Automated Algorithm Configuration for ILP-based Process Discovery

Eindhoven University of Technology

TU/e

J.R. Slabbekoorn

Supervisors:

B.F. van Dongen
S.J. van Zelst
Magnum vectigal est parsimonia
1 Acknowledgements

I’m grateful for all the efforts made by all the people helping me during this project. I will not mention all of them, only the ones directly involved to this project. I would like to thank my supervisors S.J. van Zelst and B.F. van Dongen. They are among the brightest persons I have met.

At the start of the project I already indicated to my supervisors that I really liked the subject. In my point of view it all comes down to being as frugal as possible with our computational resources. I tried to find the boundaries of what is possible, in many ways. I hope after reading this report you will find out if we succeeded.
2 Abstract

Automated algorithm configuration is used for optimising an algorithm using another algorithm. In this research automated algorithm configuration is used for optimising the performance of ILP-based process discovery. The underlying ILP problems of ILP-based process discovery are currently solved with the default configuration of an ILP-solver. Automated algorithm configuration will search for configurations which optimise the performance of solving these underlying ILP problems. In this research the automated algorithm configuration method ParamILS is used with different configuration scenarios. After applying ParamILS to the set of underlying ILP problems a decision tree is used to predict the optimal configuration for each individual ILP problem. The prediction is made using the information known before the solving of an ILP problems. In order to find the optimal configuration the prediction is compared to the outcome of the automated algorithm configuration.
3 Introduction

Process mining [1] is a research discipline that aims at the understanding and improvement of business processes. Process mining can be divided into three branches: process discovery, conformance checking and process enhancement. Process discovery constructs process models based on event data originating from (business) process execution. Conformance checking compares existing process models against the event data of the same process. It is used to check if the actual process recorded in the event data conforms to the process models and in reverse the process models conforms to the event data. Process enhancement aims at improving process models including repairing, extending and adjusting the models.

The scope of the research is focused on the branch of process discovery. Several process discovery techniques exist, differing in both their internal working and resulting process modelling formalism. A technique guaranteeing formal properties with regard to the found process models is region-based process discovery. Region-based process discovery originates from Petri net synthesis [2]. Petri net synthesis aims at deciding whether there exists a labelled graph that exactly describes a systems behaviour. In region-based process discovery a Petri net is considered as a system which can be constructed according to the results of this synthesis. When using region-based approaches two classes exist: state-based and language-based. In state-based region theory a transition system describes the systems behaviour. In language-based region theory a language describes the systems behaviour.

An example of a process discovery algorithm using language-based region theory is the Integer Linear Programming (ILP)-based process discovery algorithm [3]. The algorithm translates event data into a language which is transformed into a set of underlying ILP problems. The results of solving the ILP problems are used to construct a Petri net. An advantage of applying the ILP-based process discovery algorithm is the inherited formal properties from region theory. A disadvantage is that the algorithm uses multiple ILP problems to obtain these results, which are computational expensive. Hence, performing process discovery on real life event logs results in poor performance because the real life event data is translated by the ILP-based process discovery algorithm into a large number of ILP problems.

The computational cost of solving an ILP problem depends on the elements: the ILP problems complexity, the solution method and the associated solution method parameters. The complexity of the ILP problem depends on characteristics of the process discovery algorithm, i.e. how the algorithm translates the event data into an ILP problem. For example the process discovery algorithm determines the constraints of the ILP problem. These constraints are influencing the computational costs. The next element is the solution method chosen to solve the ILP problem. The computational costs of the solution method are determined by the number of steps and the needed computations per step. Each solution method has a number of parameters. The parameters influence the behaviour of the solution method. Therefore the parameters influences the number of steps or the computations per step.

In general different solution methods combined with their parameters are suitable for solving ILP problems. When solving an ILP problem the chosen solution method and parameters can have different computational costs depending on the ILP problems. Thus the performance for a selected solution method and parameters can be optimal in terms of computational costs for a certain ILP problem and sub-optimal for another ILP problem.
Solving the ILP problems software is used which has implemented the solution methods and parameters. Software that is capable for solving ILP problems is defined in this research as the ILP-solver. ILP-solvers are widely used for the optimisation of computational costs. Optimisation of computational costs is referred in this research as performance optimisation. ILP solvers are used for a variety of Linear Programming (LP) problems performance optimisation because the solution methods and parameters can easily be adapted. However, for ILP-based process discovery no optimisation is still performed. By adjusting the solution method and parameters it is still possible that the performance will improve.

Optimisation of the computational costs can be performed by an optimisation algorithm. An algorithm which is optimising another algorithm is defined as automated algorithm configuration. Automated algorithm configuration for optimising the performance of solving of ILP-based process discovery is not a trivial task. The task is complex because the relationship between the performance and the ILP solvers solution methods and parameters is unknown. To handle this advanced algorithm configuration methods are therefore needed. The development of these methods and the abundance of computational resources enables the use advanced algorithms. In this research the combination of these factors is therefore used for the optimisation of the computational costs for solving the underlying ILP problems of ILP-based process discovery.

3.1 Research Goal

The goal of this research is to minimise the computational costs of ILP-bases process discovery by applying automated algorithm configuration on the ILP-solver used for solving the underlying ILP problems of ILP-based process discovery. The computational costs in this research are defined as the time needed to solve an ILP problem. In this research the time needed to solve an ILP problem is also referred as the performance of an ILP problem. In order to minimise the computational costs the scope of this research doesn’t include the hardware part influencing the performance, despite the effects of the hardware on the performance can be significant.

In this research an ILP-solver is used which contains multiple solution methods and parameters. The possible combinations of the solution methods and parameters are very large. For the ILP-solver that has been used there are in total a number of about $9 \times 10^{21}$ combinations. Finding a solution method and parameters which minimises the computational costs is not therefore trivial. Besides the large number of combinations the relations between the solution methods and parameters and the performance is complex. The effect of each of these solution methods and parameters on the performance is unknown. To optimise the computational costs for ILP problems used in process discovery by manual search is therefore not possible. In order to optimise the computational costs the use of an automated approach is opted in this research. For the automated approach used in this research the ILP-solver is considered as an algorithm which can be optimised. The optimisation of this algorithm is performed by another algorithm; this is referred as automated algorithm configuration. In this research the configurations are the different combinations of solution methods and parameters. In order to find out if there is an automated algorithm configuration method which is applicable for using on ILP-based process discovery a literature study is performed to assess the different automated algorithm configuration methods. The assessment is performed the compare the methods according requirements for automated algorithm configuration on ILP-based process discovery. The automated algorithm configuration methods are used to search for improvements in the performance by adjusting the solution methods and parameters. Further the information known prior to solving an ILP problem are used in order to optimise the performance.
of solving the ILP problems.

- What is the best automated algorithm configuration method suitable for solving ILP problems?
- Is the default configuration the most optimal configuration for ILP based process discovery?
- Is it possible to predict for an ILP problem instance of ILP based process discovery which configuration is optimal?

3.2 Thesis Outline

In the previous introduction section we introduced the concepts of process discovery and automated algorithm configuration. The research questions are stated. In this outline an overview of remaining parts of this thesis are provided.

In Chapter 4 the first research question is answered by providing a literature review which assesses the different criteria defined for automated algorithm configuration for ILP-based process discovery. The different automated algorithm configuration methods are evaluated and a single automated algorithm configuration method is used in the next sections.

In Chapter 5 the selected automated algorithm configuration method in Chapter 4 is used to optimise the performance for ILP-based process discovery. An ILP-solver is used to solve the underlying ILP problems for the ILP-based process discovery. Different automated algorithm configurations scenarios are used to optimise the configurations of the ILP-solver.

In Chapter 6 the results of the automated algorithm configuration are used in combination with the information about the ILP problems which is known before the solving of the ILP problems. The information is used for minimising the performance of ILP-based process discovery.
4 Automated Algorithm Configuration Methods

The research discipline of automated algorithm configuration aims at the automated configuration of an algorithms behaviour in order to optimise the algorithms objectives. The automated configuration is performed by an algorithm which is referred to as the configuration algorithm. The algorithm that has to be optimised by the configuration algorithm is referred to as the target algorithm. An algorithm is a sequence of operations. Within the algorithm an operation or a set of operations can be represented by a parameter. A set of parameters is defined as the configuration of an algorithm. In the automated algorithm configuration these configurations can be changed by the configuration algorithm. Changing the configurations therefore changes the sequence of operations determining the behaviour of the algorithm.

An example of changing an algorithms parameters and therefore changing its behaviour is given by a simple algorithm. The target algorithm performs a simple mathematical calculation. First the algorithm takes two numerical values as input. Next the target algorithm performs an operation on these two numerical values. A parameter represents the operation performed on two numbers. When the parameter is set on the two numbers will be added and when the parameter is set off the two numbers will be subtracted. When the output of the target algorithm has to be maximised the confirmation algorithm has to set the parameter to on to find the optimal solution for the target algorithm.

The configuration algorithm will change the configurations of the target algorithm depending on the automated algorithm configuration method. Since there are many types of target algorithms and parameters performing automated algorithm configuration is not a trivial task. To handle these differences the field of automated algorithm configuration research makes a distinction between different types of automated algorithm configuration. Broadly the automated algorithm configuration methods are specified into two main categories. These categories are:

- Optimising an algorithm’s configurations given an objective function.
- Automating an algorithm’s construction.

Optimising the algorithm’s configuration is about the optimisation the set of parameters and parameter values, i.e. the configuration of an algorithm. A set of parameters and parameter values is used for the optimisation the target algorithm. The target algorithm itself will not change during the automated algorithm configuration. Which implies that the set of parameters also does not change during the automated algorithm configuration. On the other hand the automated construction of algorithms can change the algorithm during the automated algorithm configuration. The parameters included in the automated algorithm configuration are enabled to add, remove, synthesise or extract parts of the algorithm. Meaning that the parameters and parameter values of the algorithm can change according the changes in the algorithm.

The scope of this research is limited to the approach of optimising an algorithms configurations given an objective function. For automated algorithm configuration a number of methods have been developed in different research fields. Not all automated algorithm configuration methods are equally suitable for using them on automated algorithm configuration for ILP-based process discovery. A literature study is used to make an assessment between these automated algorithm configuration methods. The assessment is based on a number of selection criteria which take into account the characteristics of the underlying
ILP problems of process discovery. The selection criteria which are used for the assessment are: parameter types, number of parameters handled per algorithm configuration run, convergence properties and improvement quality of the algorithm configuration.

1. **Parameter types**: Describes the type of the parameters that the automated algorithm configuration methods can handle, e.g. numerical, ordinal or categorical parameters. For ILP based process discovery the parameter types are given by the ILP-solver. The ILP-solvers solution methods and associated parameter values are represented by different type of parameters. The automated algorithm configuration method therefore has to be able to handle a set of multiple types of parameters.

2. **Number of parameter handled**: The amount of parameters that the automated algorithm configuration method can handle in a run. The number of possible parameters included in a configuration change according the evaluations of the configuration methods. It is not possible due to the large number of possible configurations to evaluate all the possible configurations, which is equivalent to evaluating all the parameters. The automated algorithm configuration methods therefore have to limit themselves to a number of parameters.

3. **Convergence properties**: The convergence properties are the ability of the automated algorithm configuration method to provide a robust result. Convergence is related to the reliability of the configuration method. When applying the automated algorithm configuration method multiple runs using the same settings, the outcome should provide the same parameters. Convergence rate is about how fast the convergence is reached.

4. **Improvement quality**: The manner in which the outcome of the objective function of a configuration differs from the outcome of other configurations. The improvement quality is related to the ability of the algorithm configuration method to evaluate different configurations.

The different selection criteria used for the assessment of the automated algorithm configuration methods are interdependent. Interdependency between the selection criteria parameter types handled and convergence properties is determined by the number of possible configurations. Each parameter type will lead to an amount of total possible configurations. A large number of configurations will decrease the possibility of finding an improved configuration; decreasing the convergence properties as convergence rate. Interdependency between the selection criteria convergence properties and the improvement quality is determined by the fact that when convergence occurs the improvement quality is related to the converged outcome.

### 4.1 Method Overview

In the method overview the literature study will describe a number of automated algorithm configuration methods. The selection criteria described in the previous section are used to asses the different methods.

#### 4.1.1 Hill Climbing Algorithm Configuration

Hill climbing algorithm configuration is a local search optimisation technique iterating through the search space until the objective function of the algorithm stops increasing. Hill climbing uses only numerical parameters and requires that these parameters are discretised. The hill climbing algorithm
configuration only searches in the neighbourhood of the parameters. The neighbourhood of the numerical parameters are discretised values next to the current value in that parameters domain. The hill climbing will only check the numerical parameter values in non-decreasing order. This type of search enables the hill climbing to handle a large number of parameters.

Results on hill climbing as performed in the research is performed by [4,5] shows that the hill climbing algorithms are working well under certain objective functions, convergence however is not guaranteed. In cases where there exist many local optima a perturbation step has to be included in the hill climbing algorithm [6–8]. Whenever the outcome of the algorithms objective function stops increasing the perturbation step takes a random step in the search space, i.e. all the possible configurations. If local optima do not have a large influence hill climbing is able to find improved configurations. Strong characteristics of the hill climbing is that the number of parameters is can handle is unbounded. The limitation of hill climbing is that it only uses numerical parameters.

4.1.2 Genetic Algorithm Configuration

Genetic algorithm configuration starts with a population containing a set of parameters. Using the population the objective function of the genetic algorithm is calculated. The outcome of the objective function is compared to an optimisation criterion [9,10]. The criterion indicates which parameters in the population have to be changed by comparing the outcome against the previous populations. The genetic algorithm continues with a new population of parameters according a small randomised movement on the population [11]. Changing of the populations in this manner has the property that it has a good improvement quality. The strength of genetic algorithms is that many different types of parameters can be used because the representation of the parameters in the genetic algorithm. A limitation of the genetic algorithm is the slow convergence because of the optimisation criterion which takes many evaluations of all the different populations of parameters.

4.1.3 Racing Algorithm Configuration

Racing algorithm configurations methods solve all the problem instances with a single configuration per iteration. Accordingly it eliminates the problem instances that are not solved within the time of a time-out value [12,13]. The problem instances which are solved are reused for a next iteration which is solving the problem instances with a different configuration. The racing algorithm stops whenever no problem instance can be eliminated or whenever there are no more configurations and therefore the most optimal configuration is known. Starting the racing algorithm with all the problem instances results in an increase in time of the initial iteration. The time increases when the number of problem instances increases. However as more problem instances are eliminated the racing algorithm will speed up and converges faster towards an optimal configuration. The strength of the racing algorithm configuration is therefore that it will converge after a small number of iterations because of the eliminations of the problem instances. Especially for a small number of total configurations this will result in a fast the convergence. A limitation of the racing algorithm is that it is not capable of handling heterogeneous sets of problem instances [14]. When a problem instance is eliminated it cannot be reused for another configuration. While another configuration used on this problem instance could be optimal for that problem instance.
4.1.4 Model-based Optimisation

Model-based optimisation algorithm configuration combines statistical methods and algorithm configuration in order to find optima in the algorithms objective function \[15\]. In model-based optimisation the algorithm configuration is considered as black box optimisation problem \[16,17\]. Exploring the configurations the output of the objective function are mapped onto the domain of an undefined function. The shape of this function is unknown therefore the configuration is changed into small adjustments in order to detect its shape. The form of this function indicates which parameters should be changed for detecting the rest of its shape. This is referred to as exploring a surface response model. The strength of model-based optimisation for algorithm configuration is that they are able to handle non-parametric shapes of the surface respond model \[18\]. With different surface response models the model-based optimisation has proven to find that it is able to find optimal configurations, meaning the model-based optimisation has strong convergence. However the research on model-based optimisation is performed for only numerical parameters. A limitation of model-based optimisation is that an accurate surface response model needs for a large number of adjustments in the configurations to become accurate.

4.1.5 Iterative Local Search

Iterative local search algorithm configuration is an optimisation technique searching the neighbourhood of a configuration for improvements \[19\]. Iterative local search will evaluate only evaluate the parameters in the search space that are in the neighbourhood of the current configuration. The search space is the set of possible configurations. The neighbourhood is defined as the parameters which are next to each other in the parameters domain. The domain is differing for each type of parameter. Searching the domain of discretised numerical and ordinal parameters is trivial because, however for categorical parameters their neighbour is defined as a random value in their domain. The search stops whenever no improvement can be found in the neighbourhood of a configuration. After all the possible configurations have been evaluated a perturbation is made to overcome local optima \[20\]. The strength of iterative local search is the local search property for multiple types of parameters which provides strong convergence properties. The structured approach of handling simultaneous changes in the parameters neighbourhood is also a strength of iterative local search. A limitation of iterative local search is that the parameters have to be discretised. Whenever there are many discretised parameters there is accordingly a large search space.

4.2 Method Selection

Based on the literature study of the automated algorithm configuration an assessment is made. The assessment contains the selection criteria as previously described to determine which automated algorithm configuration method has to be used for ILP-based process discovery automated algorithm configuration.

The score of each of the algorithm configuration methods is shown in Table 1. The table indicates three different scores. A green checkmark, yellow square and a red sign. For the score of the selection criteria the checkmark is a positive score, the yellow square is undecided and the red sign is a negative score. The scores show that half of the automated algorithm configuration methods can handle different types of parameters. The selection criteria number of parameters the methods can handle per run however is restricted for some methods as shown by the negative scores. The convergence properties for more than half of the algorithm configuration methods are good. For almost all the automated algorithm configuration methods the improvement quality has a positive score.
4 AUTOMATED ALGORITHM CONFIGURATION METHODS

Table 1: Comparison of different automated algorithm configuration methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
<th>Number</th>
<th>Convergence</th>
<th>Improvement Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hill Climbing</td>
<td></td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Genetic</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Racing</td>
<td>✔</td>
<td>x</td>
<td>✔</td>
<td>□</td>
</tr>
<tr>
<td>Model-based optimisation</td>
<td></td>
<td>□</td>
<td>✔</td>
<td>□</td>
</tr>
<tr>
<td>Iterative Local Search</td>
<td>✔</td>
<td>□</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Comparing the scores in Table 1, the most positive score overall is obtained by iterative local search algorithm configuration. The method is the best suited for ILP-based process discovery because it is able to handle the multiple parameter types, has good convergence properties and has a good improvement quality. The method chosen is the parameter iterative local search (ParamILS) framework [19]. The ParamILS framework is used in the further part of this research. The ParamILS framework is developed for iterative local search method with option of using multiple settings for the automated algorithm configuration.

4.3 The ParamILS Framework

The Parameter Iterative Local Search framework is selected from the assessment made on the literature study. The ParamILS framework consists of a number of algorithms, which provide advanced functionality for automated algorithm configuration. Not all the functionality of the framework is needed to perform automated algorithm configuration for ILP-based process discovery. In this research focuses only on the basic functionality belonging to the ParamILS framework. The functionality that is used is described in this section.

Figure 1: A flow chart of the ParamILS framework.

The ParamILS framework starts with an initialisation. In the initialisation the inputs of the meta-parameters and parameter values are provided to the configuration algorithm. The meta-parameters are the parameters of the configuration algorithm. An example of a meta-parameter is the total time the framework in total may use to find new configurations. Accordingly the initialisation of the target algorithms parameters is provided to the configuration algorithm. These are the parameters which are going to be configured. For each of these target algorithm’s parameters a parameter domain and a starting value have to be provided.

The ParamILS framework needs to know which problem instances the target algorithm will be optimising. In this research the problem instances are given by a set underlying ILP problems of ILP-based process discovery. The set of ILP problems is put in random order into a sequence; this sequence is
defined as the seed of the ILP problems. The seed is determined before at the initialisation; therefore performing multiple runs of the configuration algorithm will not have an influence on the seed.

Accordingly all the parameters, parameter values, domains and seed are provided to the configuration scenario. In the configuration scenario automated algorithm configuration is performed by the configuration algorithm on the target algorithm. The configuration scenario will be explained in more detail in the next section. The target algorithm in this research is given by the ILP-solver LpSolve v5.5 which solves the ILP problems. The output of the solved ILP problems is provided back to the configuration algorithm. The configuration scenario determines how the output is used, which is described in the next section.

4.3.1 Configuration Scenario of ParamILS framework

The configuration scenario starts with the configuration algorithm. The configuration algorithm takes the inputs mentioned in the previous section. Accordingly the configuration algorithm determines what to do with these inputs. Using the basic functionality of ParamILS the configuration algorithm will select a subset of problem instances following the order of the seed. The configuration algorithm will then determine how the configuration is configured, i.e. what parameters are included in the configuration. The problem instances and the configuration are provided to the target algorithm which solves each the problem instances. In this research this is given by an ILP-solver which solves a single ILP problem at the time according a given configuration. The output of the target algorithm is used by the configuration algorithm to make adjustments to the configuration. After an improvement in the outcome is found the ParamILS framework will start at the beginning of the seed using the adjusted configuration.

Figure 2: A flow chart of the ParamILS framework.
4.3.2 Adjustments to ParamILS

The basic version of the ParamILS has some deficiency regarding the automated algorithm configuration of ILP-based process discovery. It is not able to overcome these deficiencies by using the more advanced functionality of ParamILS. In order to overcome this the ParamILS framework is adjusted so it is better suited for automated algorithm configuration for ILP-based process discovery.

The first adjustment is made to the configuration algorithm is regarding to the objective function. When the target algorithm solves an ILP problem it is possible that the ILP problem cannot be solved, i.e. it is infeasible. The configuration algorithm omits the infeasible solved ILP problems in the current situation. The disadvantage of this approach is that this affects the objective function. For example whenever the outcome of the objective function is an average over a number of ILP problems the average is influenced by omitting an outcome. Especially when the number of ILP problems used for calculating the average is small. The configuration algorithm is adjusted so that the objective function will use all of the solved ILP problems. Previously omitted ILP problems are now included in the objective function by assigning a maximum performance, i.e. time-out to the infeasible ILP problems.

The second adjustment made to the configuration algorithm is regarding to the iteration of the configuration algorithm. For each configuration the ParamILS framework will start at the beginning of the seed in the current situation. This approach is less suitable for heterogeneous ILP problems because it reuses the ILP problems in the beginning of the seed while it has the possibility of not using the ILP problems at the end of the seed. The configuration algorithm is adjusted so that it iterates over the seed whenever a new configuration is evaluated. After a new configuration is found the endpoint in the seed is the starting point of the new configuration. Using all the problem instances of the seed.
5 Applying Iterative Local Search on ILP-based Process Discovery

In the previous chapter different automated algorithm configuration methods are evaluated. As a result of this evaluation the ParamILS framework is selected for further usage. The ParamILS framework is used to optimise the performance of solving a set of underlying ILP problems of ILP-based process discovery. The set of ILP problems is generated by the Integer Linear Programming (ILP)-based process discovery algorithm [3]. In total 540 ILP problems are generated using this algorithm. The total set of ILP problems can be split into a training set and a test set using the 80/20 model. Meaning that the training set consists of 80% of the data and the test set consists of 20% of the data. This model is assuming that 80% of the data is representative for the entire population, which can be tested on the remaining 20% of the data. The ILP problems of the training- and test set are selected random out of the total set. Thus using a total set of 540 ILP problems the training- and test set contain randomly selected 432 and 108 ILP problems.

5.1 Baseline Measurement

An indication of the performance of the ILP problems in the training set is obtained by a baseline measurement. In the baseline measurement each ILP problem is solved with the default configuration, i.e. the default solution method and parameter values of the ILP-solver. The default configuration is currently also used by when performing ILP-based process discovery via the open-source process mining tool ProM [1].

The performance of a solved ILP problem is divided into the elements: load time, presolve time and solve time. The load time is the time needed for the ILP-solver to load the ILP problem into the ILP-solver. The load time is measured from the moment the solver starts loading the ILP problem until it the ILP-solver starts any presolving or solving activities. When mentioning presolve or solving activities we mean the usage of the algorithms of the ILP-solver. The presolve time is more complicated. It is the time needed to perform any activities on the ILP problem to make adjustments influencing the solving of the ILP problem. An example of a presolve activity is changing the ordering of the columns of the ILP in order that it will improve the solving activities. Presolve activities start after the loading of the ILP problem and stop whenever the solving of the ILP problem starts. However, the presolve activities are not required for solving of the ILP problem. Therefore, the ILP problem can be solved directly after it is loaded into the ILP-solver. The solve time describes the time needed to perform the solving activities of the ILP problem. The solve time also includes the time needed to provide the results of the solving activities of the ILP problem.
The performance of each solved ILP problem is stochastic, i.e. the performance of a ILP problem varies over multiple runs. To obtain a reliable baseline measurement the set of ILP problems is solved multiple times. In total the baseline measurement is solved for 10 runs with a time-out value of 180 seconds. The time-out value is chosen arbitrarily. The average and standard deviation of these runs are shown per performance element in Table 2. The standard deviation is relatively large compared to the average, this is an indication that there are large deviations in the performance of the ILP problems in the baseline measurement.

In order get a better indication of the deviations a histogram is made of the the baseline measurement. The histogram shows how the ILP problems are distributed over different intervals. The intervals or bins are taken with a distance of 10 seconds. For example the first bin indicates that there are about 120 ILP problems in the baseline measurement performance between 0 - 10 seconds. The histogram shows that there are multiple ILP problems which have an equivalent performance, because they form larger bins. For example the histogram shows that there is a large number of ILP problems who’s performance is under 10 seconds. Another large group is the performance of ILP problems between 10 - 20 - 30 seconds. Last it is shown in the histogram that there is a significant group which is at the bin of the timeout value between 170 - 180 seconds. The histogram of the solved ILP problems is therefore skewed towards ILP problems which do need up to 50 seconds for solving an ILP problem.

![Histogram of Baseline Measurement](image.png)

Figure 3: A histogram of the ILP problems in the baseline measurement.

<table>
<thead>
<tr>
<th>Element</th>
<th>Average performance [s]</th>
<th>St. dev. performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load time</td>
<td>0.59</td>
<td>0.18</td>
</tr>
<tr>
<td>Presolve time</td>
<td>0.29</td>
<td>0.11</td>
</tr>
<tr>
<td>Solve time</td>
<td>55.31</td>
<td>58.64</td>
</tr>
<tr>
<td>Total time</td>
<td>56.19</td>
<td>58.64</td>
</tr>
</tbody>
</table>
5.2 Settings of the Configuration Scenarios

The ParamILS framework is executed for applying automated algorithm configuration using the training set using different configuration scenarios. The settings of the configuration scenario determine the behaviour of the ParamILS framework as described in Chapter 4. Many settings are initialised before the start of each run. During a run the settings of the ParamILS framework are not changed.

5.2.1 Meta-Parameters

The meta-parameters which are determining the configuration scenarios are listed below.

- Type of objective function of the ParamILS framework
- Number of ILP problems used for objective function of the ParamILS framework
- Time-budget for the ParamILS framework
- Time-out for the ILP problems

The first two meta-parameters are related to the objective function. The type of objective function for the configuration scenario is set at the value: *average speedup*. The speedup is defined as the performance of a solved ILP problem compared to the baseline performance of the same ILP problem. The *average speedup* is the average performance of the speedup of set of ILP problems. For all of configuration scenario the type of objective function is the value: *average speedup*. Because the value *average speedup* is used another meta-parameter has to be used. This meta-parameter is the number of ILP problems used for calculating the objective function. The number of ILP problems is a meta-parameter. An example is provided how these meta-parameter have to be interpreted. The type of objective function is *average speedup* and the number of ILP problems used for the objective function meta-parameter $O_{obj}$ is set at 20. The configuration algorithm uses the seed to determine a starting point. Accordingly the target algorithm the next 20 ILP problems from the seed. The average performance is calculates over the outcome of these 20 ILP problems. The configuration algorithm will change the starting point of the seed; the seed is starting now after the 20th ILP problem in the seed for the next iteration. In this research there are four different settings used for the meta parameters regarding the objective function. The configuration scenarios uses the values $O_{obj} = 10, 50, 100$ and 200 ILP problems.

The next meta-parameter are the time-related meta-parameters. In each configuration scenario there are two different time related settings that have to be determined. The *time-budget* and the *time-out* values. The *time-budget* is the total time available for a configuration scenario. The *time-out* value is the time available for a single ILP problem. The ILP problems use a *time-out* value of 180 seconds and all the runs use a *time-budget* of 5184000 seconds.

5.2.2 Results of Configuration Scenarios

Each configuration scenario is used by the settings settings as described in previous section. The outcome of each scenario is a configuration which provides the most optimal configuration for the objective function used. The outcomes of performing automated algorithm configuration using the configuration scenarios is that each there are four different optimal configurations. The results of the automated algorithm configuration are shown in Figure 4. Each of the lines in Figure 4 represents a different configuration scenario. The last point for each line in Figure 4 i.e. the point on most the right is the
performance of the optimal configuration of configuration scenario.

Each point on the line in Figure 4 represents an improvement over the objective function. The points on the line are plotted against wall-clock time opposed to solving time. In Figure 4 it is shown that during the automated algorithm configuration there are multiple moments in time where improvements are found. These moments in time represent an improved average speedup over the previous best average speedup, given as percentual change over the baseline.

Figure 4: An overview of the different runs of the automated algorithm configurations using the averages: 10, 50, 100 and 200 ILP problems measured in speedup over the baseline.

Comparing the progress of the different configuration scenarios, i.e. the different lines in Figure 4 shows that whenever the value of $O_{obj}$ decreases there are more points on the line. For example, the configuration scenario of $O_{obj} = 10$, has the most points on the line, i.e. the most increases in the objective function during the run of the configuration scenario. While the configuration scenario of $O_{obj} = 200$ has the least increases in the objective function. The explanation for these differences is straightforward. A smaller number of $O_{obj}$ means that the scenario changes the configurations more often. This results in a larger number of evaluated configurations which potentially lead to more improved configurations.
Although in Figure 4 it seems that as the configuration is changed more often this will increase the improvement of the objective function, overall this might not be the situation. The best results in Figure 4 are obtained by the smaller $O_{\text{obj}}$ because the smallest $O_{\text{obj}} = 10$ obtains the largest improvement of the objective function. However, the results in Figure 4 do not indicate if this improvement is a local- or a global improvement. Whenever $O_{\text{obj}}$ is smaller the possibility of a local improvement is higher because the configuration is only evaluated on a small number of ILP problems.

A further remark on the comparison between the configurations regards the moment when an improvement is found. When there is a point on the line in Figure 4 where a new improvement is found; often thereafter a number of improvements follow in a short amount of time. The improvements are found because after a new improvement is found and are only small changes which give a slightly better improvement. They are found shortly after an improvement is found because only the evaluated configurations are changed partially per iteration of the ParamILS framework. Therefore parts of the configuration that perviously where unchanged result a in slight improvement. When the entire configuration is changed all the easily found improvements are done and finding a new improvement becomes more rare. Another explanation is that some configurations perform better on a subset of the ILP problems. Because the ParamILS framework is iterating over the total ILP problems it takes a while before this subset is reached. However, when it is reached multiple improvements can be found in each others neighbourhood.

A last remark is comparing the differences in order size of the improvements between the configurations in Figure 4. The performance of configuration $O_{\text{obj}} = 200$ and $O_{\text{obj}} = 10$ does not compare to $O_{\text{obj}} = 50, 100$. Due to locality the results could be deviating. For $O_{\text{obj}} = 200$ it is still likely that convergence towards the performance $O_{\text{obj}} = 50, 100$ occurs after some amount of time. However the performance of $O_{\text{obj}} = 10$ is deviating in such a large order that a local optimum is the most reasonable explanation for this performance. The exact speedups obtained via the different configuration scenarios are given in Table 3.

Table 3: The outcome of the automated algorithm configuration runs for each of the different configuration scenarios measured in speedup over the baseline.

<table>
<thead>
<tr>
<th>$O_{\text{obj}}$</th>
<th>Speedup over baseline [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1140.0</td>
</tr>
<tr>
<td>50</td>
<td>404.0</td>
</tr>
<tr>
<td>100</td>
<td>416.0</td>
</tr>
<tr>
<td>200</td>
<td>298.0</td>
</tr>
</tbody>
</table>
5.2.3 Evaluation of Configuration Scenarios

The different configuration found in the configuration scenarios in the previous section are evaluated on the test set of ILP problems. All the configurations result in an improvement cumulative performance over the baseline. In Figure 5, the configuration scenario of $O_{obj} = 10$ is mapped onto Configuration 1, $O_{obj} = 50$ is mapped onto Configuration 2, $O_{obj} = 100$ is mapped onto Configuration 3 and $O_{obj} = 200$ is mapped onto Configuration 4. For each configuration, including the baseline configuration, the cumulative solving time is shown for the test set of 108 problem instances.

In Figure 5 the baseline measurement, i.e. the default value has the largest cumulative performance.

Figure 5: Cumulative solving time of set of problem instances for best found configurations in the different configuration scenarios.

The configuration scenario $O_{obj} = 10$ is performing the best in Figure 4, while in Figure 5 it is performing as third. The configuration scenario $O_{obj} = 100$ is performing second in Figure 4 and is performing the best in Figure 5.

Each of the configurations can be compared on the time elements regarding solving as in Table 2. The time elements of the test set show in Table 3 that the configurations have different presolve times compared to the default configuration. The presolving leads in some configurations to an improved solve time however in other configurations there is no significant effect. Regarding the standard deviations in Table 5 the configurations have significant differences which could be the result of the test set.
Table 4: The outcome of the average times performed on the test set for all the configurations.

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th>Configuration 1</th>
<th>Configuration 2</th>
<th>Configuration 3</th>
<th>Configuration 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load time</td>
<td>1.05</td>
<td>1.90</td>
<td>1.29</td>
<td>2.09</td>
<td>1.91</td>
</tr>
<tr>
<td>Presolve time</td>
<td>0.35</td>
<td>1.31</td>
<td>0.40</td>
<td>1.25</td>
<td>1.27</td>
</tr>
<tr>
<td>Solve time</td>
<td>37.79</td>
<td>15.00</td>
<td>12.54</td>
<td>10.57</td>
<td>15.30</td>
</tr>
<tr>
<td>Total time</td>
<td>39.20</td>
<td>18.22</td>
<td>14.23</td>
<td>13.78</td>
<td>18.48</td>
</tr>
</tbody>
</table>

Table 5: The outcome of the standard deviation for the test set for all the configurations.

<table>
<thead>
<tr>
<th></th>
<th>Default</th>
<th>Configuration 1</th>
<th>Configuration 2</th>
<th>Configuration 3</th>
<th>Configuration 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load time</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>Presolve time</td>
<td>0.03</td>
<td>0.32</td>
<td>0.03</td>
<td>0.23</td>
<td>0.50</td>
</tr>
<tr>
<td>Solve time</td>
<td>48.40</td>
<td>24.42</td>
<td>37.67</td>
<td>37.12</td>
<td>49.88</td>
</tr>
<tr>
<td>Total time</td>
<td>48.40</td>
<td>24.42</td>
<td>37.67</td>
<td>37.12</td>
<td>49.88</td>
</tr>
</tbody>
</table>

5.2.4 Assessing the Different Configuration Scenarios

In this section an assessment is made of each of the configurations that are found in the previous section. The parameters in a configuration can be divided by parameter type. The parameter type of a parameter is determined by the ILP solver. The ILP solver that is used LpSolve. From its parameters that have been used there 7 categorical parameters and 58 binary parameters. An overview of the parameters is given in Appendix D. A binary parameter is regarded as special type of categorical parameters. These parameters represent functionality of the ILP solver; therefore the parameters can be grouped based on their functionality. The functionality groups of the ILP solver are:

- Branch-and-bound(B): Solving strategy used for searching the optimal feasible solutions.
- Scaling(S): Adjusting the values of the constraints of the ILP problem in order to compare the constraints.
- Pivoting(P): Solving rules directing the search strategy.
- Crash mode(C): Introducing artificial elements for improving the solving.
- Simplex(X): Solving strategy used to determine the feasible solutions.
- Iterative Improvement(I): Improvement on the ILP problem during the solving.
- Naming(N): Regards the inclusion of predetermined naming of the constraints and variables of the ILP problem.
- Degeneracy(D): Activities used for solving to prevent cyclical iterations and deciding on improved solutions.
- Presolving(L): Activities performed by the ILP solver to enhance the solving of the ILP problem.
- Factorisation(F): Decomposition of the ILP problem in order to provide an improved structure of the ILP problem.

The parameters of the found configuration in the previous section can be compared per functionality group. In Table 6 all the configurations are shown and which functionality groups are included in the parameter.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>B</th>
<th>S</th>
<th>P</th>
<th>C</th>
<th>X</th>
<th>I</th>
<th>N</th>
<th>D</th>
<th>L</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Config</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Configuration 1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Configuration 2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Configuration 3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Configuration 4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

The functionality groups do not make provide any distinctive information between the configurations and the default configuration. Almost all of the groups are included. Therefore the unique parameters of each of the configurations are examined in the next section.

### 5.2.5 Differences of the parameters per configuration

Each configuration has its own unique parameters which do not occur in any of the other configurations. The unique parameters are shown below. A complete overview of all the parameters in the configuration are given in Appendix C.


Configuration 2: "B5", "BG", "Bw", "improve2", "pivll", "presolvecol", "sp".

Configuration 3: "degeni", "norownames", "pivh", "presolveq".

Configuration 4: "B2", "Bb", "C0", "degenl", "simplexpp".

The difference in unique parameters shows that performance cannot be determined by a single parameter. It is the effect of the composition of the parameters that determines the effect on the performance. Determining the effect of each on the individual parameters is therefore non trivial since the effect of the individual parameters has to be tested on the entire set of ILP problems.
6 Prediction of Solver Configuration for ILP Problems

In the previous chapter a number of configurations are obtained via different configuration scenarios. It is shown that there are different configurations providing an optimal performance for the ILP problems. In this section we try to predict the most optimal configuration of an ILP problem using the information which is known before the solving of an ILP problem. Of all the possible configurations the most optimal configuration for an ILP problem is obtained by the configuration with the minimum performance for that ILP problem. At forehand the configuration with the minimum performance is unknown. In order to select the most optimal configuration for a given ILP problem a prediction of this configuration has to be made based on the information known at forehand. To make such a prediction, features of the ILP problem have to be used which provide information about the ILP problem. These features are extracted from the matrix representing the ILP problem, which is known a priori, i.e. before solving the ILP problem. After the prediction of the configuration is made it’s compared to the most optimal configuration, in order to find out if the most optimal configuration is found.

The prediction of the configuration on an ILP problem is made using a decision tree. A decision tree makes a classification of a categorical parameter by using other parameters. These parameters used for the classification of the categorical parameter can be chosen arbitrarily. An example of a decision tree is given by the case where these parameters represent characteristics of a categorical parameters as given in the data set of Table 7. In Table 7 data is given about the characteristics of the categorical parameter Healthy Lifestyle. This categorical parameter represents whether a person has a healthy lifestyle. The characteristics that determine if a person has a healthy lifestyle are given by the other parameters, i.e. the by the columns Hobby, Education and Savings Account. The parameters Hobby is the kind of sport of a person exercisers, Education is the highest form of education a person completed and Savings Account is the amount of money on his current savings account.

<table>
<thead>
<tr>
<th>ID</th>
<th>Hobby</th>
<th>Education</th>
<th>Savings Account (K)</th>
<th>Healthy Lifestyle</th>
</tr>
</thead>
<tbody>
<tr>
<td>004</td>
<td>Athletics</td>
<td>University</td>
<td>40</td>
<td>Yes</td>
</tr>
<tr>
<td>005</td>
<td>Cycling</td>
<td>High School</td>
<td>2</td>
<td>No</td>
</tr>
<tr>
<td>006</td>
<td>Athletics</td>
<td>University</td>
<td>35</td>
<td>Yes</td>
</tr>
<tr>
<td>007</td>
<td>Athletics</td>
<td>High School</td>
<td>12</td>
<td>No</td>
</tr>
<tr>
<td>008</td>
<td>Cycling</td>
<td>High School</td>
<td>8</td>
<td>No</td>
</tr>
</tbody>
</table>

Consider a prediction tree based on the data in Table 7. The decision tree makes its prediction by splitting the data on specific values. In this manner the decision tree makes nodes which form a tree-structure as shown in Figure 6. Take for example splitting the example data of Table 7. In this example there are two persons who have a healthy lifestyle. The decision tree will use the other parameters Hobby, Education and Savings Account to split their value of the parameters Healthy Lifestyle so it can predict whether a person has a healthy lifestyle.
Using the parameter *Hobby*: the value "Athletics" is common for the cases which have a healthy lifestyle. Meaning that the decision tree splits the parameter *Hobby* and when the value contains "Athletics" is predicted as a healthy lifestyle. Including another parameter changes the prediction tree. For the next parameter *Education* there are two cases which have a healthy lifestyle. The value of the parameter *Education* for both of these cases equals "University". Whenever the decision tree uses the parameters *Hobby* and *Education* there is one combination for which case there is a healthy lifestyle. The combination (*Hobby:* "Athletics", *Education:* "University"). The parameter *Savings Account* is a numerical parameter. Splitting this parameter can be done choosing an arbitrary numerical value. There are two cases for which this parameter has a healthy lifestyle which have both a value larger than 30 K. Combining the two parameters with the parameter *Spending Pattern* gives the total decision tree learning rules. The number of combinations remains equal. The total of the combination is (*Hobby:* "Athletics", *Education:* "University", *Spending Pattern:* "≥ 30K"). The example of the learning rules can be put into a graphical notation representing the decision tree. An example of a graphical notation of a decision tree is given in Figure 6. For simplicity is the feature *Savings account* is not included in this example.

Figure 6: A decision tree for the healthy lifestyle of a person based on the predictor parameter *Hobby* and *Education*.

### 6.1 Features of the ILP Problem’s Matrix

The prediction of the ILP problems is based on the configurations found in the different runs of the configuration scenarios. For each run performed in Chapter 5 the last configuration of that run is selected. The runs provide therefore 4 different configurations. Including the default configuration in total there are 5 different configurations. Each ILP problem is solved using these configurations. The best performing configuration is chosen as the most optimal configuration for an ILP problem. The categorical parameter used for the decision tree is therefore the most optimal configuration.

The characteristics of an ILP problem on which the prediction tree is based have to be known before solving an ILP problem. The parameters representing these characteristics have to be created before the solving. To create the parameters we have to take into account the information that is known before solving. The information known before solving is the matrix representation of the ILP problem. An example of a matrix representing an ILP problem in Table 8.

The matrix representation of an ILP problem contains characteristics which can be used by the decision tree. To provide the decision tree these characteristics we put these characteristics into parameters. The
used parameters are extracted features of the ILP problem’s matrix. In general more characteristics of the processes execution data can be provided into the decision tree, however the scope is limited to characteristics related to the ILP problems matrix.

The matrix of the ILP problems is build up by linear equations which can be subdivided into: the objective function, the constraints and the right-hand side variables. For each ILP problem a set of features can be created from the ILP problem’s matrix using its corresponding the objective function, the constraints or the right-hand side variables.

Table 8: A matrix representation of an ILP problem

<table>
<thead>
<tr>
<th>Z</th>
<th>3</th>
<th>5</th>
<th>12</th>
<th>11</th>
<th>2</th>
<th>5</th>
<th>7</th>
<th>18</th>
<th>-1</th>
<th>-3</th>
<th>-9</th>
<th>-8</th>
<th>-6</th>
<th>-5</th>
<th>-7</th>
<th>= 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c9</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
<td>≥ 0</td>
<td></td>
</tr>
<tr>
<td>c12</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 1</td>
<td></td>
</tr>
<tr>
<td>c13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>≥ 1</td>
<td></td>
</tr>
</tbody>
</table>

An example of a feature which extracted from the matrix is the number of rows in the matrix of Table 8. The objective function of this matrix is provided in the first row, the constraints are denoted with the letter ”c” in the name of the constraint. When this feature is applied to the matrix in Table 8 the number of rows equals 13, equal to the number of constraints. Other features use more complicated characteristics of the matrix. For example a characteristic of the matrix often used in the features is the position of the variables of which the right hand side are equal or larger than one. In the matrix in Table 8 these are constraints 12 and 13, which have the position of counting from the left the 3th and 10th column. In the matrix these constraints are important because they contain distinctive information about the matrix. These constraints are unique for each ILP problem. Indicating that the ILP problems generated from the same process execution data only differ on these constraints. Many other features are used. An overview of all the 39 features is given in Appendix A where all the used features are explained in detail.

For the prediction of the decision tree a new set of ILP problems is generated. The new set of ILP problems is generated because the sample size of the previous set of ILP problems is not sufficient learning a decision tree. The new set of ILP problems is therefore about ten-fold of size of the previous set of ILP problems. The generated set is divided into a training set and test set by using previously mentioned 80/20 model. In total, a number of 5728 ILP problems are generated for the learning and the
evaluation of the decision tree. The training set consists of 4582 ILP problems and the test set consists of 1146 ILP problems. After the features are extracted on the training set of ILP problems they can be used for the learning and the evaluation of the decision tree.

Extracting the features from the matrices will require some additional time. However, the features can be extracted at the same time the ILP problems are generated. The influence of extracting the features on the performance is out of scope. Therefore, it is assumed that it's possible to generate the ILP problems and at the same time extract the features without any significant additional time.

6.2 Learning the Decision Tree

Several techniques can be used to obtain a decision tree. The specific techniques of learning a decision tree and the differences between the settings is out of the scope of this research. Standard settings of software enabled to learn a decision tree. The software program is described in Appendix B. In general the decision tree of this software tool uses the Gini Index and no pruning. More information about the settings of this software tool can also be found in Appendix B.

After a decision tree is learned on the training set it is shown that the decision tree mainly uses three different features to categorise ILP problems. The most common used features inverse-constraint-, inverse-variable-, sparsity-feature and the relative-position-feature are explained in detail.

The inverse-constraint-feature is a feature which is a combination of multiple characteristics of the matrix. It uses the positions of the variables in the constraints which right-hand side is equal or larger than one. Whenever the constraints contains a variable on this position then the absolute number of variables in the constraint are counted. As example in the matrix of Table 8 one variable which right hand side is equal or larger than one has its position in the 3th column. Take the 4th constraint of the matrix which includes this variable in this constraint there are 6 other variables. Therefore the the absolute sum of variables is 6. The inverse is taken from the sum of all the constraints. Next this sum is divided by the sum of the total number of constraints. Last the calculation is divided by the total number of number of variables in the matrix.

The inverse-variable-feature is equal to the inverse-constraint-feature except for the normalisation by the number of variables. For this feature the normalisation step performed in the inverse-constraint-feature is replaced by normalisation performed on the number of constraints.

The sparsity-feature is using the elements in the matrix and compares them on their coefficients. For example for the matrix in Table 8 the coefficient for the element on the first row and first column is equal to one. Whenever the coefficient of an element in the matrix is equal to zero the variable is counted. In this manner the total number of zeroes and non-zeroes can be determined. The total number of non-zero elements is divided by the total number of elements in the matrix to determine the sparsity.

The relative-position-feature takes the constraints with the right-hand side equal or larger than one and relates them to their position regarding the other variables. The relative position is measured from the distance to the first variable. The explanation for using the feature is that it is related to the working of the solving algorithms, which often start at the most left-sided variable solving the matrices. When these variables are more at the left side of the matrix this could influence solve time since the solving
always starts with the elements on the left side of the matrix.

6.3 Prediction Quality

The quality of the decision tree, i.e. how well does the decision tree predict. This is measured by comparing the performance of the predicted configuration against the performance of the optimal configuration. For all the instances in the test set the optimal configuration is determined. A prediction is made for the ILP problem in the test set using the decision tree learned on the training set.

The evaluation of the prediction against the optimal configuration is given in a confusion matrix in Table 9. In the confusion matrix there are different measurements which indicate the correctness of the prediction. The decision tree makes a prediction for a certain ILP problem. Whenever the prediction is correct, i.e. the prediction is equal to the optimal configuration, it is defined as TruePositive. Whenever the prediction is not correct it is stated as FalsePositive. Also, the decision tree can make a prediction that an ILP problem is not optimal for a certain configuration. Whenever the TruePositive and FalsePositive are known also the TrueNegative and FalseNegative are determined because they are the reciprocal of each other. The combinations TruePositive, FalsePositive, TrueNegative and FalseNegative define the quality of the decision tree prediction. An overview of the combinations is given in Table 9. Overall the quality of the prediction is measured using the accuracy. The accuracy measures the fraction of correct prediction configurations, i.e. the TruePositive instances over the total number of ILP problems. The accuracy for the learned decision tree is about 0.57. The accuracy is given in Table 9.

<table>
<thead>
<tr>
<th></th>
<th>TruePositive</th>
<th>FalsePositive</th>
<th>TrueNegative</th>
<th>FalseNegative</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1</td>
<td>8</td>
<td>1121</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Configuration 1</td>
<td>467</td>
<td>216</td>
<td>324</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>Configuration 2</td>
<td>113</td>
<td>103</td>
<td>784</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>Configuration 3</td>
<td>18</td>
<td>47</td>
<td>1012</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>Configuration 4</td>
<td>51</td>
<td>122</td>
<td>324</td>
<td>139</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.57</td>
</tr>
</tbody>
</table>

The confusion matrix does not allow us to deduce what the effect of an incorrect prediction is for the optimal performance. An incorrect prediction can still have an increase performance over the default performance of an ILP problem. The confusion matrix used for prediction configuration has a drawback when the predictor variable is an ordinal variable. In order to compare the effects of the prediction the predicted configurations will be solved for the test set and compared to the other configurations.

The test set is solved for the default-, prediction- and configurations as found in Chapter 5. The cumulative time of all of these configurations taken. All the ILP problems in the test set are solved using a time-out value of 1800 seconds. The time-out value is changed from previous used settings because the ILP problems generated in the new set differ from previous set of ILP problems. The time-out values is taken a ten-fold from the original value to ensure that there is enough time to solve these new
ILP problems. Whenever the ILP problem is stopped by the time-out value the ILP problem is solved as sub-optimal. The results of solving the ILP problems with these settings are shown in Figure 7. The results in Figure 7 show that the default configuration is the worst performing configuration. The default configuration performs about four times as slow as the optimal configuration. It is shown that the best performing configuration is Configuration 3. The prediction configuration is not performing very well. The prediction configuration also includes the time for making the prediction of the decision tree. However, this time is insignificant compared to the cumulative time as it is less than a percent of the cumulative time of the prediction configuration. The difference between the configurations is mainly made at the points where some configurations remain constant and other configuration increase stepwise. This indicates that the difference occurs when an ILP problem is solved sub-optimal and the time-out value is reached. When a time-out value is reached it will look as a stepwise increase in Figure 6.3. It seems that the configuration that has the best performance also has the most optimal solved ILP problems. To indicate this relationship the configurations have to be compared on quality, i.e. the number of optimal solved ILP problems.

Figure 7: The cumulative time of the (default) configurations, the prediction and the optimal configuration.
After solving each configuration, the ILP problems are evaluated in terms of optimality. Whenever an ILP problem is solved sub-optimal this influences the performance the most because the maximum time is used. Therefore the percentage of optimal solved ILP problems is measured. In Figure 8 the cumulative percentage of optimal solved ILP problems is given. It is shown that the cumulative optimal solved ILP problems are directly related to the order of the cumulative time. The order of the configurations in Figure 7 is the direct reverse order of the configurations in Figure 8. For example the default configuration has the highest cumulative time and the lowest cumulative percentage of optimal solved ILP problems. The difference in Figure 7 are therefore directly related to the time-out value. The configuration which is performing the best in Figure 8 on the cumulative percentage will therefore perform proportionally better according the value of the time-out. The prediction configuration is not able to overcome this effect by prediction more configurations which are optimal than the best configuration.

![Cumulative optimal of solved instances for test set](image)

Figure 8: The cumulative percentage of optimal solved ILP problems of the (default) configurations, the prediction and the optimal configuration.

The results of the cumulative performance in Figure 7 and the cumulative optimal solved ILP problems in Figure 8 show that there is an improvement possible. Still a gap exists between the best found configuration and the optimal configuration. Using the decision tree prediction technique it is not possible to narrow the gap between these configurations. It could be that the features do not provide enough predictive information to predict the correct configurations and therefore the gap is not able to be closed. However finding better features still improves the performance and obtains a better quality as a consequence of optimal solving the ILP problems.
7 Conclusion

In this research we applied automated algorithm configuration for ILP-based process discovery. Multiple automated algorithm configuration methods are evaluated on the criteria related to the solving the underlying ILP problems of process discovery. The evaluation is based on the type of parameter, number of parameters, convergence properties and improvement quality. After the evaluation it showed that the ParamILS framework is the method which is best suited for the needs of ILP-based process discovery.

The ParamILS framework is applied on a set of ILP problems derived from a region-based process discovery algorithm. Different configuration scenarios are used for performing automated algorithm configuration on these ILP problems. Each of the configuration scenarios resulted in a different optimal configuration. In total four different parameter configurations lead to an improved performance over the default configuration of the ILP-solver.

The different configuration scenarios show that there are multiple configurations which are optimal for solving the ILP problems. To determine the optimal configuration a prediction is made with the information known before the solving of the ILP problem. A decision tree technique predicts the optimal configuration per ILP problem based on features of the matrix of the ILP problem. A new set of ILP problems is generated to learn the decision tree. Evaluating the configuration with the minimum performance against the predicted configuration shows that the accuracy of the prediction tree is about 57 percent. Comparing the results of the predicted configuration against the other configurations it is shown that the predicted configuration is outperformed by the best configuration in the test set. Since there is still a gap between the performance of the best performing configuration and the optimal configuration improvement is still possible.

Future Work

Future work can improve many aspects of the performance of solving the ILP problems for process discovery. Improving the prediction methods used directly influences the performance. It is shown there is still a gap between the optimal configuration and the best performing configuration because of the insufficient accuracy. Future work can focus on making better predictions, using other prediction techniques. Related to the prediction are the features used for making the prediction. In recent work for example deep learning or bayesian learning frameworks the features are automatically generated, which is not the case in this research. Automatically finding better features can improve the prediction accuracy which therefore leads to better performance.

Another improvement can be found when changing the structure of the ILP problems. Whenever the coefficients of an ILP problem are changed, i.e. changing the ordering of the constraints. This does not change the outcome of solving the ILP problem. Many research is performed on this fill-reducing ordering which influences the performance of an algorithm on an ILP problem. It is in fact the combination of the algorithm and the structure that influence the performance of the ILP problem.

Another improvement can be made on the ILP-solver. The ILP-solver uses algorithms which are not specific designed for ILP-based process discovery. The current structure of the ILP problem causes that the ILP-solver is doing rework for multiple ILP problems. Since a part of the constraints used in the ILP problems are overlapping. The improvement could result in an increase in performance whenever the solving of the overlapping part of the ILP problem is used in later solving of other ILP problems.
A Appendix A

In this appendix the features used for the decision tree are explained. It has to be noted that the ILP problem can contain multiple i-th variables.

**Matrix Columns**: This feature is equal to the total number of columns of the matrix.

**Matrix Rows**: This feature is equal to the total number of rows of the matrix.

**Total Elements**: This feature is equal to the product of the rows and columns of the matrix.

**Sparsity**: This feature is equal to the total number of non-zero elements of the matrix divided by the total number of elements.

**Constraint(i)**: This feature is the position of the i-th constraint of the matrix. The i-th constraint is the constraint for which the right-hand side is larger or equal than 1.

**Objective Value**: This feature is difference between the objective-value of the different i-th constraints. A smaller difference means that less decision variables have to be used in order to minimise this difference.

**Column Sum**: This feature measures the sum of the variables in a constraint when the difference between the i-th variable is present in that constraint.

**Row Sum**: This feature measures the sum of the constraints when the difference between the i-th variable is present in that constraint.

**Column Sum(i)**: This feature measures the sum of the variables in a constraint when the i-th variable is present in that constraint.

**Row Sum(i)**: This feature measures the sum of the constraints when the i-th variable is present in that constraint.

**Row position**: This feature measures the sum of the positions of the constraints when the i-th variable is present in a constraint. The first variable is defined at position zero.

**Column position**: This feature measures the sum of the positions of the variables when the i-th variable is present in a constraint. The first variable is defined at position zero.

**Row position relative**: This feature measures the sum of the positions of the constraints when the i-th variable is present in a constraint. This is divided by the total number of positions to obtain the relative position. The first variable is defined at position zero.

**Column position relative**: This feature measures the sum of the positions of the variables when the i-th variable is present in a constraint. This is divided by the total number of positions to obtain the relative position. The first variable is defined at position zero.
Row position(i): This feature measures the sum of the positions of the constraints when the i-th variables are present in a constraint. The difference of these positions is taken. The first variable is defined at position zero.

Column position(i): This feature measures the sum of the positions of the variables when the i-th variables are present in a constraint. The difference of these positions is taken. The first variable is defined at position zero.

Row position relative(i): This feature measures the sum of the positions of the constraints when the i-th variables are present in a constraint. The difference of these positions is taken. This is divided by the total number of positions to obtain the relative position. The first variable is defined at position zero.

Column position relative(i): This feature measures the sum of the positions of the variables when the i-th variables are present in a constraint. The difference of these positions is taken. This is divided by the total number of positions to obtain the relative position. The first variable is defined at position zero.

Inverse: This feature is the calculation of (1 / Row Sum / Column Sum).

Inverse Row: This feature is the calculation of (1 / Row Sum / Column Sum / Matrix Rows).

Inverse Column: This feature is the calculation of (1 / Row Sum / Column Sum / Matrix Columns).
Appendix B

The program KNIME Analytics Platform is used to build the decision tree. The version of KNIME Analytics Platform is version 12.11.3.

The decision tree node in KNIME is used given the settings:

- **Quality measure: ”Gini Index”**
  The Gini index is a measurement of how often a random chosen element from a set is incorrectly classified according to the distribution of a subset.

- **Pruning Method: ”No pruning”**
  Pruning is a technique of reducing the tree by removing instances that will not apply to a certain pruning metric. In this manner the number of learning rules is reduced and therefore the depth of the decision tree is reduced as well.

- **Minimum records per node : ”4”**
  The number of records that have to occur per node before the decision tree classifies an instance to this node.

- **Average split point: ”True”**
  When numerical variables are used in the data set, then the decision tree may use the average numerical value to split these value of variables.

- **Thread per node: ”8”**
  The number of threads per node determine the number of processors used per node in the decision tree learning.
C  Appendix C

The default configuration includes the parameters:

\[-B0 \text{-piv2 -s1 -si -se -improve2}.\]

The next configuration is Configuration 1 which includes the parameters:

\[-B0 \text{-BB -BR -Bc -Bd -Bf -Bi -Bo -Bp -Br -C -bfp bfp etaPFI -degen -degenf -degenn -degenp -degens -improve4 -nocolnames -nonames -piv1 -piva -pivf -pivla -presolvebnd -presolvec -presolvecold -presolved -presolvefd -presolvek -presolve -presolveowd -presolve -s1 -se -si simplexd}.\]

The following configuration is Configuration 2 which includes the parameters:

\[-B5 \text{-BB -BG -Bc -Bi -Bo -Bw -C2 -bfp bfp etaPFI -degen -degenf -degenn -improve2 -piv2 -piva -pivla -pivll -pivr -presolvecol -presolvef -presolvefd -presolveg -presolveowd -presolve -presolveeslk -s1 -si -sp}.\]

The next configuration which is Configuration 3 includes the parameters:

\[-B0 \text{-Bc -Bi -Bo -Bs -C2 -bfp bfp etaPFI -degen -degenb -degenf -degeni -degenp -nocolnames -norownames -piv2 -pivh -pivm -pivr -presolvebnd -presolvef -presolvefd -presolveg -presolveq -presolve -presolveeslk -s1 -se -si -simplexd}.\]

The last configuration is Configuration 4 which includes the parameters:

\[-B2 \text{-BB -BR -Bb -Bd -Bi -Bo -Bs -C0 -bfp bfp etaPFI -degenb -degenp -degenl -improve1 -nocolnames -nonames -piv1 -pivm -pivr -presolve -presolvefd -presolveg -presolvek -presolve -presolveeslk -s1 -se -si -simplexpp}.\]
D Appendix D

Functionality groups

Branch-and-bound:
B0
B1
B2
B3
B4
B5
B6
Bw
Bb
Bg
Bp
BR
Bf
Br
BG
Bd
Bs
BB
Bo
Bc
Bi

Scaling:
s0
s1
s2
s3
s4
s5
s6
s7
sp
si
se

Pivoting:
piv0
piv1
piv2
piv3
pivf
pivm
piva
pivr
pivl
pivla
pivh
pivt

Presolving:
presolve
presolverow
presolvecol
presolve
presolves
presolver
presolvek
presolveq
presolvem
presolvefd
presolvebnd
presolved
presolvef
presolveslk
presolveveg
presolveb
presolvevec
presolverowd
presolvecold

Crash mode:
C
C0
C2

Simplex:
simplexpp
simplexdp
simplexpdp
simplexdd

Iterative improvement:
improve0
improve1
improve2
improve4
improve8
Degeneracy:
  degen
degenc
degend
degenf
degens
degenn
degenu
degenl
degeni
degencb
degenr
degenp

Names:
  nonames
  norownames
  nocolnames

Factorsation:
  bfp bfp etaPFI
  bfp bfp LUSOL
  bfp bfp GLPK

Categorical parameters:
- (B0, B1, B2, B3, B4, B5, B6)
- (s0, s1, s2, s3, s4, s5, s6, s7)
- (piv0, piv1, piv2, piv3)
- (C, C0, C2)
- (simplexpp, simplexdp, simplexpd, simplexdd)
- (improve0, improve1, improve2, improve4, improve8)
- (bfp bfp LUSOL, bfp bfp etaPFI, bfp bfp GLPK)
REFERENCES

References


