BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

AUTOMATIC DETECTION OF PERFORMANCE DEGRADATION

AUTOMATICKÁ DETEKCE DEGRADACE VÝKONU

PROJECT PRACTICE
PROJEKTOVÁ PRAXE

AUTHOR **ŠIMON STUPINSKÝ**
AUTOR PRÁCE

SUPERVISOR **Doc. Mgr. ADAM ROGALEWICZ, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2018

Contents

Chapter 1

Introduction

Measuring of the performance is a critical factor in optimizing. Optimal performance is sustainably achieving multiple, often conflicting, objectives under changing conditions. Manual control of performance of the application is difficult and poses several challenges. First, the responsibility of running performance regression tests is shifted to users. The second challenge is precise preservation of the history, mainly because results of previous tests can be lost. And the last is sustaining the context of the executed performance test. Though many projects are supervised by Version Control Systems (VCS), they pose a certain limitations.

In particular, these systems are missing more precise context for storing profiles and already mentioned automation. Moreover, they are not friendly for users, because they require manual annotation, and manually added hooks. Some of these problems could be resolved by using a database to store, but by using it one has to solve a brand new challenges. The database systems are robust, too centralized and most of all too generic. However, we would like a system, which will be light-weight, distributed and instead of generic we need something specific for profiles.

These calls for solutions are directed to performance management. Systems with a similar role are called Performance Versioning Systems, and one of them is a tool Perun, which will be introduced in Chapter 2. Perun addresses the mentioned challenges and manages the precise history of the performance of a particular projects. However, the interpretation processes, whether any performance change happened with new version of the project is still left to the user. Thus the main goal of this work is to explore the possibilities of automating the detection of the performance degradation (resp. improvement) between two project version based on statistical methods.

The rest of this report is structured as follows. In Chapter 3, will be introduced principles of processing the performance data and their uses for analysis of performance. After that, the main idea of the automated analysis will be proposed, mostly in terms of input and output, in the Chapter 4. In the next chapters, the design of the automatic detection will be shown, with the usage of various statistical methods. The achieved results will be summarised in the Chapter 6.

Chapter 2

Perun

Perun (Performance Under Control)[1] is an open source light-weight Performance Versioning System. It serves as a wrapper over the existing version system and takes care of managing profiles for a broad range of project types. Its main goal is to enable easy and automatized management of profiling of a program in course of its development.

Moreover, it offers a suite of tools allowing to reduce the whole process of to a single command, for the purpose of automation of regression tests of the performance. Another advantage against usual version control systems is the possibility of postprocessing of stored profiles or their interpretation. Each such performance profile, based on `JSON` notation, is assigned to a concrete project minor version, adding the missing context to profiles.

2.1 Architecture of Perun

Internal architecture of Perun can be divided into several major units. The overall architecture is described in the Figure 2.1.

The first unit is **data**, which includes manipulation of profile and supported wrappers over the existing version control systems. The next unit is the **logic**, which is liable to manipulation with higher logic, is at a charge of automation and takes care of actual generation of the profiles. The last unit of architecture is **a suite of tools**, which includes collectors for generation of profiles, set of postprocessors for transformation and visualization and wrappers for graphical and command line interface.

Perun architecture was designed to allow simple extension of both its internals and tool suite.

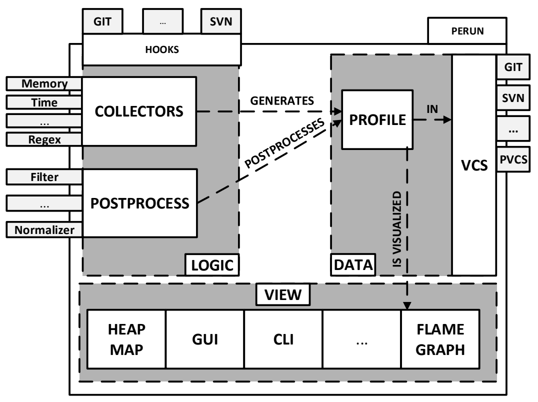


Figure 2.1: The scheme above shows the basic decomposition of the architecture of Perun into the units (logic, data and visualization part). The arrow marks the mutual communication between the modules.

[1]https://github.com/tfiedor/perun

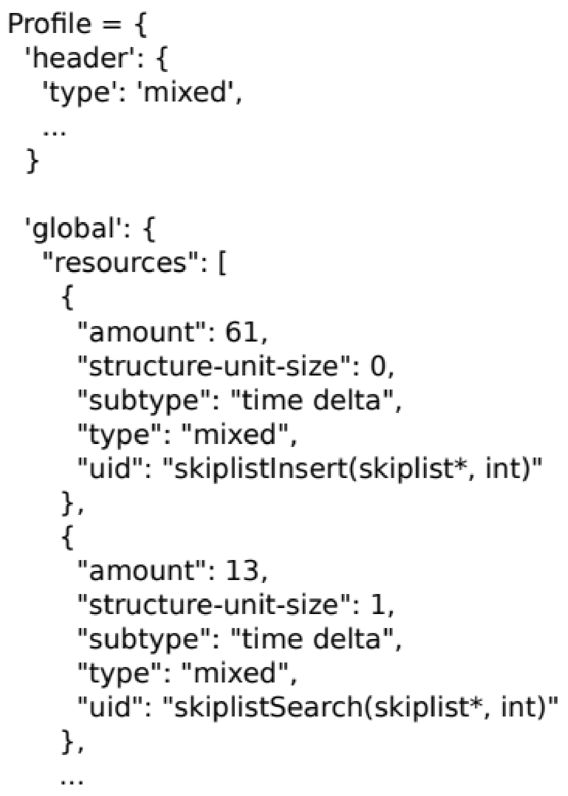
Chapter 3

Profiles and regression models

3.1 Lifetime of a profile

The format of the performance profiles is based on the `JSON`[1] structure. The important property of a profile is a **unification** of various metrics (time, memory, etc.). Usage of the `JSON` format brings advantages such as easy processing and well-supported on many levels. This format is used as a simple, portable interface between all the modules, which manipulate with the profiles. Profiles are structured into a few basic regions, see a picture 3.1.

For every input data (binary file, source file, etc.) collectors are run individually, which generate profiles. These profiles are subsequently processed by using a series of postprocessors. Job matrix, which are the core of automation in Perun, generates a set of profiles. Profiles themselves are stored in a storage. Together with profiles can be stored another information such as a version of a program. Generated profiles can be visualised by using various visualization techniques. That workflow will be devoting ourselves next Chapter 4.



```
Profile = {
  'header': {
    'type': 'mixed',
    ...
  }

  'global': {
    "resources": [
      {
        "amount": 61,
        "structure-unit-size": 0,
        "subtype": "time delta",
        "type": "mixed",
        "uid": "skiplistInsert(skiplist*, int)"
      },
      {
        "amount": 13,
        "structure-unit-size": 1,
        "subtype": "time delta",
        "type": "mixed",
        "uid": "skiplistSearch(skiplist*, int)"
      },
      ...
```

Figure 3.1: Example of the JSON profile.

3.2 Regression models

The statistical methods, which are devoted to finding relationship between measured variables from performance profiles, are implemented in **Regression analysis** module. Specifically, the module focuses on finding the relationship of values of `dependent variables` (e.g. time) to the values of `independent variables` (e.g. size of the underlying structure),

[1]https://www.json.org/index.html

where the result is expressed in the form of the regression function. The regression analysis methods can be divided into several groups, however, in our analyses we chose the **linear regression analysis**. The calculation is carried out over only the two variables, i.e. so called **simple linear regression** [1].

The regression analysis module limits itself such that dependent variables are represented by the running time of the program and independent variables are represented in the form of the size of the data structure. Note that non-linear models (the so-called *intrisically linear*) were transformed to linear models, which includes logarithmic, power and exponential models. The main goal of the module is to find a model, which best fits dependency of time on the size of the structure. Finding such a model then can be solved using e.g. the `method of least squares`, which is used uncurrently in Perun [3].

The result of the analysis are fully computed models of form $y = b_0 + b_1 * f(x)$, represented by their types (e.g. linear, logarithmic, etc.) and concrete coefficients b_0 and b_1. From these computed regression models, it is then necessary to selected one concrete model, which expresses the dependence between variables in the most accurate way and use it as a representant of performance of one minor version. This property is best expressed by `coefficient of determination`, denoted R^2 and is suitable for measuring the fitting of the model.

Chapter 4

Regression between commits

When one commits new changes, in supported version control system, he wants to be sure, that new profiles for each new minor or major version of the project due generated. Perun automatically execute performance regression tests and store the generated profile for every minor version of the Version Control System (e.g. commits) in a given project. The goal of our work is to explore effective heuristics for automatic detection of performance degradation between two project versions (e.g. between two commits). These heuristics are based on statistical methods, mentioned above in chapter 3.2.

4.1 The core of the workflow

After each version of the project release (i.e. commit) the following actions should be run automatically.

First set of collectors are run. Perun currently supports three types of collectors — `complexity`, `memory` and `time` collector. `Complexity` collector collects running times of functions (e.g. C/C++) along with the size of the structures they were executed on. The simplest, `Time` collector, collects overall running time of arbitrary commands. The last collector is `Memory` collector, which collects specifications of allocations in programs, such as the type of allocation or the full call trace. The role of these collectors is to generate profiles.

After the collection of data resulting into performance profile, profiles can be further refined and transformed by a sequence of postprocessing steps. The first option is using of `Normalizer Postprocessor`, which scales the resources of the given profile to the interval (0, 1). Mentioned `regression analysis`, in Chapter 3.2, is the next option.

Profiles are then stored in the Perun directory. These profiles can be further interpreted by a set of visualization techniques like e.g. Flame Graph, Scatter Plot, or generic Bars Plot and Flow Plot.

This process is applied to the each version of the project. Thanks to the fact that resulting models are stored in a Perun directory, two performance points (e.g. two commits) can be compared. Methods to compare two commits and methods of detection of performance degradation between two commits, will be introduced in the next Chapter 5. The scheme in figure 4.1 depicts the overall process of automatic detection of degradation.

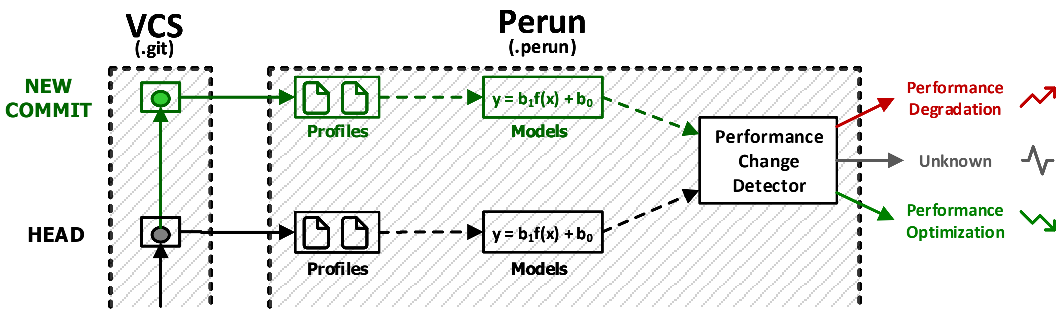


Figure 4.1: Workflow of analysis.

Chapter 5

Methods of performance degradation

In the previous chapters we laid a fundamentals, needed for detection of performance degradation within Perun. In this chapter, we will be introduce methods for this detection, which were researched in the frame of this work.

The core of this process is based on obtained profiles with their relevant models. For these experiments we use profiles, which illustrates the dependence of the *amount* per *structure unit size*. The selected structure was the *Skiplist* [4], because its properties such as a balance or more pointers in the one element, are advantageous for these experiments. While its memory complexity ($O(n \log n)$) lags behind other structures ($O(n)$), on the other hand, *Skiplist* features a high speed of the operations (*Insert, Delete, Search*). An operation selected in our case is *Search*. Its time complexity is in the average case equal to $\Theta(\log n)$.

The profile, created at the beginning for further experiments of this process (see appendix A.1), is a baseline for all other measurements. In the next sections we will gradually reveal procedures of detection performance degradation on the individual models.

5.1 The aspects of detection

Although it seems, that the detection of degradation itself can be sometimes difficult, we can invent heuristics depending of different kinds of error to be detected.

The kind of errors, which we focused on, were `constant`, `linear` and `quadratic` errors. These errors are derived from the type of models. The options of determining an error rate, will be depending on the selected type of statistic method. As we will see later, some methods better fit to detect certain types of errors.

Whereas the methods for detection of regression between two profiles there are several, two, which were used, will be introduced. The first method is to perform the detection between dependent variables themselves. The other is to perform the detection between the best fitting models of both profiles. The selection from these methods then depends on the best fitting models from profiles. If the value of *Coefficient of determination* of best model is high enough, the better option is the second one, otherwise the first.

5.2 Statistical methods

In this chapter we will gradually introduce used statistical methods of detection of performance degradation. The results of these methods will be summed up in the Chapter 6.

5.2.1 Absolute and relative error

The **Absolute** error is the difference between the measured or inferred value and the actual value. Also, it is the amount of physical error in a measurement and it is one of the ways to consider error when measuring the accuracy of values [5]. In our experimentations, this error expresses how both dependent variables vary. In other words, expresses difference between the new dependent variables and their baseline values. Absolute error is expressed by the following formula:

$$\Delta x_i = f(x_i) - f_b(x_i), \tag{5.1}$$

where, $\mathbf{\Delta x_i}$ is the absolute error, $\mathbf{f_b(x_i)}$ is the value of the dependent variable from the baseline model and $\mathbf{f(x_i)}$ is the value of the dependent variable of the target profile.

The **Relative** error models of how it good measurement is relative to the size of the thing being measured. In other words, it is the ratio of absolute error to one of the measurement values [6]. This error has a big meaning in our experiments, because it has the potential to express how much degradation occurred. Relative error is expressed by a formula:

$$\delta x_i = \frac{f(x_i) - f_b(x_i)}{f_b(x_i)} = \frac{\Delta x_i}{f_b(x_i)} = \frac{f(x_i)}{f_b(x_i)} - 1, \tag{5.2}$$

where, $\delta\mathbf{x_i}$ is the relative error, $\mathbf{\Delta x_i}$ is absolute error, $\mathbf{f_b(x_i)}$ is the value of dependent variable from the baseline model and $\mathbf{f(x_i)}$ is the value of dependent variable of the target profile.

5.2.2 Studentized residual

The residual is the difference between a predicted value and the observed value. The **Studentized residual** is the quotient resulting from the division of a residual by an estimate of its standard deviation. This turns out to be equivalent to the ordinary residual divided by a factor, that includes the *mean square error* based on the estimated model with the i^{th} observation, MSE_i and the *leverage*.

Mean squared error is defined as the average of squares of the errors and it is expressed by the formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - f_b(x_i))^2, \tag{5.3}$$

where, **MSE** is *mean squared error*, **n** is the size of the data structure, $\mathbf{f_b(x_i)}$ is the value of the dependent variable from the baseline model and $\mathbf{f(x_i)}$ is the value of the dependent variable of the target profile.

Leverage is a measure of how far an independent variable deviates from its mean. Leverage points can have an effect on the estimate of regression coefficients and is expressed by the formula:

$$h_i = \frac{(f(x_i) - \bar{X})^2}{\sum_{j=1}^{n}(f(x_j) - \bar{X})^2} + \frac{1}{n}, \tag{5.4}$$

where, $\mathbf{h_i}$ is the leverage, $\mathbf{X}$ is the mean value of dependent variables of the target profile, $\mathbf{n}$ is the size of the data structure, $\mathbf{f_b(x_i)}$ is the value of the dependent variable from the baseline model and $\mathbf{f(x_i)}$ is the value of the dependent variable of the target profile [2].

After the simplification of these formulae, one can define the formula of the studentized residuals as:

$$t_i = \frac{\Delta x_i}{s(d_i)}, \tag{5.5}$$

where, $\mathbf{\Delta x_i}$ is the absolute error, $\mathbf{s(d_i)}$ is the standard deviation of absolute error and further as:

$$t_i = \frac{\Delta x_i}{\sqrt{MSE_{(i)}(1-h_i)}}, \tag{5.6}$$

where, $\mathbf{\Delta x_i}$ is the absolute error, $\mathbf{MSE_{(i)}}$ is the mean squared error, $\mathbf{h_i}$ is the leverage.

The difference between the standartized residual and the studentized residual, is this, that standartized residuals use the MSE for the model based on all observations, while studentize residuals use the MSE based on the estimated model with i^{th} observation.

Chapter 6

Experimental Evaluation

Profiles, which are part of our experiments, are assumed to be profiles with an ideal properties. It mainly means that there is at least one model, fitting the profile well. This property of the profiles is a guarantee, that one can try to detect the regression between the models instead of real values. To simulate real conditions, random noise was applied to the final results, as a multiple of the original values at a rate of one percent.

In the following sections, the results of the analysis of degradation will be presented. As we already mentioned, errors which have been examined in these studies are **constant**, **linear** and **quadratic** errors and "no degradation". All types of errors were analysed by the statistical methods mentioned in Section 5.2 and were analyzed on all types of models.

6.1 No error

Represents the situation, when almost no change between the two profiles is identificated. However, detection of a situation, when no degradation has occurred, is still important. Not every situation, where regression profiles are not almost the same, necessarily means degradation of performance. In these situations, the question arises, which value of the threshold is appropriate for determining the limit of the error.

The situation, when both profiles are almost the same, is simply identificated by using the **sum of the absolute errors**. Its value has to be close enough to a zero. This assumption was experimentally confirmed by performing the tests between models of the two identical profiles, of course with the noise applied, see the Appendix A.2 for graphical visualization.

Suitability of the **sum of the absolute errors** for detection of no degradation is a discussed in Appendix A.3. In this case, the some changes of performance occurs through the interval of measurement, as opposed to the first example of detection of no degradation. In some parts of the interval it is degradation, in others optimization. In the final result is very important, that no changes occur in the profile as a whole. This confirms the value of the Sum of the absolute errors, which is close to zero (roundly 6.54e-11).

In summary, detection of no changes in regression is the easier case, as confirmed by our experiments. We can identify this by identity of both components of the regression or by using the simple method, the sum of the absolute error.

6.2 Constant error

In a scientific experiment, the constant error causes measurements to deviate consistently from their expected value. Contrary to random errors, which causes measurements to deviate by varying amounts, these errors cause the same amount of deviation only in one direction.

In our experiments, constant error is mainly represented by change of coefficients b_0 of the actual profile against the baseline profile. With the exception of `exponential` and `power` models, it has been observed in the most of the cases, that change of coefficient b_0, approx increased by the relevant constant with minimal change of coefficients b_1, in all the remaining models. In the Appendix A.4, you can see a change between both profiles, where the best model is a linear and in the Appendix A.5 the best model is an exponential.

Another suitable indicator, in the case of the constant error, is a **mean squared error**. The constant degradation is assumed at every points from the resources of the baseline. With this assumption, it can be expected, that the value of the Mean squared error, will be approximately equal to the squared value of the relevant constant. Table 6.1, shows the experimental verification this assumption. In the table, we use the square root of the mean squared error, instead of the squared value of the constant.

Type of model	**Linear**	**Quadratic**	**Logarithmic**	**Power**	**Exponential**
$\sqrt{\textbf{MSE}}$	9999.98	10003.37	10009.78	10026.27	10055.68
	4986.72	4990.37	4997.93	5495.47	4699.01

Table 6.1: The value of the MSE, when detecting the constant error. In the first attempt of the detection, it was the constant `z = 10000` and in the second attempt, `z = 5000`. This means, that the value of dependent variables from the baseline, was changed by a formula: $f(y) = f_b(y) + z$.

By observing these values, it is possible to detect a constant error. However, once again the question arises, what threshold is appropriate for this detection. With the same problem, we should be observe a value of the coefficients b_0 of models in the actual profile, where an error has occurred. The value of these coefficients, of the most models, approximates to the value of the constant, too. This property of the coefficients of the profile, can be seen in the Appendices A.6 and A.7, where are the results of the analysis by the using **absolute error**.

Another appropriate method is the **relative error**. It can be used to determine the error rate. With increases of the value of independent variables, the rate of deterioration decreases. It is obvious, that constant error at first value will be several times higher, as to the value 100 times more than first. This is confirmed by our experiments that can be seen in the Appendices A.8 and A.9.

The results of the analysis, using the **studentized residuals**, are also remarkable. As already mentioned, this method is suitable for identifying the outliers. Because it is a constant error, where error at every points is approximately the same, outliers should not be here. Our experiments show, that this assumption is well usable to deduce the following properties. The value of the **mean squared error**, calculated from the individual residuals, is almost equal to one. Again it applies, that coefficients b_0 of the most models from the actual profile, approximate this time to the value one. The results of this method, can be seen in the Appendices A.10, A.11 and in the table 6.2.

Type of model	**Linear**	**Quadratic**	**Logarithmic**	**Power**	**Exponential**
MSE	1.00145	1.00193	1.00248	1.00231	1.00181
$\mathbf{b_0}$	0.99296	0.97196	1.18174	1.20521	0.99296

Table 6.2: The value of the Mean squared error, which is close to the one. In the third row, is shown the coefficients b_0 from the selected profile of this analysis.

We show, that the detection of the constant error is possible, by using various statistical methods. Under favorable circumstances, it is possible to determine the type of this error and even the rate of this constant error. However, as it has been outlined, the problem arises with determining the threshold, for error detection.

6.3 Linear error

One possible definiton, of a linear error is by the idea of a classic linear function. Linear function can be expressed by the form $f(x) = b_1 * x + b_0$, where b_0 and b_1 are constants.

From our perception, when linear error has occurred, the error will increase with an increasing value of indepedent variables of the resources. Contrary to the constant error, where the relative error gradually decrease. In our experiments, this kind of error will be represented by the change of coefficients b_0 and b_1 of the target profile against the baseline profile. However, this assumption was observed only in the linear models of these profiles. As we can see in the Appendices A.12 and A.13, under ideal conditions in our experiments, the linear model was changed just in its coefficients. We followed, that the value of both coefficients of the linear model, has the same behavior in all cases, in which, the best fit models has changed.

		Linear	**Quadratic**	**Logarithmic**	**Power**	**Exponential**
$\mathbf{b_0}$	baseline	238.5889	-7092.9499	7942.4242	2308.9694	1838.5053
	target	248.0437	-7081.8701	7953.0452	2318.5934	1848.1188
$\mathbf{b_1}$	baseline	0.0491	0.0656	0.0336	0.0428	0.0459
	target	0.0590	0.0756	0.0436	0.0529	0.0558

Table 6.3: The value of both coefficients of linear model from the baseline and target profile. We can see, that in all types of models, values are changed by the same constant.

The coefficient b_0, expresses the offset on the axis (*y-intercept*), where a dependent variable is represented. Its change against the baseline profile is affected by the error in the first point of the resources of the baseline profile. The coefficient b_1, which expresses the slope of the line, is affected by the error in the every point of the resource of the baseline profile. We confirmed this by the experiment, listed in the Table 6.3, where the initial error has a value of 10 and every other error has increased by the same value.

Of other specified statistical methods, the only applicable is, the **relative error**. The relative error can express, how big the error has occurred in every point of the resources the baseline profile. The error rate will be increased, with increasing of the value of independent variables, from the beginning quickly until it is almost the same, in the end. The error rate is steady after some sections as can be seen in the Appendices A.14. However, this claim is valid only in the profiles, where the best fit models is linear model.

Detection of the **linear** error is not possible by most of the methods. The results of these methods are not as obvious, as in the case of the constant error. Under favorable circumstances, it is possible to determine the type of this error, by checking the coefficients of the linear model from the baseline and target profile.

6.4 Quadratic error

The quadratic model is represented by the form $f(x) = b_1 * x^2 + b_0$, where b_0 and b_1 are constants. Quadratic error represents the faster increase of the value of the error, compared to linear or constant errors. The example of this error, can be seen in the Appendix A.15.

Detection of this kind of error is also difficult. Its properties have different behaviors for different types of models. Whereas the error in the previous two types has traceable properties, for this error it does not apply. One thing, which we can expect with certainty is that error will be have fast growth, at least on the some parts of the measurement interval. Neither at one model has not confirmed, that this growth lasts for the entire the interval. The example of the results of the absolute error method, can be seen in the Appendix A.16.

In the first experiments, we monitored change of the coefficient b_1 of the linear models. At first it seemed, that its behavior depends on the size of the error in each point of the resources. However, various cases of this error were later tested and has proven, that this assumption is not correct. Perhaps the most appropriate aspect in an effort of detection quadratic error is the **standard deviation**, which is a measure of the dispersion of a set of data from its mean. The indivindual errors are unevenly distributed, which means, that the value of the standard deviation has the potential to be high. Contrary to linear errors, where the errors have the potential be spread approximately the same, it is assumed, that the standard deviation will be the highest just for the quadratic error.

	Kind of error	**Linear**	**Quadratic**	**Logarithmic**	**Power**	**Exponential**
σ	**Constant**	2892.66	2799.13	2395.15	3803.29	3983.17
	Linear	160.472	160.520	160.506	1169.87	2011.64
	Quadratic	12572.9	12088.4	13445.9	12578.8	12354.3

Table 6.4: Comparing the values of the standard deviation in the different kinds of errors. The assumption, that value will be the highest for the quadratic error, was demonstrated by experiments.

The detection of this error does not have clear prerequisites for the success. The results of the statistical methods using for the detection other errors is not applicable for quadratic error. However, as we have shown, under certain favorable conditions and with the exclusion of other errors, is possible to detect the quadratic error using the same methods. This error will still be the subject of further studies to detection of degradation.

Chapter 7

Conclusion

This work describe methods for detection of the performance degradation between two profiles. We have demonstrated individual types of errors injected on selected types of models and their detection, through our experiments. Some of these errors are detectable using the selected statistical methods. However, some errors are not easily detectable or not yet explored and will be subject to further studies.

The next step will be the formalization of proper algorithm for automatic detection of performance degradation (or other changes) based on explored statistical methods. The implmented algorithm will be then integrated in the Perun tool workflow and will serve to detect degradations in real world code.

Bibliography

[1] Devore, J.: *Probability and Statistics for Engineering and the Sciences*. Cengage Learning. 2011. ISBN 9780538733526.

[2] O'Halloran, S.: Model Checking. online.
Retrieved from: http://www.columbia.edu/~so33/SusDev/Lecture_5.pdf

[3] Pavela, J.: *Knihovna pro profilování datových struktur programů C/C++*. Bachelor thesis. Brno University of Technology, Faculty of Informarion Technology. 2017.

[4] Puntambekar, A.: *Advanced Data Structures and Algorithms*. Technical Publications. 2008. ISBN 9788184314786.

[5] Weisstein, E. W.: Absolute Error. online. from MathWorld–A Wolfram Web Resource.
Retrieved from: http://mathworld.wolfram.com/AbsoluteError.html

[6] Wenning, C. J.: Absolute and Relative Error. online. October 2014. student Laboratory Handbook.
Retrieved from:
http://www2.phy.ilstu.edu/~wenning/slh/Absolute%20Relative%20Error.pdf

Appendix A

Profiles with models

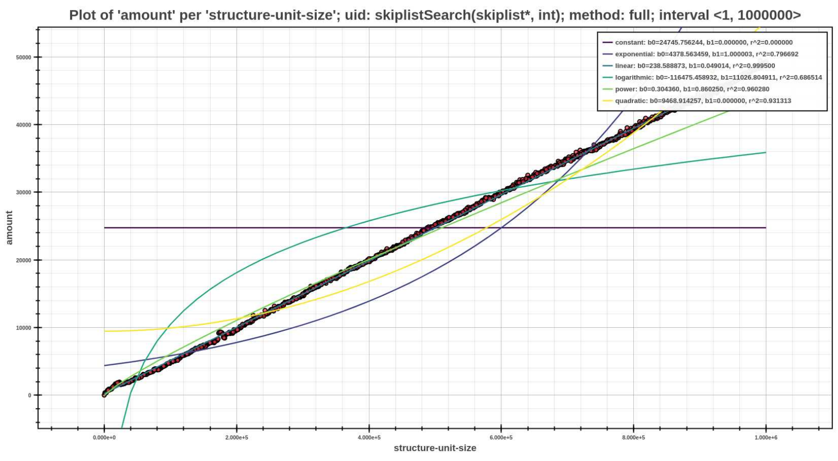


Figure A.1: Data points of profile with its relevant models, corresponding to baseline profile for all expreriments.

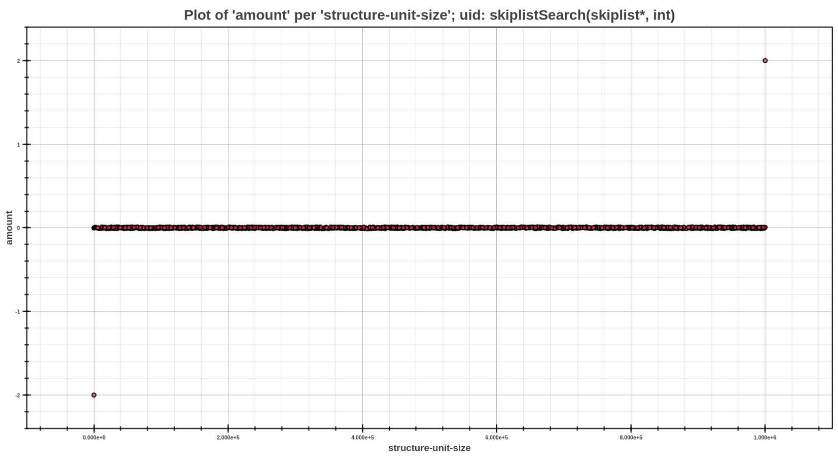


Figure A.2: Difference between models of the two identical profiles. Mean squared error is close to zero, according to the expectation.

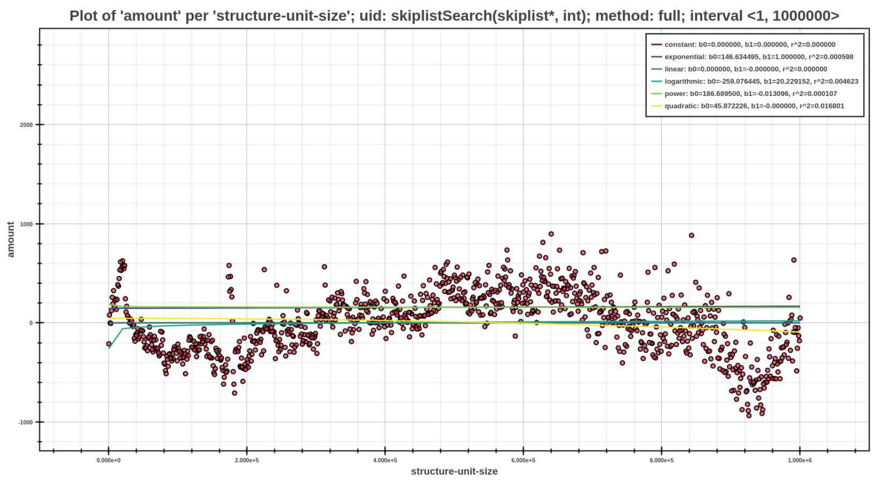


Figure A.3: Example of test, where degradation has not occured, but the change is not close to zero, which means, that profiles are not identical.

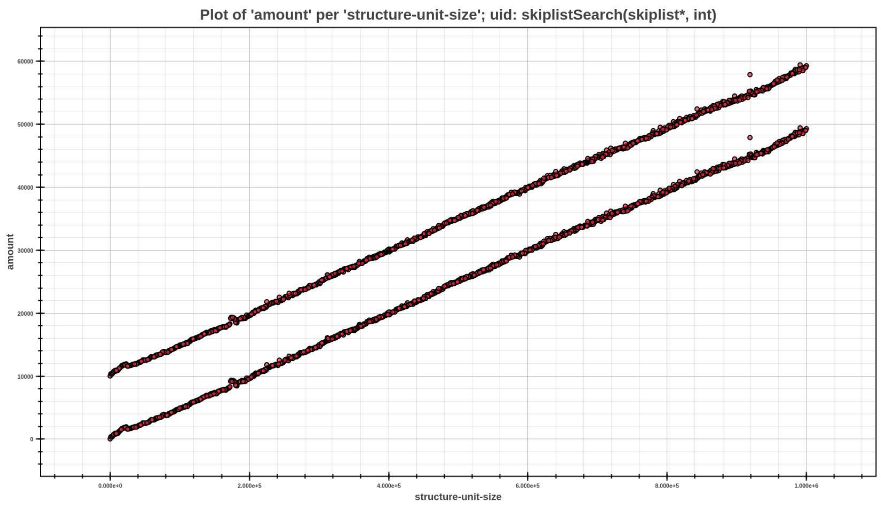


Figure A.4: Resources of the two profiles, where the best models, are linear. The points which are below, are resources from the baselines. The points above them, are resources from the target profile, which contains injected constant error.

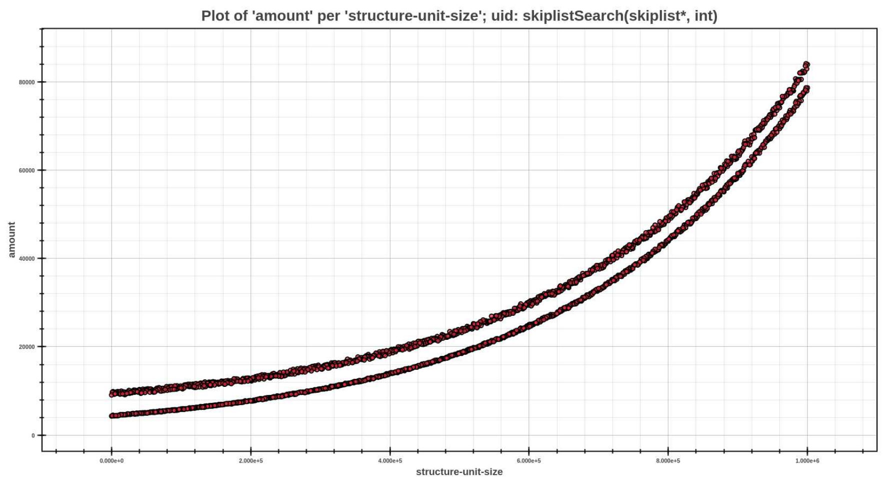


Figure A.5: Resources of the two profiles, where the best models, are exponential. The points which are below, are resources from the baseline model. The points above them, are resources from the target profile, with injected constant error.

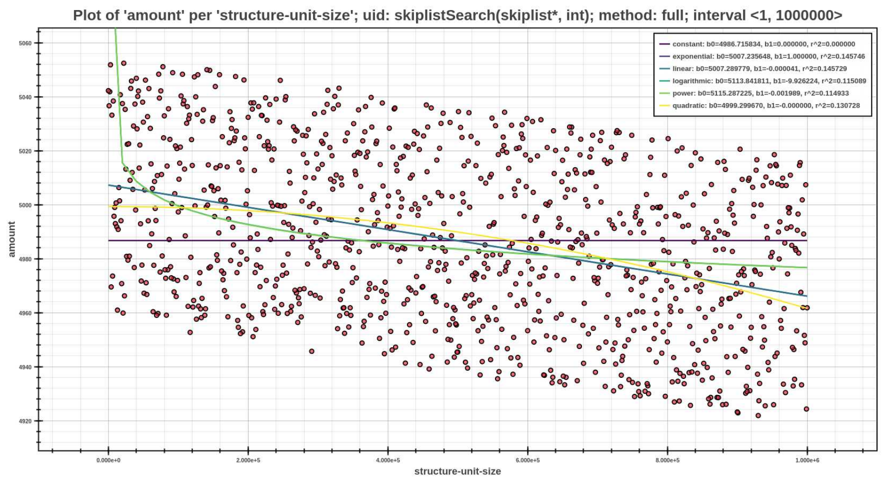


Figure A.6: The results of the experimental detection of the constant error for linear models. The analysis was performed using the absolute error method. It can be seen, that the values are in the range close to the constant. We can notice the values of the coefficients b_o on all the models.

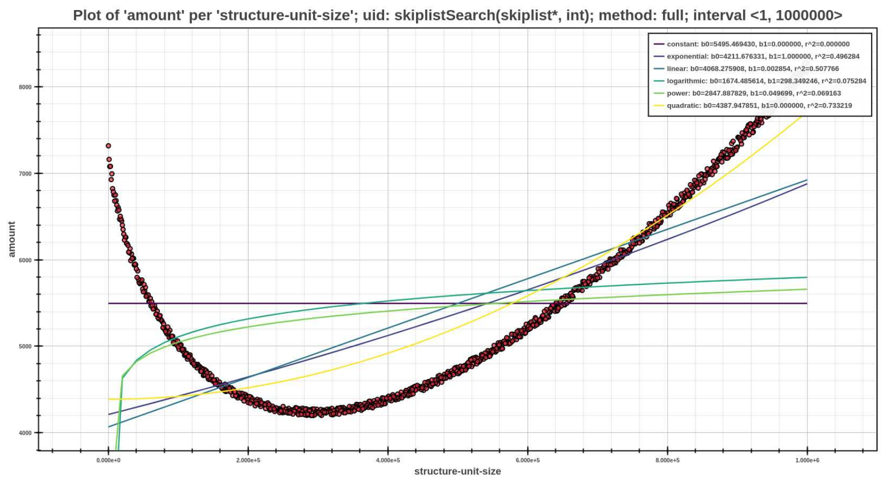


Figure A.7: The results of the experimental detection of the constant error in the power model. The analysis was performed using the absolute error method. It can be seen, that the values are in the range close to the constant. We can notice the values of the coefficients b_0 on all the models.

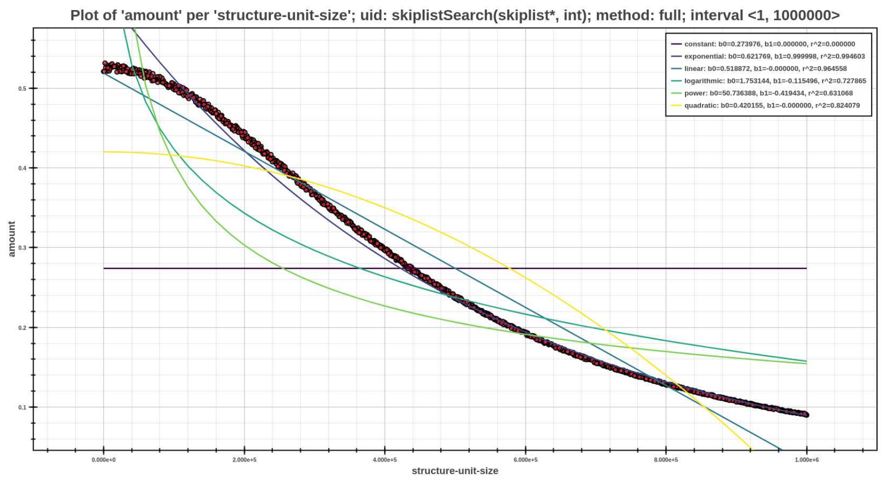


Figure A.8: The results of the experimental detection of the constant error in the quadratic model. The analysis was performed using the relative error method. We can see decreasing rate of deterioration.

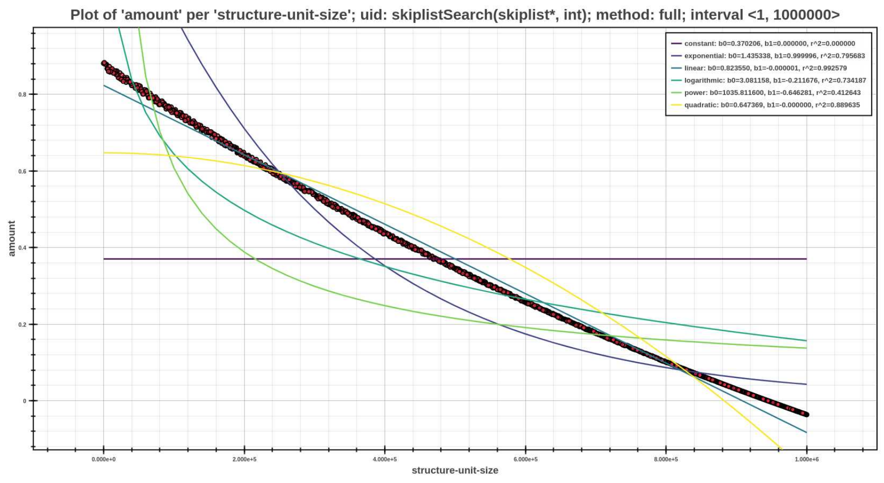


Figure A.9: The results of the experimental detection of the constant error in the exponential model. The analysis was performed using the relative error method. We can see decreasing rate of degradation.

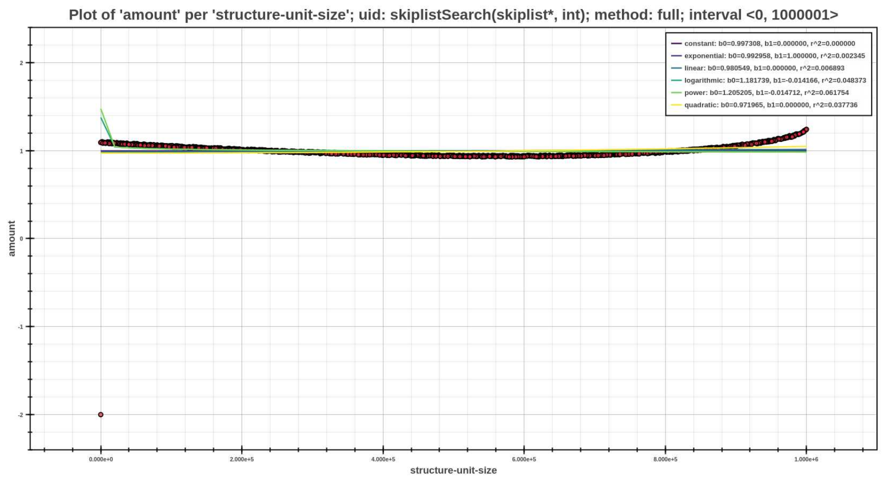


Figure A.10: The result of the experimental detection of the constant error in the power model. The analysis was perfomed using the studentized residuals method. We can see, that the values approximate the value of one, which quarantees, that the value of MSE will be almost equal to one. We can notice, that the values of coefficient b_0 in the all models, have approximately value of one.

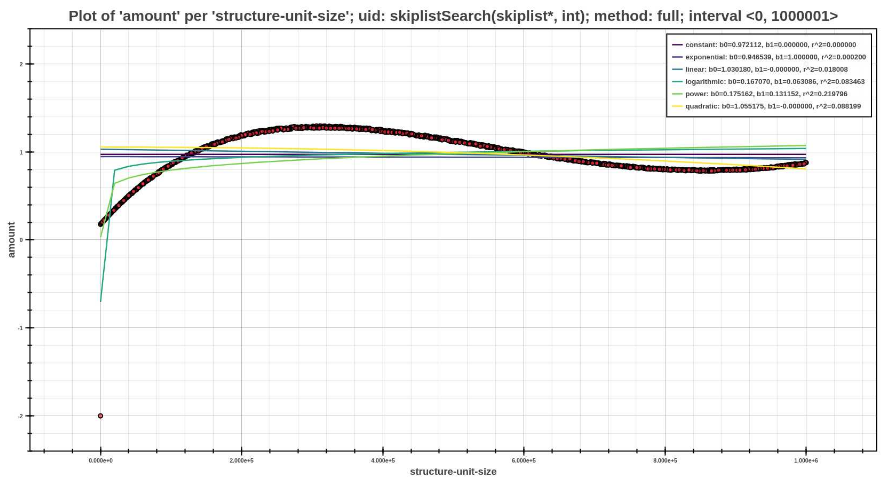


Figure A.11: The result of the experimental of detection of the constant error in the exponential model. The analysis was perfomed using the studentized residuals method. We can see, that the values approximate the value of one, which quarantees, that the value of MSE will be almost equal to one. Too, we can notice, that the values of coefficient b_0 in the all models, have approximately value of one.

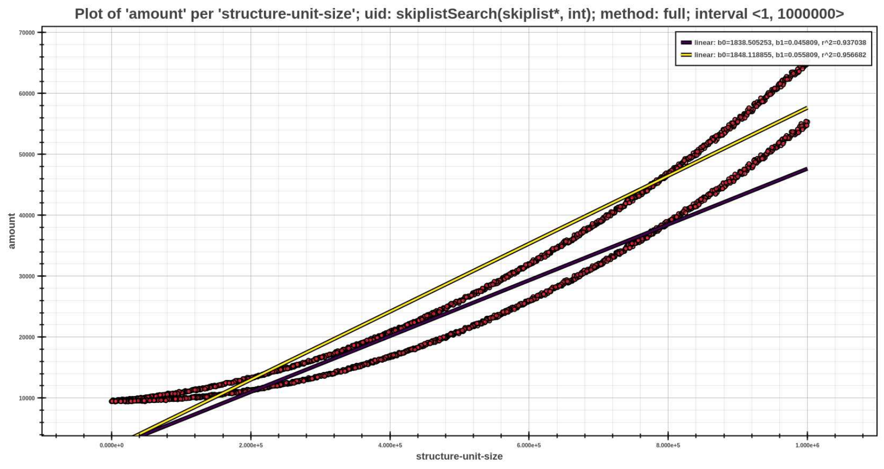


Figure A.12: The linear error of profile, where the best fit models of the baseline profile is quadratic model. Both linear models, of the baseline and target profile, are shown. Their coefficients are changed in dependence of the linear error, in particular its initial value and its rate.

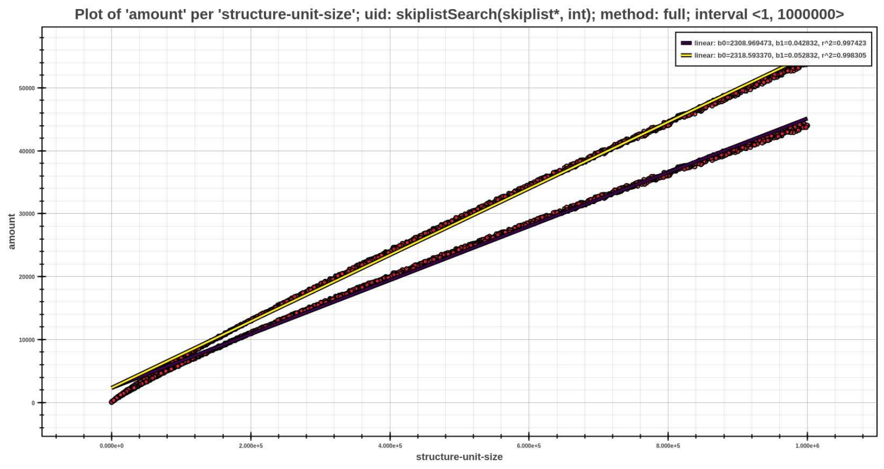


Figure A.13: The linear error of profile, where the best fit models of the baseline profile is power model. Both linear models, of the baseline and target profile, are shown. Their coefficients are changed in dependence of the linear error, in particular its initial value and its rate.

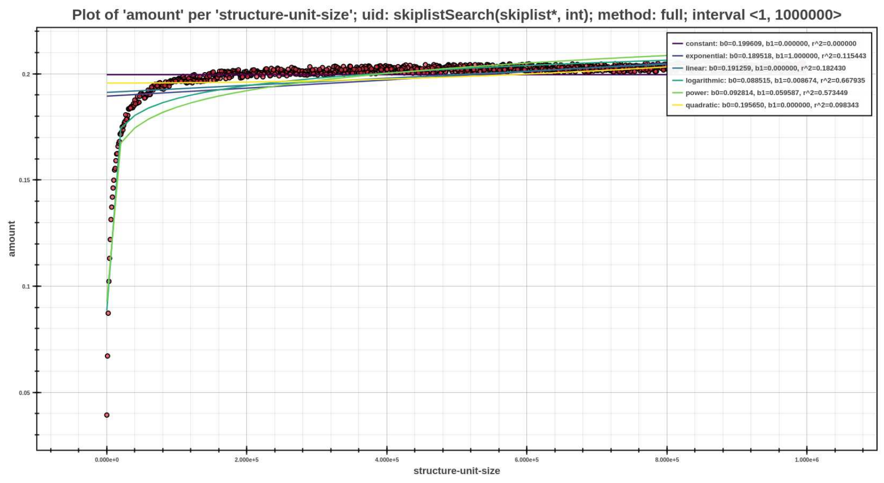


Figure A.14: The result of the experimental detection of the linear error in the linear model. The analysis was performed using the relative error method. We can see, that the error rate increases, with the increase of the value of independent variables.

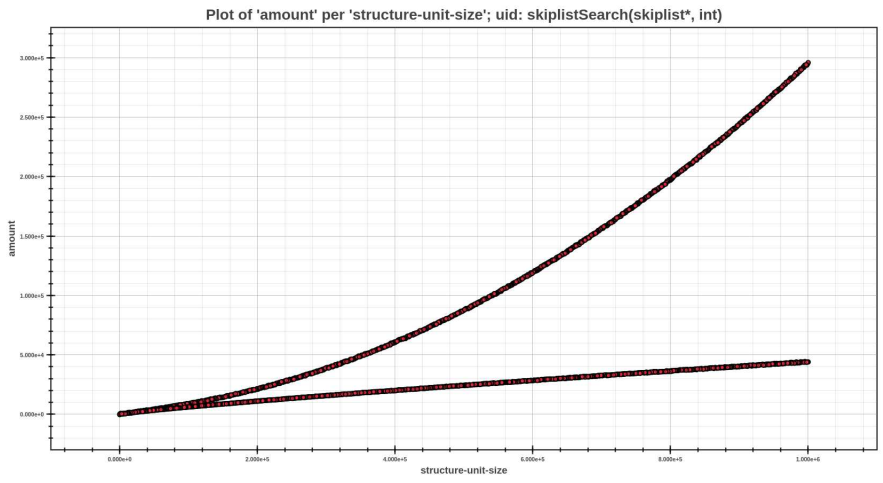


Figure A.15: Resources of the two profiles, where the best models, are power. The points which are below, are resources from the baselines. The points above them, are resources from the target profile, which contains injected quadratic error.

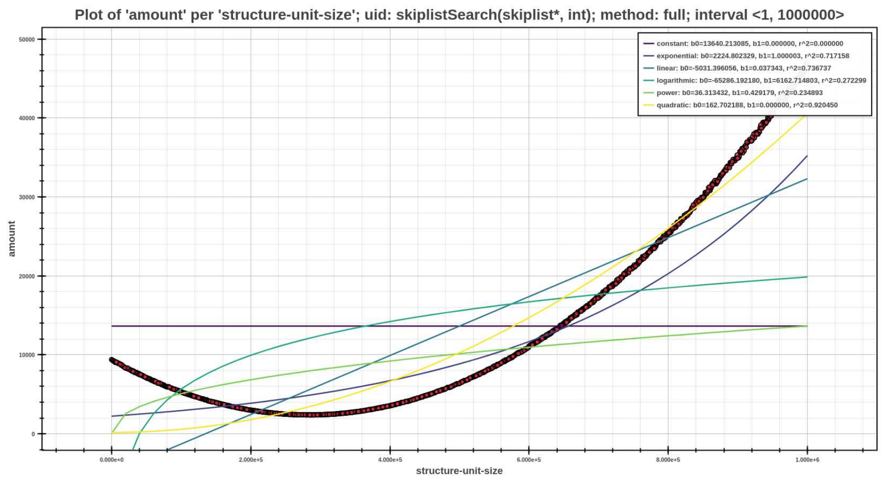


Figure A.16: The result of the experimental of the quadratic error in the linear model. The analysis was performed using the absolute error method. We can see instability of growth of the error on the measurement interval.