Chapter 1

Introduction

Most software system nowadays are configurable. This configurability allows end-users, developers and administrators to adapt these software systems to their specific needs. But this adaptibilty comes at a high cost.
The more options there are to configure a system the harder it gets to find configurations optimal for the specific needs of the user. While the functional aspects of a configuration option are usually well known and documented, the non functional properties are mostly unknown. This makes it hard to configure a software system optimal for a non functional property like performance, power-consumption or memory footprint.
The naive way to test all possible configurations to determine the optimal configuration for a given non functional property is unfeasable. This is due to the combinatorial complexity of the configuration space and the constrains on the configuration options. Since there is no straightforward way to determine the influence of configuration options on non functional properties, these properties are mostly ignored in the choice of a fitting configuration [**?**] or are only chosen by previous "tribal knowledge", possibly leaving already build in optimizations in software systems untapped.

In literature, a often used method to help in choosing the optimal configuration for a non functional property is to create a model describing the influences of configuration options on the non functional property [**?**]. This model can then be used to find optimal configurations and predict the non functional property without any further testing of the system. A developer for example, then might use this model to find a configuration which minizes the power consumption of the software to reduce operational cost. To create these kind of models, machine learning techniques are a simple go-to. While the creation of such models is a research field of its own, there are many well understood machine learning techniques, which can be used to create a model for the influence of software configurations on non functional properties. Since

How?

for more related work on performance modeling, see related work, .e.g.

related work

the performance of the resulting model of almost all machine learning techniques is dependent on the data used to create the model, this is a major aspect, which has to be looked at to create well performing models. Because it is infeasable to test all possible configurations, a subset of configurations must be selected. This process is called sampling. While sampling in traditional machine learning applications might be as easy as just randomly picking data points to train the model, it is not as straightforward fot configurations options in the presence of constarins. This is mainly due to the constrains the configuration options have, that makes the sampling of configurations difficult.

[?]

There are several techniques already tackling the problem of sampling on software configurations. In **??** an overview of a few sampling techniques is presented. These techniques allow a user to sample software configurations and select pseudo random configurations under specific critera. While some results are quite good with accuracies under 1% [**?**], the motivation to further reduce the needed amount of samples is still there, to make predictions more performant and easy to use. This reduction of required data to model the influence of multiple parameters on a specific variable is not a problem specific to software configurations. In [**?**] and [**?**] an experimental design for sensitivity analysis is described which reduces the amount of needed simulations to less than the amount of parameters and which is still able to give meaningfull information of the influence of those parameters. This experimental design is called group sampling. While the core concept of analyzing the influence of multiple parameters on a single variable is quite similar to the problem of analyzing the influence of configuration options on non functional properties, there are a few problems in adapting these techniques to software configurations. This is, again, mainly due to the constraints and interactions of configuration options.

289-Ende ... error?

Clearify, prediction error/needed data

The constraints on configuration options makes it hard to look at the influence of a single configuration option. This is because one option might not be able to be turned on indipendently of other configuration options. Therefore, measurements conducted on this pair of configuration options might not be able to tell the specific influence of the single configuration options in question. Further, certain configuration options might interact with eacht other, cancaling out or multipling the influence of each other. This is easly unterstood by looking at two common configuration options in compression programms. The encryption and the compression used. If the compression is more aggressivly compressing the data, the encryption part of the programm has less data to encrypt, making the programm more perfomant. A more detailed explaination can be found in **??** and **??**, respectively.

In this work, we analyse how the experimental design of group sampling can be adapted to software configurations, by implementing a group sampling algorithm for software configurations and answering follwing questions:

- **How can we mitigate the effect of constraints on the configuration options for a group sampling approach?** In literature, in group sampling, constrains on the parameters are non-existent, but with configuration options these constraints make it impossible to create arbitrary groupings. We try to mitigate the resulting partitioning by allowing configuration options to be present in multiple groupings and optimize those groupings to minimize the Hamming distance between them.

- **How effective is a group sampling algorithm compared to random sampling?** We answer this questions with experiments on real world and synthetic data sets. By comparing group sampling and random sampling with linear regression with varying sample and group sizes, we determine under which conditions, if any, group sampling is superior to random sampling.

& independen feature stuff

- **How scalable is group sampling?** Since group sampling is an experimental design for large amounts of parameters, the resulting model should be useful for large software systems where random sampling might not be of use because of large sample sets required to create a performant model. This is done by creating synthetic data for a software system with a large configuration space.

- **Can group sampling be used to effectively identify feature interactions and include them in the resulting model?** Since the grouping of configuration options already includes the influences of feature interactions, group sampling can be used to identify these interactions and effectively include those in the resulting model. We answer this question with experiments on synthetic data, where the amount of feature interactions is at our discussion.

Further we will have a look at optimization techniques, which are already described in literature to get better results when using a group sampling algorithm.

hier sollte dann wahrscheinlich noch eine grobe beschreibung des ergebnisses rein ...