implemented also includes a method to map a latent action to an action prediction through the decoder, a method to map this prediction to a *projectAction*, either deterministically or by sampling randomly and a method to train the autoencoder on a batch of actions including the one hot encoding of the actions. As parts of these methods are also needed by the actor-critic algorithm, these shared functions are written in a separate file.

The complete encoder-decoder-architecture as keras model is shown in Figure 4.2 for a fictive project with two activities and no events. The activities have 3 and 6 variants respectively and are represented each with its own input layer. The variants are one hot encoded, thus each activity's input layer has as many input neurons as the number of the activity's variants. The two schedule compression factors (one for each activity of this project) make up the last input layer.
After concatenating the input, the encoding takes place through an intermediate dense layer with 8 neurons, out of which the mean values and natural logarithm of the variance, from which the latent representation are sampled from, are calculated. In this example, the latent vector is 4-dimensional, the number of neurons in the intermediate encoding layer are set to be twice as big, thus 8 in this example. The last layer of the encoder is the sampling layer, which samples a random vector from a Gaussian distribution given the mean and variance values from the layers before.
The decoder takes a latent vector, passes it through the intermediate decoding layer and then computes an action out of it. Again the categorical variables such as the activity and event variants are represented with a one hot encoded output layer each, the activation function used is the softmax function. The schedule compression factors are computed with a dense layer followed by a layer to clip the outputs into the range of [0.75; 1]. The losses used are cross-entropy for the softmax output layers, mean squared error for the continuous output layers and the Kullback-Leibler-Divergence between the Gaussian distribution defined by the mean and log-variance layer and a Gaussian Distribution with zero mean and unit variance.

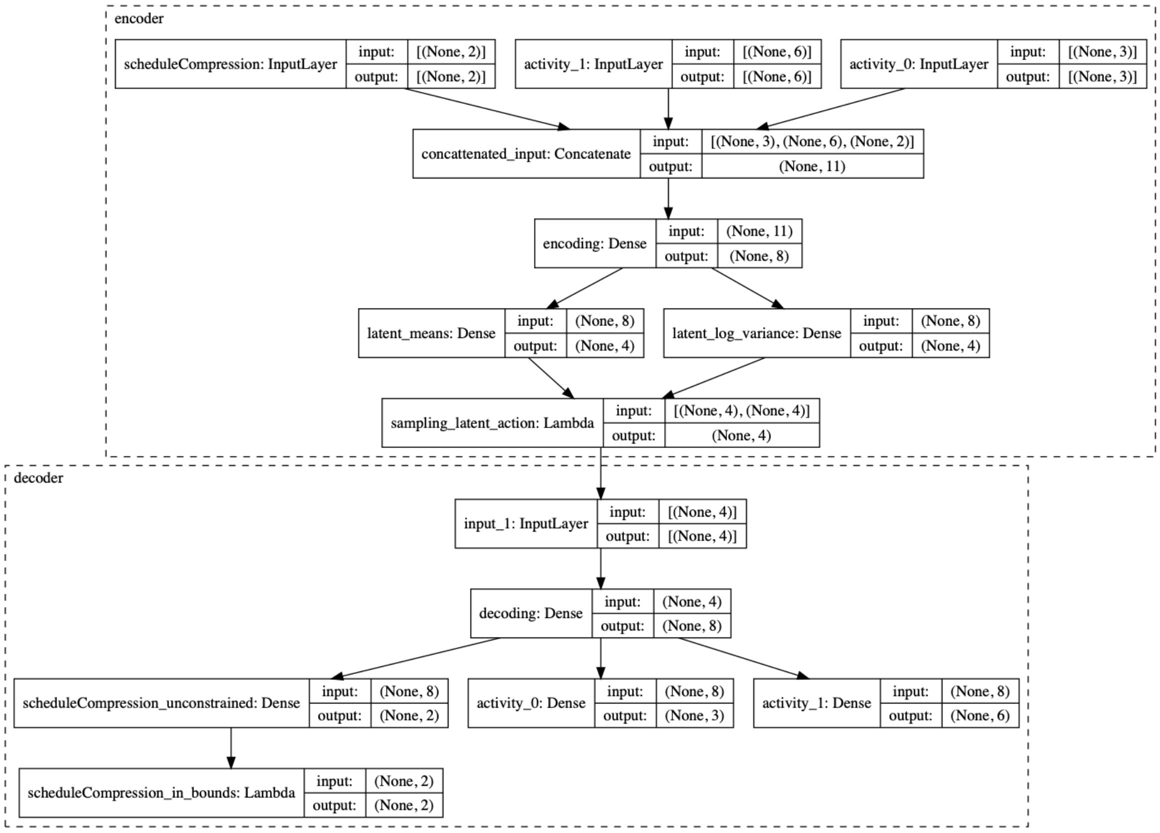


Figure 4.2.: keras architecture of variational autoencoder

For the real project models, the latent dimension was set to 32 for all models except the ones with schedule compression factors, for which it was set to 64.

4.3.2. Incorporation of variational autoencoder into interface

The variational autoencoder is incorporated into the interface between the optimization algorithms and the project models by the class *VAE_Interface*. This class extends the *DefaultInterface* class in the following ways: In its *simulate*-method it first maps the latent action to the *ProjectActionSpace* by using the decoder. Then it runs the yielded *ProjectAction* through the project model and returns the received performance. Furthermore the variational autoencoder may be trained on a subset of the samples provided to the simulate function, depending on the type of training chosen.

4.3.3. Training of variational autoencoder

As described in 3.6, two options to train the variational autoencoder were implemented: The first one trains the autoencoder before the optimization run with a subset of random samples. This option is called pre-training in the following. The second option trains the variational autoencoder online during the optimization run on a subset of the samples the optimization algorithm evaluates.

The pre-training uses the pseudocode shown in Algorithm 5: First a threshold is determined out of 1000 random samples, computed as the mean of the best loss and lower confidence bound of the losses. Next many samples are generated in a loop and the variational autoencoder is trained on all samples having a better loss than the threshold. The pre-training is done in a separate script and the trained VAE is saved in the end. When an optimization algorithm using this VAE runs, the VAE is not trained again, but rather the already saved trained model is loaded. This allows to train the VAE only once for a project model setup and use it for all optimization algorithms on this setup.

Algorithm 5 Pseudocode for the pre-training of the variational autoencoder

```
Require: $actionSpace, projectModel, autoencoder$
    $actions \leftarrow [actionSpace.sample()\ for\ i\ in\ range(1000)]$
    $losses \leftarrow [projectModel.simulate(act)\ for\ act\ in\ actions]$
    $bestL, meanL, stdL \leftarrow min(losses), mean(losses), std(losses)$
    $threshold \leftarrow mean(bestL, meanL - stdL)$
    while $notSufficientlyTrainedYet$ do
        $actions \leftarrow [actionSpace.sample()\ for\ i\ in\ range(batchSize)]$
        $goodActions \leftarrow [act\ for\ act\ in\ actions\ if\ model.simulate(act) < threshold]$
        $autoencoder.trainOnBatch(goodActions)$
    end while
```

The online-training of the variational autoencoder takes place simultaneously with the run of the optimization algorithm. It uses the pseudocode shown in Algorithm 6: As it takes place in the *simulate*-function of the interface, the actions to evaluate and the corresponding losses are already computed. A baseline is defined as output of a low-pass filter applied on the stream of losses and the variational autoencoder is only trained on the action samples with a better loss than the baseline.

Algorithm 6 Pseudocode for the online training of the variational autoencoder

```
Require: $actions, losses, baseline$
    $baseline \leftarrow (1 - baselineFactor) * baseline + baselineFactor * mean(losses)$
    $goodActions \leftarrow [act\ for\ act, loss\ in\ zip(actions, losses)\ if\ loss < baseline]$
    $autoencoder.trainOnBatch(goodActions)$
```

4.4. Implementation of optimization algorithms

The optimization algorithms are all implemented in the following form: The algorithm itself is either a third-party library or implemented in its own class. Then a short script applies the algorithm on the problem and outputs the performance.

4.4.1. Implementation of genetic algorithm

The genetic algorithm is implemented in form of a *GeneticOpt class*. This class handles all parts of the algorithm shown in the pseudocode 1: It generates the initial population and does all steps of the loop:

1. Calculation of losses for each individual
   A list of losses is generated by applying the loss function provided by the project model on each individual of the current population.

2. Selection of breeding pool
   The breeding pool consists of two groups: The best individuals are always chosen and make up the first group of the breeding pool called elite population. The second group consists of individuals chosen randomly from the current population. These individuals are not chosen with equal probability each, but a higher probability is assigned to individuals with a lower loss. These probabilities are calculated by applying a softmax-function to the vector of inverse losses.

3. Crossover and mutation
   The new population consists of three groups: First all individuals of the elite population get into the new population. Second all individuals of the elite population get into the new population after applying a random mutation on them. Third the rest of the new population is made up of randomly chosen individuals of the breeding pool, which are first crossed with other individuals of the breeding pool and then mutated.

Each individual is represented as instance of the class *Chromosome*. As it inherits from the Class *Action*, each chromosome can be used directly as input for the *simulate*-function. The class includes methods for mutating a chromosome and crossing it with another chromosome, which are needed for the breeding mechanism of the algorithm. The mutation is applied independently on each variable of the action space: With a certain probability a categorical variable is mutated by assigning a new value to it. Continuous variables are mutated by adding Gaussian noise with zero mean and a small variance to it. The crossover takes place by replacing the value of each variable of one chromosome with 50% probability by the value of the same variable of the other chromosome.

The script of the genetic optimization algorithm defines the project model and instantiates the *GeneticOpt* class. Then it runs the genetic algorithm by running the *generateNewPop*-method of the *GeneticOpt* class multiple times. This function equals the loop of the genetic algorithm with the three steps listed in 4.4.1.

4.4.2. Implementation of Bayesian optimization

There are already many implementations of Bayesian optimization, so there is no need to implement it again. Instead the libraries Emukit [34], GPyOpt [35] and Hyperopt [36] are used. The implementation of the optimization script for all methods is the same: After generating the project model, the search space and objective function are defined in the way needed by the library's optimization algorithm. The search space has to be defined using classes of the respective library and points to evaluate need to be translated from the format of the library to instances of the class *Action*. Finally the optimization is run until a termination criterion is reached and the performance is recorded and printed.

4.4.3. Implementation of actor-critic algorithm

The actor-critic algorithm was implemented in two files: One for running the algorithm itself according to the pseudocode in Algorithm 4 and one for modelling the policy. Functions needed are one for defining the policy in form of a neural network and the five functions according to the pseudocode:

- policy.defineModel()
  This method defines the neural network representing a parametrized policy and outputting an action. Categorical variables of the action space each have a dense softmax-layer as output space with one unit per variant. Their loss is the categorical cross-entropy, similar to classification tasks. The real-valued schedule compression factors have a dense layer with one unit per factor, followed by a lambda layer cutting off values such that the schedule compression factors are always in the range of [0.75; 1]. The loss used is the mean squared error, similar to regression tasks. The function generateActionOutputLayer(actionSpace) for generating the keras output layers out of an action space is provided in a separate file, as it is also used by the variational autoencoder.
- policy.sampleAction()
  This method allows to randomly sample an action given the current policy. First the neural network predicts the output action. The softmax output of the categorical variables can be treated directly as probability distribution over all variants, from which one variant is sampled. The real-valued schedule compression factors are randomized by adding a uniformly sampled value in the range of [-0.1; 0.1] to the predicted schedule compression factor. If this yields points outside the range of [0.75; 1], the points are clipped into that range. Finally the variants and schedule compression factors are saved in form of an action to feed them directly to the project model.
- environment.step(action)
  This function is the objective function of the project model and is thus already implemented.
- VValue.update(reward)
  This function of updating the V-Value or baseline is done in one line of code, thus no function is implemented for it.
- advantage = reward-VValue
  This is another one-liner.
- policy.update(action,advantage)
  This method applies one gradient descent step on the neural network: it calculates the gradient of the neural network's loss function (cross-entropy and mean squared error) given the action with respect to all parameters of the neural network. The twist is, that this gradient is multiplicated by a scalar weight before applying it. For the categorical variables the weight is simply the advantage scalar. This way good actions having a positive advantage become more likely to be sampled. The higher the advantage is, the higher is the change in probability. When the advantage is negative, i.e. the action is bad, the gradient is multiplicated with a negative value, thus the gradient update step is performed in the opposite direction, making this action more unlikely.

In an earlier version of the algorithm the gradient of the continuous schedule compression factors was treated like the one for the categorical variables and also multiplicated with negative advantage weights. This had lead the schedule compression factors to go to the extrema of the [0.75; 1] range.
The problem was solved by multiplicating the gradient of the schedule compression factors only with the advantage if it is positive, otherwise the gradient is set to zero. The performance of the algorithm increased significantly as a result.

5. Performance of optimization algorithms

The performance of the optimization algorithms was evaluated by running the optimization scripts on a computer with medium hardware (Intel i5-6500 with 4x3.2GHz, Nvidia GTX 1060 6GB for Keras models). Each algorithm was given approximately 2000 seconds of evaluation time. There were in total 16 different combinations of models i.e. objective functions to evaluate the performance, which are listed below. They consist of three different project models, up to three different options per project model and two different interfaces.

The 16 different models are the following:

1. refinery base model
2. refinery base model with VAE as interface
3. refinery model with schedule compression
4. refinery model with schedule compression and VAE as interface
5. TopSim roller coaster base model
6. TopSim roller coaster base model with VAE as interface
7. TopSim roller coaster model with schedule compression
8. TopSim roller coaster model with schedule compression and VAE as interface
9. TopSim roller coaster probabilistic model
10. TopSim roller coaster probabilistic model with VAE as interface
11. TopSim roller coaster probabilistic model with schedule compression
12. TopSim roller coaster probabilistic model with schedule compression and VAE as interface
13. MIS base model
14. MIS base model with VAE as interface
15. MIS model with schedule compression
16. MIS model with schedule compression and VAE as interface

Together with the five optimization algorithms and the hyperparameters of both the optimization algorithms and the variational autoencoder this equals a huge number of combinations to evaluate, thus it would need very long to run every combination of them multiple times. Hence the best combination of hyperparameters was only determined once by hand for each optimization algorithm and the variational autoencoder and then used for every of the following evaluation runs on the different models.

5.1. General findings

There are some findings which are true for all project models:

- The VAE with online-training interleaved with the optimization performed very badly. It can be expected that this is due to the fact, that the training of the VAE changes the objective function for the optimization algorithms. Thus only the pre-training of the VAE was used.
- The three Bayesian optimization algorithms profit from the re-parametrization of the action space by the VAE as long as the model does not include schedule compression factors. The VAE does not help finding an optimum on the model with schedule compression. This can be explained under the assumption that the Bayesian optimization algorithms can handle continuous variables in the action space much better than categorical ones: The re-parametrization to continuous variables helps the Bayesian algorithms more than the unperfect re-parametrization prohibits them from finding a good point. On the other hand, if the action space already contains the continuous-valued schedule compression factors, the gain by the re-parametrization cannot outweigh the fact that the algorithms cannot find the best schedule compression factors directly.
- The genetic algorithm and actor-critic algorithm always perform worse or equal when the action space is re-parametrized by the VAE.
- The genetic algorithm and actor-critic algorithm outperform the Bayesian optimization algorithms in most models. It can be expected that this is due to the much higher number of samples they are able to evaluate in the same time.
- When adding schedule compression to the project model, the genetic and actor-critic algorithm nearly always perform better than without while the Bayesian optimization sometimes perform worse as they cannot handle the increased number of variables well.
- The loss on the probabilistic model is higher than on the deterministic one due to the effect described previously in 1.4.
- Introducing schedule compression factors to the models without the VAE reduced the project duration while increasing project cost.

5.2. Performance comparison on Refinery model

The refinery model only has 25 categorical variables and with schedule compression an additional 25 continuous variables, making it the project model with the smallest action space. Because of this the optimization algorithms converge relatively fast compared to the other project models. The performance was evaluated on four different model variants: the base model, the base model with the variational autoencoder as interface, the model with schedule compression and the model with schedule compression and the variational autoencoder as interface. As the model does not include any probabilistic activities or events, there is no probabilistic variant. The performance indicators used in the following are the loss (defined as the product of project duration and project cost), the duration (in days) and the cost (in R$). The performance of the human players was added for comparison.

The loss of the algorithms on the four different models is shown in Table 5.1, while a complete table including all performance indicators, the computation time of the algorithms and the number of samples they evaluate is given in the appendix in Table A.1.

Model: refinery				
with VAE	no	yes	no	yes
with schedule compression	no	no	yes	yes
Optimizer				
human best	930764848			
human average	1790662190			
heuristic	811353180			
BayesOpt_hyperopt	845264556	952613640	866710000	1042473215
BayesOpt_emukit	1081099264	809758246	1310106050	1409331020
BayesOpt_gPyOpt	813981250	877000150	888042232	1119161197
GeneticOpt	790309738	803089590	718297620	794785876
actorCritic	800762800	800762800	756566415	1439269763

Table 5.1.: Performance of optimization algorithms on refinery model (lower and greener is better)

For the base model without the VAE and schedule compression, the genetic and actor-critic optimization algorithms outperform the Bayesian optimization algorithms and are the only ones to be better than the heuristic. It can be expected that this is due to the much higher number of samples they are able to evaluate in the same time. Re-parametrizing the action space with the VAE helps the Emukit algorithm a lot, it becomes nearly as good as the actor-critic algorithm. The reason for this behaviour is unknown. The other algorithms do not profit or even perform worse.
When adding schedule compression factors to the action space, only the genetic and actor-critic algorithm are able to reduce their loss, all other algorithms perform worse. It seems as if they are not able to cope with the increased number of variables and the many new local minima introduced by them. Introducing the VAE now causes all algorithms to perform worse. The reason might be that in this model the heuristic baseline is very good, which shows that the choice of the best activity variant for each activity is mostly independent from another. Thus there are only very little local minima, making the optimization much easier. The VAE with random global sampling cannot use this at all, thus it is trained on far too many bad points.
In all four setups, the genetic algorithm has performed best and was able to have the highest number of function evaluations in the same time.

5.3. Performance comparison on TopSim roller coaster model

The roller coaster model by TopSim has 52 activities with 2 to 4 variants each. They differ in duration, cost, technology index and quality index. The final performance vector also consists of these four values, out of which the loss (i.e. negative profit) of the whole project is calculated using the tables described in [21] (page 15). The profit, technology index and quality index should be maximized, cost and duration should be minimized. The table showing only the profits i.e. negative losses of the algorithms on the various setups of this model is Table 5.2, while a table showing more information including all performance indicators is given in the appendix in Table A.2 for the deterministic setup and Table A.3 for the probabilistic one.

Model: TopSim roller coaster								
with VAE	no	yes	no	yes	no	yes	no	yes
with schedule compression	no	no	yes	yes	no	no	yes	yes
probabilistic	no	no	no	no	yes	yes	yes	yes
Optimizer								
default	-2425							
heuristic	-10125							
BayesOpt_hyperopt	2250	3760	2735	2582	1097	2739	2726	3969
BayesOpt_emukit	-125	3710	1284	759	163	2764	1773	2472
BayesOpt_gPyOpt	2140	4060	3025	348	2812	2735	2501	3675
GeneticOpt	4175	4350	6288	4869	4184	4047	6221	4690
actorCritic	4760	1565	5930	256	3735	464	5674	2539

Table 5.2.: Performance of optimization algorithms on roller coaster model (higher and greener is better)

The default optimizer always uses the default action variant 0 and is the starting point from which humans playing the game start to optimize. Again as for the refinery model the genetic and actor-critic optimization algorithm are able to outperform the Bayesian optimization algorithms for all setups without the VAE.
Replacing the default interface with the VAE helps the Bayesian optimization to perform better except for the deterministic setup with schedule compression (columns 3 and 4). Adding schedule compression increases the profit of all algorithms in all setups except the deterministic model with the VAE (columns 2 and 4).
The probabilistic model samples the duration of each activity variant from a PERT-distribution depending on the risk of the variant as described in 2.2. As this increases the expected duration and the probabilistic nature makes it harder to find an optimum it can be expected that the algorithms perform worse compared to the deterministic model. This is also found empirically for all setups except the one with the VAE and with schedule compression

(columns 4 and 8).
As there are no human performances for this model as baseline, the algorithms can only be said to perform much better than the default and heuristic configuration. As the cost of the whole project was about 10000 and the algorithms were able to gain a profit of about 4000 on the probabilistic model without schedule compression (column 5, it is closest to the real TopSim model), a profit margin of 40% was achieved. This is much higher than the average operating margin in the construction industry, which is under 10% [37].

5.4. Performance comparison on MIS model

The MIS model has 18 activities with no variants to choose from. Furthermore it has 179 events, making up a total of 179 categorical variables. As these events may have random outcomes, all base model of the MIS project manager are already probabilistic. Besides this the model also needs much longer to evaluate than the other models: About 80 ms compared to less than 1 ms for the two other models.
There are in total 4 setups of this project model: the base model, the base model with the VAE as interface, the model with schedule compression and the model with schedule compression and the VAE as interface. The table showing only the overall performance i.e. negative losses of the algorithms on the various setups of this model is Table 5.3, while a table showing more information including all performance indicators is given in the appendix in Table A.4.

Model: MIS				
with VAE	no	yes	no	yes
with schedule compression	no	no	yes	yes
Optimizer				
human - best	62.00			
human - average	58.93			
heuristic	59.10			
BayesOpt_hyperopt	62.80	63.20	63.35	61.58
BayesOpt_emukit	62.59	62.31	62.02	60.65
BayesOpt_gPyOpt	64.29	64.04	64.48	61.94
GeneticOpt	65.05	65.04	65.1	63.69
actorCritic	65.04	64.18	65.01	64.56

Table 5.3.: Performance of optimization algorithms on MIS model (higher and greener is better)

Again the genetic and actor-critic algorithms outperform the other algorithms and the VAE reduces the performance on the setup with schedule compression. All algorithms perform better than the best human group. Interestingly the heuristic optimizer, which always chooses the first event reaction variant in this model, performs as good as the average human group.

6. Summary

This thesis shows that even complex project management problems including all aspects listed in Table 1.2 except resource allocation can be modelled as a mathematical problem. It provides a modular framework for modelling any kind of project having these properties, allowing to quickly implement project models of new projects or business games. As the interface was kept very general as well, other optimization algorithms can be added easily and the optimization algorithms may also be applied on any other black-box optimization problem given that it defines its input space as instance of the class *Action*.
The project management problems modelled have a huge decision space and the project model itself may be complex and stochastic. Humans have difficulties finding a good solution to such optimization problems and often rely on heuristics and experience. These heuristics may be biased and work badly on new projects and the experience first has to be gained by years of working on projects.
Because of these limitations, algorithmic optimization algorithms offer a way to optimize project management decisions and have shown superior performance than both humans and a heuristic in the implemented project models. This allows to finish projects earlier, cheaper and/or with a higher quality or other advantages.
Concerning the algorithms it was shown that genetic algorithm outperforms all other tested algorithms as it is able to have the highest number of function evaluations in a given time. This aligns with the literature on the subject also using mostly genetic algorithms. The reparametrization by the VAE helped the Bayesian optimization algorithms but never increased the performance of the best algorithm.
Even though algorithms can replace humans for these project management decisions and perform better than them, human experience is still needed: The algorithms need a project model being as accurate as possible, for which humans must make up the project network, add events, and most importantly estimate the duration and cost of activities.
Future research may be conducted in several directions: The algorithms may be applied on more project models, possibly from business games or even project models of real projects. Furthermore the aspect of resource allocation may be added to the projects. On the algorithmic side more advanced algorithms like several different variants of evolutionary algorithms and ensembles may be used to find an even better optimum.

A. Appendix

A.1. German abstract

Das Problem, die Aktivitätsvarianten eines Projektes so zu wählen, dass die Gesamtdauer des Projektes und die Kosten minimiert werden, ist in der Literatur als Discrete Time-Cost Trade-off Problem (DTCTP) bekannt. Dieses Problem wird erweitert und verallgemeinert, indem stochastische Ereignisse und Aktivitätsdauern hinzugefügt werden und eine kontinuierliche Zeitplankompression von Aktivitäten ermöglicht wird, was es zu einem gemischten (kategorischen und kontinuierlichen), probabilistischen und nicht-linearen Optimierungsproblem macht. Diese Arbeit hat zwei Beiträge:
Zunächst zeigt sie, dass Algorithmen zur Lösung dieses generalisierten DTCT-Problems bessere Ergebnisse liefern als Menschen und Heuristiken und vergleicht die Performance von Bayes'schen Optimierungs-, Genetischen und Actor-Critic Algorithmen bei dieser Art von Problemen. Zweitens bietet sie ein Framework für die Modellierung komplexer Projektmanagementprobleme wie stochastische Aktivitätsdauern, stochastische Ereignisse und kontinuierliche Zeitkompression, das für die Bereitstellung eines Simulationsmodells von drei Beispielprojekten auf der Grundlage von Projektmanagement-Planspielen verwendet wird.

A.2. Performances

A.2.1. Performance on Refinery model

Table A.1 shows the performance of the optimization algorithms on the various setups of the Refinery model with the following performance indicators: loss, duration and cost. They should all be minimized. Additionally the calculation time (in seconds) the algorithms needed and the number of samples they evaluated are given.

optimizer	**loss**	**duration**	**cost**	**calculation time**	**samples**
base model					
human best	930764848	367	2536144		
human average	1790662190	579	3044707		
heuristic	811353180	330	2458646		
BayesOpt_hyperopt	845264556	333	2538332	2392	10000
BayesOpt_emukit	1081099264	376	2875264	2057	3000
BayesOpt_gPyOpt	813981250	326	2496875	4111	900
GeneticOpt	790309738	326	2424263	662	360000
actorCritic	800762800	328	2441350	1673	80000
base model - with VAE					
BayesOpt_hyperopt	952613640	363	2624280	2593	10000
BayesOpt_emukit	809758246	326	2483921	985	3000
BayesOpt_gPyOpt	877000150	338	2594675	2475	700
GeneticOpt	803089590	326	2463465	665	360000
actorCritic	800762800	328	2441350	1864	320000
with compression					
BayesOpt_hyperopt	866710000	295	2938000	3211	7000
BayesOpt_emukit	1310106050	382	3429597	3581	3000
BayesOpt_gPyOpt	888042232	308	2883254	2705	700
GeneticOpt	718297620	268	2680215	4789	360000
actorCritic	756566415	285	2654619	2726	51200
with compression and VAE					
BayesOpt_hyperopt	1042473215	365	2856091	1766	4000
BayesOpt_emukit	1409331020	430	3277514	3802	3000
BayesOpt_gPyOpt	1119161197	371	3016607	1600	700
GeneticOpt	794785876	313	2539252	2207	900000
actorCritic	1439269763	431	3339373	2204	512000

Table A.1.: Performance of optimization algorithms on refinery model (lower and greener is better)

A.2.2. Performance on TopSim roller coaster model

Tables A.2 and A.3 show the performance of the optimization algorithms on the various setups of the TopSim roller coaster model with the following performance indicators: profit, duration, cost, technology index and quality index. The duration and cost should be minimized and the technology and quality index should be maximized. Additionally the calculation time (in seconds) the algorithms needed and the number of samples they evaluated are given.

optimizer	profit	duration	cost	techn.	qual.	calc. time	samples
			base model				
default	-2425	73	8525.0	0	0		
heuristic	-10125	53	12370.0	-26	-38		
BayesOpt_hyperopt	2250	68	9370.0	21	27	2308	7000
BayesOpt_emukit	-125	72	9595.0	19	18	2955	3000
BayesOpt_gPyOpt	2140	63	9480.0	13	23	1765	700
GeneticOpt	4175	68	9145.0	28	37	2081	1800000
actorCritic	4760	68	9260.0	30	42	2253	25600
			base model - with VAE				
BayesOpt_hyperopt	3760	68	9560.0	27	38	2450	7000
BayesOpt_emukit	3710	65	9310.0	23	33	1881	3000
BayesOpt_gPyOpt	4060	69	9310.0	29	39	1872	700
GeneticOpt	4350	69	9320.0	31	40	2900	1800000
actorCritic	1565	69	9705.0	20	27	222	25600
			with compression				
BayesOpt_hyperopt	2734.69	58.52	10335.3	16	23	2541.00	4000
BayesOpt_emukit	1283.64	61.15	10636.4	16	18	5227	3000
BayesOpt_gPyOpt	3024.68	62.6	10345.3	19	32	1778	700
GeneticOpt	6288.33	59.88	9781.7	29	42	2379	900000
actorCritic	5929.52	62.95	10240.5	34	45	2995	25600
			with compression and VAE				
BayesOpt_hyperopt	2582.06	65.16	9337.9	20	25	2119	4000
BayesOpt_emukit	759.18	66.94	9660.8	18	14	4723	3000
BayesOpt_gPyOpt	347.96	70	9672.0	20	17	2667	700
GeneticOpt	4869.48	60.96	9700.5	26	32	2111	900000
actorCritic	255.88	64.86	9874.1	9	15	1217	256000

Table A.2.: Performance of optimization algorithms on deterministic TopSim roller coaster model (greener is better)

optimizer	profit	duration	cost	techn.	qual.	calc. time	samples
			probabilistic				
BayesOpt_hyperopt	1097.22	65.59	9712.8	15	19	986	4000
BayesOpt_emukit	163.29	67.03	9691.7	18	9	3034	3000
BayesOpt_gPyOpt	2812.33	64.88	9517.7	19	28	1789	700
GeneticOpt	4184.36	68.41	9325.6	29	38	1618	900000
actorCritic	3734.99	63.76	9655.0	22	33	2628	25600
			probabilistic - with VAE				
BayesOpt_hyperopt	2739.48	65.31	9790.5	18	32	1111	4000
BayesOpt_emukit	2764.36	64.34	9845.6	19	29	1362	2500
BayesOpt_gPyOpt	2735.36	64.64	9564.6	20	26	1515	700
GeneticOpt	4047.22	66.36	9402.8	25	37	2310	900000
actorCritic	463.56	61.18	10286.4	13	8	1844	512000
			probabilistic with compression				
BayesOpt_hyperopt	2725.61	61.41	10694.4	24	25	2160	4000
BayesOpt_emukit	1773.24	57.94	10692.8	13	19	3173	2500
BayesOpt_gPyOpt	2500.79	58.99	11269.2	19	28	1070	700
GeneticOpt	6220.63	59.44	10229.4	32	43	3062	900000
actorCritic	5674	61.54	10396.0	32	44	2530	25600
			probabilistic with compression and VAE				
BayesOpt_hyperopt	3969.18	69.53	9400.8	30	38	2195	4000
BayesOpt_emukit	2472.49	71.94	9577.5	26	35	2193	2500
BayesOpt_gPyOpt	3674.63	69.85	9395.4	28	38	1087	700
GeneticOpt	4689.71	68.5	9220.3	30	41	2272	900000
actorCritic	2539.19	65.23	10520.8	22	33	1120	256000

Table A.3.: Performance of optimization algorithms on probabilistic TopSim roller coaster model (greener is better)

A.2.3. Performance on MIS model

Table A.4 shows the performance of the optimization algorithms on the various setups of the MIS model with the following performance indicators: overall, cost, time, quality, user acceptance, team morale and security. They should all be maximized. Additionally the calculation time (in seconds) the algorithms needed and the number of samples they evaluated are given.

Optimizer	**Over.**	**Cost**	**Time**	**Qual.**	**User**	**Mor.**	**Sec.**	**calc.**	**samples**
				base model					
human - best	62.00	51.00	56.00	75.00	75.00	66.00	65.00		
human - average	58.93	49.90	50.70	68.80	65.60	58.30	59.80		
heuristic	59.10	53.33	50.60	66.67	65.00	62.00	57.00		
Bayes_hyperopt	62.80	53.33	51.00	69.44	73.00	67.00	63.00	2143	3000
Bayes_emukit	62.59	53.33	53.00	67.22	72.00	67.00	63.00	3543	3000
Bayes_gPyOpt	64.29	53.33	53.20	72.22	76.00	66.00	65.00	1354	700
GeneticOpt	65.05	53.33	52.20	72.78	78.00	69.00	65.00	1654	1600
actorCritic	65.04	53.33	52.60	73.33	77.00	69.00	65.00	2385	8192
				base model - with VAE					
Bayes_hyperopt	63.20	50.63	53.80	67.78	76.00	68.00	63.00	800	3000
Bayes_emukit	62.31	49.50	54.60	67.78	71.00	68.00	63.00	2044	3000
Bayes_gPyOpt	64.04	53.33	54.80	66.11	77.00	68.00	65.00	1597	700
GeneticOpt	65.04	53.33	52.60	73.33	77.00	69.00	65.00	2294	29000
actorCritic	64.18	49.97	51.80	73.33	77.00	68.00	65.00	1136	12800
				with compression					
Bayes_hyperopt	63.35	46.75	59.38	70	73	68	63	2726	3000
Bayes_emukit	62.02	46.46	59.35	68.33	75	61	62	3007	2500
Bayes_gPyOpt	64.48	42.94	63.6	73.33	77	67	63	1982	700
GeneticOpt	65.1	44.61	64.35	71.67	78	67	65	3946	45000
actorCritic	65.01	46.44	60.83	72.78	76	69	65	2382	3200
				with compression and VAE					
Bayes_hyperopt	61.58	43.92	57.2	68.33	72	64	64	1532	3000
Bayes_emukit	60.65	46.93	51.52	69.44	72	63	61	2485	2500
Bayes_gPyOpt	61.94	49.9	51.2	70.56	72	65	63	4521	700
GeneticOpt	63.69	45.88	62.34	68.89	73	67	65	4667	45000
actorCritic	64.56	46.89	59.47	70	77	69	65	2269	25600

Table A.4.: Performance of optimization algorithms on MIS model (greener and higher is better)

List of Figures

List of Tables

Bibliography

[1] P. Malik. Project management network diagrams – types, explanation, examples. [Online]. Available: http://www.pmbypm.com/wp-content/uploads/2014/06/PDM-Sample.jpg

[2] J. M. Antill and R. W. Woodhead, *Critical path methods in construction practice*. John Wiley & Sons, 1990.

[3] D. R. Fulkerson, "A network flow computation for project cost curves," *Management science*, vol. 7, no. 2, pp. 167–178, 1961.

[4] W. Meyer and L. R. Shaffer, *Extensions of the critical path method through the application of integer programming*. Department of Civil Engineering, University of Illinois, 1963, vol. 2.

[5] D. R. Robinson, "A dynamic programming solution to cost-time tradeoff for cpm," *Management Science*, vol. 22, no. 2, pp. 158–166, 1975.

[6] P. De, E. J. Dunne, J. B. Ghosh and C. E. Wells, "Complexity of the discrete time-cost tradeoff problem for project networks," *Operations research*, vol. 45, no. 2, pp. 302–306, 1997.

[7] R. H. A. El Razek, A. M. Diab, S. M. Hafez and R. F. Aziz, "Time-cost-quality trade-off software by using simplified genetic algorithm for typical repetitive construction projects," *World academy of science, engineering and technology*, vol. 37, pp. 312–320, 2010.

[8] C. Chao, "Software project management net: a new methodology on software management," 1996.

[9] L.-h. Zhang, X. Zou and Z.-x. Su, "Ga optimization model for time/cost trade-off problem in repetitive projects considering resource continuity," *Applied Mathematics & Information Sciences*, vol. 7, no. 2, p. 611, 2013.

[10] S. Mungle, "A portfolio approach to algorithm selection for discrete time-cost trade-off problem," *CoRR*, vol. abs/1412.1913, 2014. [Online]. Available: http://arxiv.org/abs/1412.1913

[11] R. de Oliveira Moraes and A. H. Batista, "Project management business game," *Journal of Industrial Engineering and Management (JIEM)*, vol. 13, no. 2, pp. 95–102, 2015.

[12] R. Atkinson, "Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria," *International journal of project management*, vol. 17, no. 6, pp. 337–342, 1999.

[13] H. EDHOLM and M. LIDSTRÖM, "Crunch time: the causes and effects of overtime in the games industry," 2016.

[14] S. E. Elmaghraby, “On the fallacy of averages in project risk management,” *European Journal of Operational Research*, vol. 165, no. 2, pp. 307–313, 2005.

[15] D. G. Malcolm, J. H. Roseboom, C. E. Clark and W. Fazar, “Application of a technique for research and development program evaluation,” *Operations research*, vol. 7, no. 5, pp. 646–669, 1959.

[16] M. Rees, *Business risk and simulation modelling in practice: using Excel, VBA and@ RISK*. John Wiley & Sons, 2015.

[17] Z. Song, H. Schunnesson, M. Rinne and J. Sturgul, “An approach to realizing process control for underground mining operations of mobile machines,” *PloS one*, vol. 10, no. 6, p. e0129572, 2015.

[18] A. Azaron, C. Perkgoz and M. Sakawa, “A genetic algorithm approach for the time-cost trade-off in pert networks,” *Applied mathematics and computation*, vol. 168, no. 2, pp. 1317–1339, 2005.

[19] K. M. Haque, M. Hasin, A. Akhtar *et al.*, “Genetic algorithm for project time-cost optimization in fuzzy environment,” *Journal of Product Development and Management (PDM)*, vol. 5, no. 2, pp. 364–381, 2012.

[20] B. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer and B. Steece, “Cost estimation with cocomo ii,” *ed: Upper Saddle River, NJ: Prentice-Hall*, 2000.

[21] T. I. S. GmbH. Topsim - project management teilnehmerhandbuch. Support of project management business game discontinued by TopSim. [Online]. Available: http://www.roesch-pr.de/Teilnehmerhandbuch%20Project%20Management.pdf

[22] A. Martin, “The design and evolution of a simulation/game for teaching information systems development,” *Simulation & Gaming*, vol. 31, no. 4, pp. 445–463, 2000.

[23] A. Martin. Mis project manager online game. [Online]. Available: http://misprojectmanager.andrewmartinphd.net/ManageEvent.aspx

[24] A. E. Eiben, J. E. Smith *et al.*, *Introduction to evolutionary computing*. Springer, 2003, vol. 53.

[25] E. Brochu, V. M. Cora and N. De Freitas, “A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning,” *arXiv preprint arXiv:1012.2599*, 2010.

[26] A. Paleyes. (2019) Time complexity of gpyopt. [Online]. Available: https://github.com/SheffieldML/GPyOpt/issues/262

[27] A. G. Barto, R. S. Sutton and C. W. Anderson, “Neuronlike adaptive elements that can solve difficult learning control problems,” *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[28] H. Michalewski. reinforcement learning. [Online]. Available: https://cdn-sv1.deepsense.ai/wp-content/uploads/2016/09/playing-atari-on-ram-with-deep-q-learning-Agent-Env-crop.png

[29] M. Minsky, “Steps toward artificial intelligence,” *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.

[30] A. Hamid, "Dopamine contributions to motivational vigor and reinforcement driven learning." 2016.

[31] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[32] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[33] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "Openai gym," 2016.

[34] T. E. authors, "Emukit: Emulation and uncertainty quantification for decision making," https://github.com/amzn/emukit, 2018.

[35] T. G. authors, "Gpyopt: A bayesian optimization framework in python," http://github.com/SheffieldML/GPyOpt, 2016.

[36] J. Bergstra, D. Yamins and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," 2013.

[37] I. CSIMarket. (2019) Construction services industry profitability. [Online]. Available: https://csimarket.com/Industry/industry_Profitability_Ratios.php?ind=205

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared.

This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

Stuttgart, 23.10.2019

Malte Ebner