Institute of Software Technology

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor‘s Thesis Nr. 331

Automated Categorization of Performance Problem Diagnosis Results

Tobias Angerstein

Course of Study:	Softwaretechnik
Examiner:	Dr. André van Hoorn
Supervisor:	Dr.-Ing. Christoph Heger, Novatec Consulting GmbH Dr.-Ing. André van Hoorn, University of Stuttgart Dipl.-Inf. Thomas Kluge, Novatec Consulting GmbH Dr.-Ing. Dušan Okanović, University of Stuttgart Dr.-Ing. Alexander Wert, Novatec Consulting GmbH
Commenced:	2016/04/26
Completed:	2016/10/26
CR-Classification:	D.2.8

Abstract

In times of big data and global networking, software performance becomes a major issue in software development. Many enterprise applications were not designed for the current highly frequented usage. Thus, more and more organizations are using Application Performance Monitoring tools, to detect bottlenecks and other performance problems in their product. Meanwhile, there are many different tools on the market which provide a detailed performance-aware analysis of enterprise applications. However the common Application Performance Management tools do not provide any additional automated performance problem diagnosis. The performance expert has to find the cause on its own. *diagnoseIT* [4], a framework for automatic diagnosis of performance problems in enterprise applications addresses this problem. *diagnoseIT* extracts performance problem instances from execution traces. The resulting set of problem instances can become huge and the analysis of the problems is very time-consuming.

We extend the general concept of *diagnoseIT* and categorize the resulting set of problem instances into a manageable number of problem categories. Therefore, a concept of categorization is elaborated. We analyze several different categorization approaches and evaluate the performance and the quality of the result. We perform a sensitivity analysis, which analyzes the influence of each attribute of a problem instance on a clustering result. The results of the sensitivity analysis indicates, that there is a potential for optimization. Thus, we introduce a concept of optimization, which optimizes the process of categorization by weighting the attributes and we compare different manual and automatic optimization approaches with regard to the improvement compared to default weights.

In the evaluation, we examine the accuracy and the performance of the approaches. The evaluation shows that k-means clustering provides the most promising and best results. Additionally, the evaluation indicates a high potential for optimization. However, the results of the evaluation show that it is difficult to optimize the weights without any knowledge about the analyzed system.

Zusammenfassung

In Zeiten von Big Data und globaler Vernetzung gewinnt die Performanz von Software Applikationen immer mehr an Bedeutung. Viele Anwendungen wurden ursprünglich nicht für die aktuelle hochfrequentierte Nutzung konzipiert. Deshalb kommen in immer mehr Entwicklungsteams Performance Application Management Werkzeuge zum Einsatz, um etwaigige Performanzengpässe zu finden und zu beseitigen. In der Zwischenzeit gibt es eine Vielzahl an Werkzeugen, welche eine detailierte Performanzanalyse von Enterprise Applications unterstützen. Allerdings unterstützen die gängigen Werkzeuge keine zusätzliche automatische Diagnose von Performanzproblemen; die Ursache der entdeckten Probleme müssen vom Performanzanalysten manuell gesucht werden. *diagnoseIT* [4], ein Framework für die automatische Diagnose von Performanzproblemen zielt auf diese Problematik ab. *diagnoseIT* analysiert Ausführungssequenzen nach etwaigigen Problemen und liefert eine Menge von Probleminstanzen zurück. Allerdings kann diese Menge schnell sehr groß werden und es wird sehr aufwändig die Probleminstanzen zu analysieren.

In dieser Bachelorarbeit erweitern wir das Konzept von *diagnoseIT* um eine Kategorisierung der Performance Problem Instanzen. Die Probleminstanzen werden in eine übersichtliche Anzahl von Problemkategorien einsortiert. Wir analysieren eine Vielzahl an Möglichkeiten der Kategorisierung und evaluieren die Performanz und die Qualität des Resultats. Zusätzlich führen wir eine Sensitivitätsanalyse durch, welche den Einfluss von jedem Attribut einer Probleminstanz untersucht. Die Ergebnisse der Sensitivitätsanalyse zeigen, dass das Kategorisierungskonzept Potential zur Optimierung hat. Deshalb stellen wir ein Optimierungskonzept vor, welches den Kategorisierungsprozess mit Hilfe der Gewichtung der einzelnen Attribute optimiert. Des Weiteren vergleichen wir unterschiedliche manuelle und automatisierte Optimierungsansätze hinsichtlich ihrer Verbesserung im Vergleich zur Standardgewichtung.

In der Evaluation untersuchen wir die Präzision und die Performance der vorgestellten Ansätze. Die Ergebnisse der Evaluation zeigen, dass das Clustering mit k-means das vielversprechendste und beste Ergebnis liefert. Zudem legen die Ergebnisse der Evaluation nahe, dass ein enormes Optimierungspotential besteht. Allerdings zeigt die Evaluation, dass die Optimierung der Gewichte nur einen Mehrwert bietet, wenn im Prozess der Optimierung Wissen über das Systems miteinfließt.

Contents

Chapter 1

Introduction

In the following we introduce the topic and present the goals of the thesis.

1.1 Motivation

Today, many Application Performance Management solutions support monitoring and early detection of performance problems. Many of the tools only support alerting and visualization of performance-relevant measures. Some tools also provide a detection of problem instances (Section 2.1).

In this thesis, we'll deal with the resulting problem instances of diagnoseIT [4]. Under real operating conditions, diagnoseIT provides a huge number of instances. However, many of the performance problem instances are quite similar and can be categorized in only few root causes. Currently, the process of analysis has to be done manually. In order to improve this situation, we'll elaborate a concept of automated categorization. A clustering approach is used to categorize the performance problem instances into a manageable set of clusters.

For each use case the approach of categorization has to be adapted and modified. An optimization algorithm will automatically tune the parameters of the categorization approach. Figure 1.1 shows the basic concept of automated categorization. diagnoseIT provides the performance problem instances. Based on the attributes of each problem instance, the instances are split into groups. Each group represents a problem category. The parametrization of categorization can by optimized by an optimization approach. Alternatively, the default parameters can be used.

This research will be done in the context of the collaborative research project diagnoseIT.

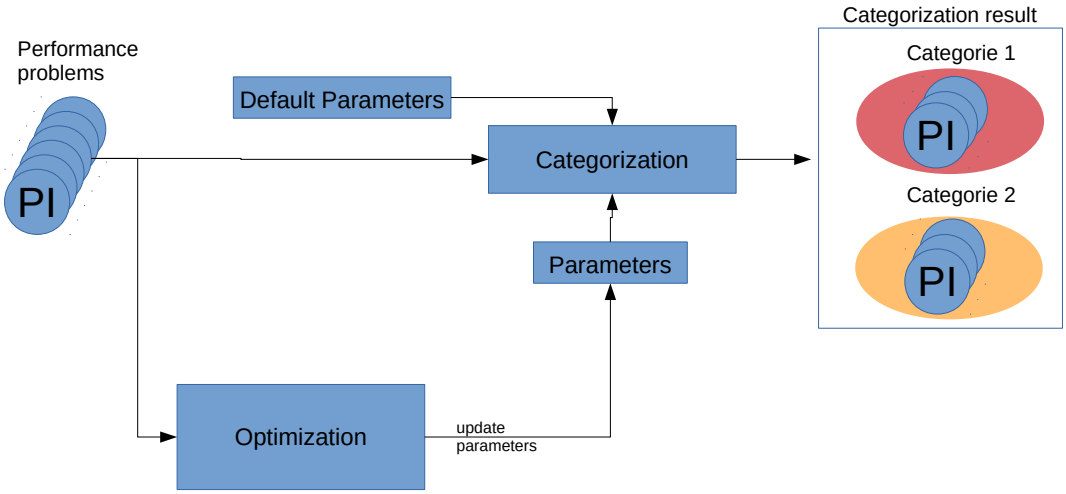


Figure 1.1. Basic concept of categorization and optimization

1.2 Goals

The goal of this thesis is to develop a concept of categorization of problem instances based on a common categorization approach. Problem instances detected by diagnoseIT with a great agreement should be condensed into one problem category (Section 2.1).

- We develop a concept of categorization of problem instances extracted from execution traces
- Based on that, we develop a concept of automated optimization.
- We'll integrate a proof-of-concept implementation into the diagnoseIT-inspectIT prototype

Chapter 2

Foundations and Technologies

In this chapter we introduce the used algorithms and technologies and define important terms, which are related to the work of the thesis.

2.1 Terms

Hereafter, important terms all around performance and categorization will be defined.

Definition 1

Performance. *"Performance is the degree to which a software system or component meets its objectives for timeliness." [22]*

Definition 2

Performance problem. *A performance problem is a use-case, which compromises the performance of an application.*

Definition 3

Before elaborating a process of categorization of problem instances, it is necessary to have a research on a general definition of a problem instance. In the following, based on the definition of Renegard [19], we define a performance problem instance.

Problem instance. *A performance problem instance describes a concrete atomic performance problem. In the context of diagnoseIT, the problem instances are extracted from an execution trace. It will be defined by several attributes, which characterize the problem. A performance problem instance can be described as vector* $\overrightarrow{V}$*. Each element* f *represents one attribute of the problem;* $f \in A \mid A$ *is a set of all attributes. The attributes represent known causes and influences of performance problems. The number of attributes must be supported by diagnoseIT.*

Examples for attributes are:

- *exclusive response time: How much time a specific method has spent for execution?*
- *method signature: Which method is to slow?*
- *business transaction: Which use case is affected?*

- *Remote calls: Which other components are affected?*

Definition 4

Problem category. *A problem category includes all problem instances, which have similar attributes and reference to the same root cause or problem*

Definition 5

Exclusive Response Time. *Exclusive Response Time describes the time, until a method has finished. The execution time of invoked methods is excluded.*

Definition 6

Reference cluster. *A reference cluster is a set of performance problem instances, which are categorized by hand. It can be used to compare it with other clustering results and determine their quality (Section 4.1)*

Definition 7

Fitness function. *A fitness function determines the fitness of a generated solution and is used by generic optimization algorithms. In our case, the fitness function uses internal and external criteria to determine the fitness of a clustering result. (Section 4.1).*

2.2 Clustering

In this section, we introduce the fundamental concept of clustering and compare different clustering algorithms.

Clustering Clustering in general is a concept for categorization. The algorithm processes on a set of input patterns. Each pattern consists of n features; one pattern P can be represented as vector $\overrightarrow{P} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$

In terms of this thesis, a pattern represents a performance problem instance and a feature represents an attribute. The pattern can optionally be weighted:

$$\overrightarrow{P_{weighted}} = \begin{pmatrix} f_1 * weight_1 \\ f_2 * weight_2 \\ \vdots \\ f_n * weight_n \end{pmatrix}$$

Based on the features of each pattern the algorithm evaluates all combinations of distances between the patterns. Patterns with a small distance are assigned to the same cluster. Usually the clustering algorithm operates, until the break condition is reached.

Example Figure 2.1 illustrates, how the process of clustering basically works: The cluster algorithm gets a set of input patterns and calculates the distance (Section 2.2.1) between each pattern and groups all patterns into clusters. Whereas feature2 of pattern *1,2,3* is between 100 and 240, feature2 of pattern *4,5,6* is between 240 and 280. Thus, the algorithm divides the patterns into two separate clusters.

Figure 2.1. Clustering example

2.2.1 Distance metrics

The behavior of a clustering algorithm is influenced by the used distance metric. There are many different distance metrics. We concentrate on these [13]:

Table 2.1. Comparison of common distance metrics

Metric	Functioning
Euclidean distance	The Euclidean is one of the most intuitive and common used distance metrics. It is mainly used to calculate a distance between two n-dimensional points. Formula for n-dimensional space: $dist(P_i, P_j) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + .. + (p_d - q_d)^2}$ **Advantages:** Good performance. **Disadvantages:** Very sensitive to distortion
Mahalanobis distance	The Mahalanobis distance tries to eliminate this disadvantage. The features are normed by the sample covariance matrix of the patterns. $dist(P_i, P_j) = \sqrt{(p-q)^T S^{-1}(p-q)}$ **Advantages:** Elimination of distortion **Disadvantages:** High computation time
Mutual Neighbor distance	The mutual neighbor distance only focuses on the neighbors of each pattern $NN(p,q)$ = number of all neighbors z_i of q where the distance $d(z_i, q)$ is equal or smaller than the distance $d(p,q)$ $MND(p,q) = NN(p,q) + NN(q,p)$ **Advantages:** The environment of two patterns is included **Disadvantage:** Much computation time

In the further steps, we use the euclidean distance, because its much faster than the other approaches and from a technical point of view, the euclidean distance is supported by the Weka framework.

2.3 Clustering frameworks

In this section, we define the requirements on a cluster framework and introduce different cluster frameworks.

2.3.1 Requirements on a clustering framework

The clustering framework should be importable as Java library, because the code of the implementation is written in Java. Several different clustering algorithms have to be

supported. We discuss Weka and the Rapidminer framework:

2.3.2 Weka framework

The Weka framework [9] is an open-source project of the Machine Learning Group at the University of Waikato. It provides users the usage of different categorization algorithms and configurations. The Weka framework provides three different types of interaction.

- Graphical user interface: The user can configure the categorization algorithms manually. Features can be imported and the results are visualized.
- Command line: All functionalities of the graphical user interface are provided in the command line mode.
- Java library: The Weka framework can also be imported as Java library: In this thesis, mainly this approach will be used.

Each clustering algorithm can be configured and can be combined with any implemented distance metric. For the evaluation of the algorithm it is only necessary to change the configuration parameter of the algorithm.

The Weka framework allows to set a weight for each attribute. However, the *EuclideanDistance* class is not using this information. Thus, we manipulated the Weka framework: The class *NormalizableDistance* now uses the weight of the given instances. The manipulation has no influence on the rest of the framework.

2.3.3 Rapidminer

Rapidminer is a platform, which can provide support for different kinds of data analysis. Rapidminer implements many different data mine and machine learning algorithms. It also provides hierarchical clustering and can be imported as Java library. Rapidminer comes with a GUI which provides to create several data models, and also test and validate the data models [3].

2.3.4 Comparison of the cluster frameworks

Whereas Weka can only be used for classification and clustering, Rapidminer supports many other features and algorithms, which we do not need in the context of our work. Thus, we decided to use Weka, because Weka is more light-weigh and improved for clustering. Moreover, the Java library of the Weka framework has a much better and understandable structure. This is also reflected by the documentation of the two tools. Unfortunately, the Rapidminer documentation mostly focuses on the usage of the GUI and not on the usage of the Java library, because it is intended to be used as standalone analysis tool.

2.4 Optimization frameworks

This section introduces two optimization frameworks and defines the requirements on such frameworks.

2.4.1 Requirements on the optimization framework

The framework should be able to provide the evolutionary optimization algorithm and the hill-climbing algorithm. Furthermore it should provide an importable Java library. We had a research on the following frameworks:

2.4.2 Opt4J Framework

Opt4J [14] is an open-source optimization framework written in Java which supports different kinds of evolutionary algorithms. Opt4J provides a graphical user interface. The GUI shows the different generations and visualizes the evolutionary optimization with several diagrams. It is part of the Java library and can be started from the Java code. For using the framework, we had to extend the optimization problem with the following classes:

- *Genotype creator*: This class generates a new randomized genotype. In our case, it generates DoubleGenotype objects which consists of the random generated weights for the cluster algorithm
- *Problem decoder*: This class converts the genotype into a rateable phenotype: It executes the cluster algorithm with the given weights and return a list of clusters.
- *Problem evaluator*: This class evaluates the given result. In our case, a fitness function computes a value, which represents the fitness of the current result.
- *Problem Module*: This class defines the problem in general and combines the genotype generator, the problem decoder and the problem evaluator.

As a result the evolutionary optimization engine returns a configuration, which provides an optimal clustering result.

2.4.3 JAMES Framework

The James framework [8] provides a Java library for executing a hill-climbing algorithm. To use this framework for our optimization problem, we had to extend the following classes:

- *Solution*: This class provides one clustering result
- *ProblemData*: This class is a data container for some general information about the current optimization problem.
- *ProblemObjective*: This class rates the given solution.

- *Opt2Move*: This class manipulates the given solution
- *OptNeighborhood*: This class returns a new move which defines the next manipulation.

As a result, the framework returns a configuration, which provides an optimal clustering result.

2.4.4 Comparison of the optimization frameworks

Unfortunately, both frameworks do not provide the full scope of algorithms. Thus, for each optimization approach we use James to implement hill-climbing and Opt4J to implement the evolutionary optimization.

2.5 State of the art performance problem analysis and detection

In this section, three existing open-source application performance management tools are introduced: inspectIT (2.5.1), kieker (2.5.2) and diagnoseIT (2.5.3).

2.5.1 inspectIT

inspectIT [5] is an open-source performance analysis tool, which consists of three components (Figure 2.2):

- Java Agent,
- CMR (Central Measurement Repository) and
- Rich Client (User Interface).

The agent is integrated into the runtime environment. The measuring points have to be defined in a configuration file of the agent. The data, measured by the agent, is sent to the CMR. The CMR accumulates data of several agents and provides it to the Rich Client. inspectIT has predefined sensors, for instance *TimerSensor* (monitors the exclusive response time of a method) and *DatabaseSensor* (monitors database calls and SQL statements). inspectIT does not change the source of the monitored software. It uses runtime bytecode injection and leaves the source code as it is. The Rich Client provides a presentation of the collected data, but the user has to analyze the causes of the problems itself.

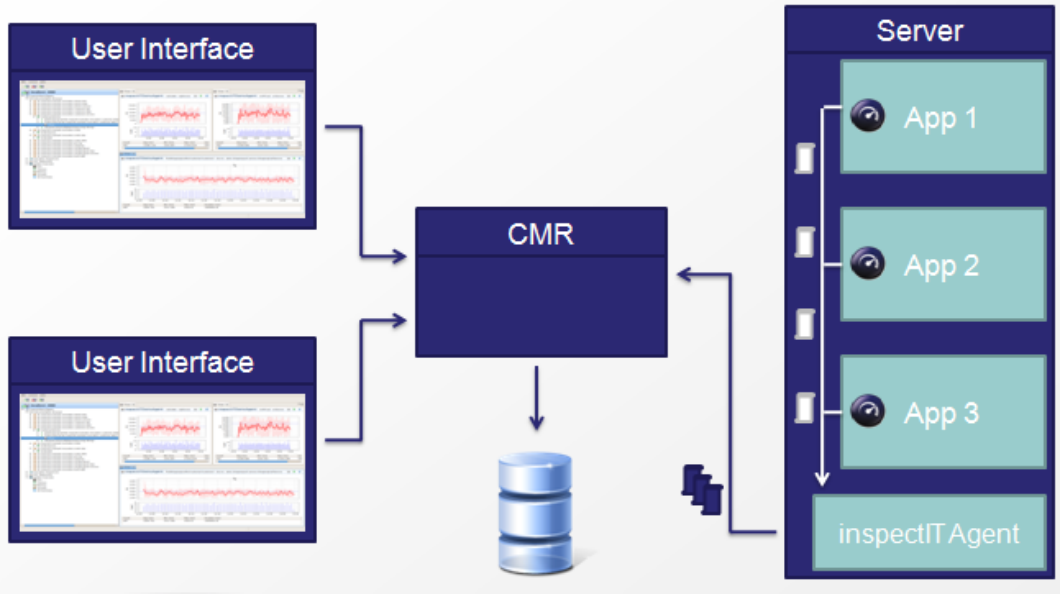


Figure 2.2. inspectIT architecture [6]

2.5.2 Kieker

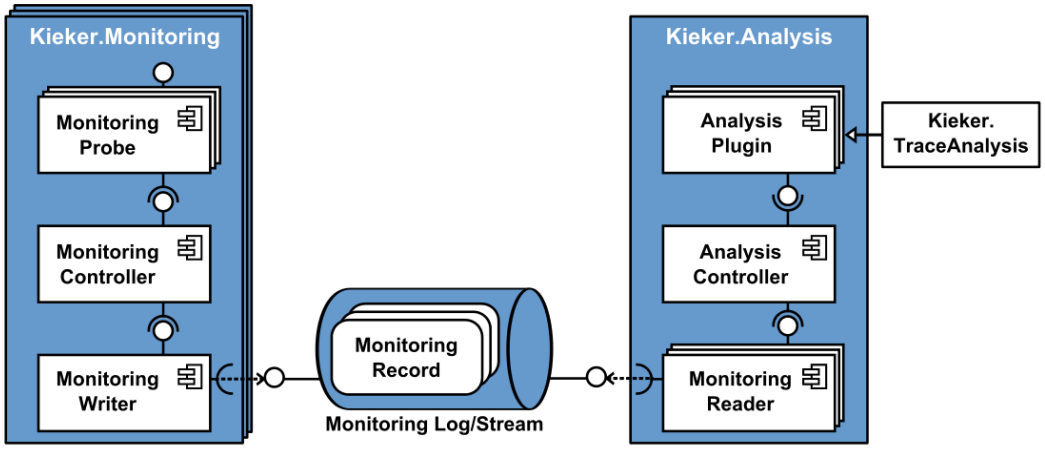


Figure 2.3. Kieker framework architecture [17]

The Kieker Framework [17] provides monitoring of response time, user sessions and execution traces, but also CPU and memory utilization for Java EE systems. Kieker also

uses a Java Agent with run-time bytecode injection. It is also easily extensible.

The tool is divided into two separated parts (Figure 2.3):

- Kieker Monitoring: The Kieker Monitoring component is responsible for the instrumentation and monitoring
- Kieker Analysis: Kieker Analysis is responsible for analyzing and visualizing the measured data. However the analysis component does not provide any problem detection and categorization.

2.5.3 diagnoseIT

diagnoseIT [4] is an analysis framework for APM tools, which detects problems from execution traces. The execution traces can be provided by any APM tool (e.g. Kieker and inspectIT), if it can be transformed in to the Open.xtrace [15] format. Figure 2.4 shows the abstract structure of diagnoseIT:

- Traces provided by a monitoring tool are converted into the OPEN.xtrace format
- Each trace will be analyzed: predefined rules are applied to the trace
- Based on this analysis results, performance problem instances are generated.

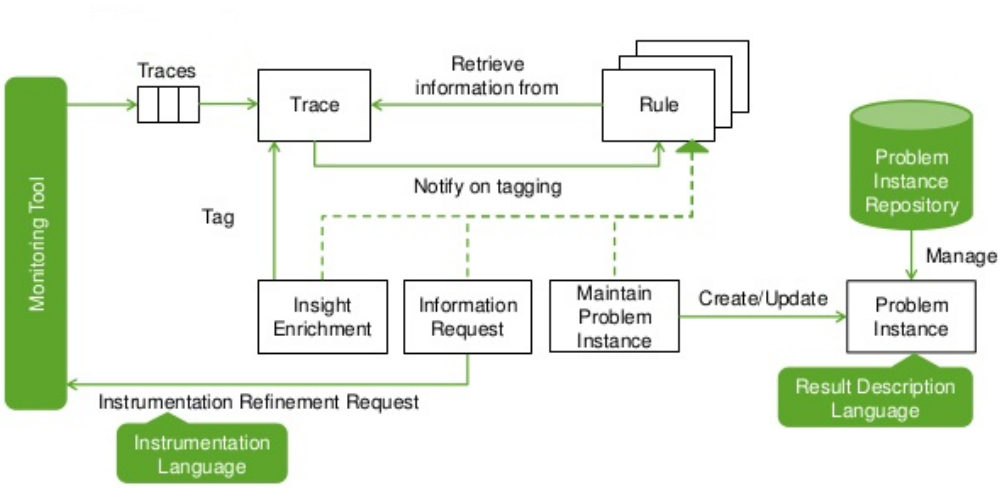


Figure 2.4. Overview DiagnoseIT [12]

Identity based categorization approach

Currently, diagnoseIT can detect performance problem instances. The performance problem instances are represented by the *ProblemInstance* class, which contains the following

attributes.

- business transaction: Which business transaction contains the detected problem?
- entry point: Which method invoked the problematic method?
- problem context: Which method is in affected?
- cause: What caused the problem?
- node type: iterative or parallel execution?
- exclusive time sum: How much time has consumed the affected method?

Currently, diagnoseIT summarizes performance problem instances using their attributes. If all attributes are equal, the problems will be aggregated. If only one attribute differs, the performance problems will not be aggregated, although they are very similar and have the same root cause. For instance, a method is very slow, two problem instances are occurring and only the execution time differs, the current approach would not aggregate the two performance problem instances. Thus, we need an approach for categorizing the detected performance problem instances.

2.6 Procedure

Tasks are divided into four work packages:

WP1 Categorization of problem instances: In the first work package we create a general concept of categorization.

WP2 Optimization of categorization: In this work package, we create a concept of automated optimization.

WP3 Implementation of prototype (Proof-of-concept): In the third work package, based on and the concepts will be implemented

WP4 Evaluation: After the implementation, different approaches of categorization and optimization will be analyzed and compared due to the predefined requirements.

Chapter 3

Approach of Categorization

In the following, we introduce a concept of categorization and compare different categorization approaches.

3.1 Data appropriation

diagnoseIT provides the performance problems as unique problem instance objects. A problem instance (2.1) has several attributes which will be used to aggregate the data. Usually, categorization approaches need the performance problem instances as a vector. As preparation, we convert the problem instance objects into a new vector based data structure, because the categorization approach can only deal with a set of attributes. Each unique vector has a unique identifier to guarantee a unique conversion to the original problem instance.

3.2 Overview of categorization approaches

There are many different state-of-the art categorization approaches, which have advantages and known disadvantages. To find the most fitting approach, it is necessary to define the requirements.

3.2.1 Requirements on the categorization approach

To rate the capability of the categorization approach, criteria have to be defined. The following requirements have to be fulfilled:

- R1: *Uniqueness.* Each problem instance should only be member in one category. For solving a detected problem, a membership must be unique.
- R2: *Independence.* The algorithm should be independent of the number of categories or the order of features.
- R3: *Performance.* The algorithm should be fast and be able to handle a large amount of data.

The compliance of the requirements will be checked during the choice of the algorithm.

There are three different approaches how to categorize data:

- *Classification*: The algorithm categorizes the instances based on training data. Using the trends of the training data, it can decide, in which category each instance belongs to.
- *Clustering*: The algorithms categorizes the instances based on the distances between instances.
- *Decision based*: The algorithm decides based on defined criteria: if two instances have equal attributes, they are grouped into the same category.

The Classification approach does not comply with our requirements (Section 3.2.1), because it needs training data, which is already labeled with the right category. Unfortunately, it is necessary to know the number of expected clusters to create the labeled training data. The decision based approach is also not useful, because it can only make hard decisions. For instance, if all attributes of two instances are equal, except a numeric value, the algorithm will put the two instances in two different clusters. A cluster algorithm is appropriate, because there are existing clustering algorithms, which comply with our requirements.

3.2.2 Clustering algorithms

According to [13], we present different clustering algorithms and check, whether all requirements are fulfilled (Section 3.2.1).

Hierarchical Clustering provides different level of clusters. The algorithm either starts with one single cluster, in which all patterns $\overrightarrow{P_i}$ are members, or each single pattern is defined as single cluster. As preprocessing, the pairwise distances between all patterns have to be computed and stored in a list, ordered ascending. The algorithm is graph based: the algorithm iterates over the list, picks a distances d_k and creates edges between all patterns $\overrightarrow{P_i}, \overrightarrow{P_j}$, where $distance(P_i, P_j) \leqslant d_k$. The break condition of the algorithm depends on the concrete implementation .

Complete Link Clustering is a concrete implementation of hierarchical clustering. It computes, until each pattern is connected to a graph, which connects all patterns. Figure (Section 3.1) shows the initial starting point. The nodes of the graph represent arbitrary patterns. The edges represent the distance between two patterns, which is computed by a defined distance metric and denoted by the number next to the edge.

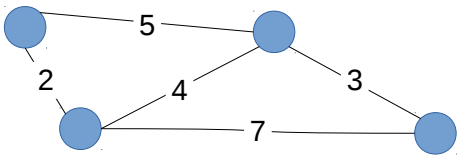
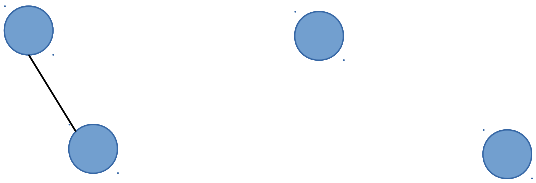


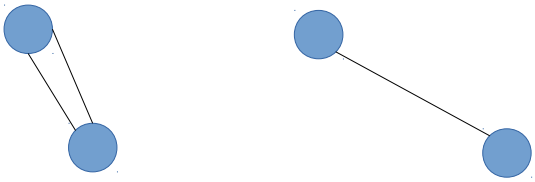
Figure 3.1. Complete-Link Example: Initial patterns

For this example, the algorithm needs four steps (Figure 3.2):

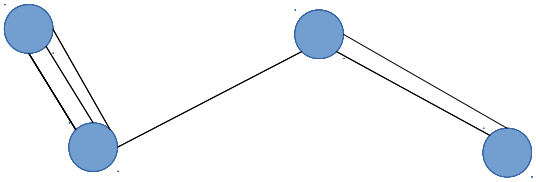
(a) Nodes with distance $\leqslant 2$ get connected



(b) Nodes with distance $\leqslant 3$ get connected



(c) Nodes with distance $\leqslant 4$ get connected



(d) The algorithm cuts the graph into two clusters at the weakest connection

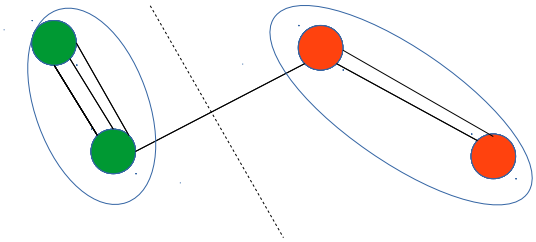


Figure 3.2. Complete-Link Example

Partitional Clustering All partitional algorithms have in common, that a preprocessing is needed, which defines k initial partitions (clusters). Hereby, the wanted number of clusters has to be known before. The algorithm iterates over each pattern and computes the cluster centers of all clusters. In each iteration, the patterns are reassigned to the closest cluster center. Because the clustering results depends on the order of patterns, many implementations executing the algorithm with a different order of patterns. The algorithm stops, as soon as the break condition is reached.

Possible break conditions:

- The algorithm reassigns, until an error-function E is lower than a predefined value.
- The algorithm computes, until the clusters are stable and do not change significantly anymore.

Often, it is very difficult to estimate the number of clusters k. With gap statistic it is also possible to determine the optimal k. The k-means algorithm is executed several times with different assignments of k and on different set of training data. The training data can be a subset of the data, which has to be clustered. From all generated clustering results, this approach picks the k which provides the clustering result with the lowest sum of squared errors.

Fuzzy clustering permits multiple memberships of one pattern in clusters. As a result, this clustering approach returns a membership matrix $P_{(Patterns)} \times C_{(Clusters)}$:

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_i \\ P_n \end{array} \begin{array}{c} \begin{array}{ccccc} C_1 & C_2 & C_3 & C_j & C_k \end{array} \\ \begin{pmatrix} 0.5 & 0.5 & 0.0 & \dots & 0.3 \\ 0.9 & 0.7 & 0.1 & \dots & 0.2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0.75 & 0.8 & 0.7 & \dots & 0.6 \end{pmatrix} \end{array}$$

The values in the matrix represent the force of a cluster-center towards a pattern in the range $0 \leqslant x \leqslant 1$ [13].

Artificial Neuronal Network Clustering can iteratively learn, which classification is the most optimal one. In addition it can handle computation perfectly in parallel. Thereby, the algorithm performs well, it is important to include information about the environment as much as possible.

Evolutionary Clustering Evolutionary Clustering is based on the standard evolutionary process. A random set of clusters will be chosen. New variations based on the "parent-solution" will be built with the standard evolutionary operators (select, mutate and recombine). As a result, the algorithm returns a list of possible solutions with a fitness $\geqslant$ a predefined $min_{fitness}$.

Comparison of different approaches

In the following table 3.1 we compare the presented clustering algorithms and elaborate their advantages and disadvantages. If a property of a clustering algorithm complies with the requirements, it will be marked with "✓". As table 3.1 shows, only partitional clustering and hierarchical clustering are worth considering. Hierarchical clustering has some advantages, for instance it provides additional hierarchical information, which is very worthy in our case. A hierarchical clustering result can represent the hierarchical structure of performance problems. Figure 3.3 shows a possible hierarchical clustering result. The four instances can be clustered into two different clusters. However, the two different clusters also have a similarity. Both inherit problem instances, which occurred in the context of a remote call. Thus, the two clusters "HTTP call" and "DB Call" can be grouped in a super cluster called "Remote call".

However k-means, a representative partitional cluster algorithm is much faster than an hierarchical approach and scales better for a huge number of patterns. Thus, we implement both algorithms and compare them in the evaluation.

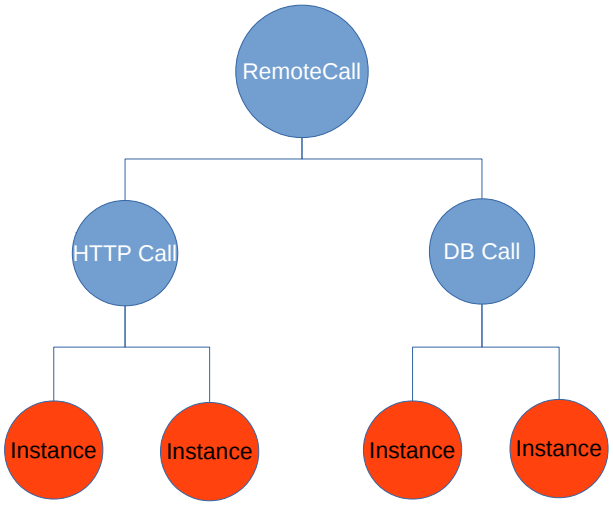


Figure 3.3. Hierarchical clustering result

Table 3.1. Comparison of different clustering algorithms

Algorithm	**Advantages**	**Disadvantages**	**Performance (Complexity)**	**Uniqueness (number of solutions)**	**Independence (no predefined number of clusters)**
Hierarchical	The result has a hierarchical structure. The result is independent of any order or randomized choice of initial clusters	Bad computation time: iterates over all distances more than one time	$O(n^2 log(n))$ ✓	1 ✓	yes ✓
Partitional k-means	Very Fast	Fitness of the result depends on the first randomly chosen partition. The number of clusters has to be known before	$O(k \times n \times l)$ ✓	1 ✓	yes (with k estimation and randomized seed)✓
Artificial Neural Network Clustering	perform well in parallel	Need knowledge about the domain to perform well	It depends on the implementation	1 ✓	no
Fuzzy (c-means)	Result has more information	Bad computation time	$O(ndc^2 i)$	1 ✓	no
Evolutionary Clustering	more than one solution as a result	Bad computation time	$O(k^6 n) \mid k > 10$	n	yes✓

3.3 Cluster engine

The cluster engine is the component, which performs the specific cluster algorithm. In our case, the cluster engine is able to apply a k-means with k-estimation (Section 3.2.2) and a hierarchical clustering on the given performance problem instances. The k-means approach is used in combination with k-estimation, because otherwise, the requirement of independence (Section 3.2.1) would be violated. Using additional training data, which can be a subset of the input instances, the optimized k-means runs itself with different number of clusters. Based on the quality of the clustering results, k-means chooses the k which provided the best result. Additionally, k-means runs multiple iterations and starts with different cluster centroids to avoid any influence of the order. The best solution will be chosen. In the further thesis, k-means includes k-estimation.

Figure 3.4 shows, that *diagnoseIT* provides the detected performance problem instances to the Cluster Engine. First, the problem instances (1) are converted into vector based data structure (2). Each data only consists of the used attributes which are captured by diagnoseIT. In Figure 3.4, the vector consists of three attributes. Afterwards the cluster engine executes the actual cluster algorithm. If k-means is used, the k-estimation is executed(*) and provides an optimal number of clusters. The algorithm computes the pairwise distances between the vectors (3). Based on the computed distances, the algorithm groups the instances into problem categories. Instances, which are nearer to each other are put in the same problem category. Finally the cluster engine returns the clustering result. The clustering result provides the member of each cluster.

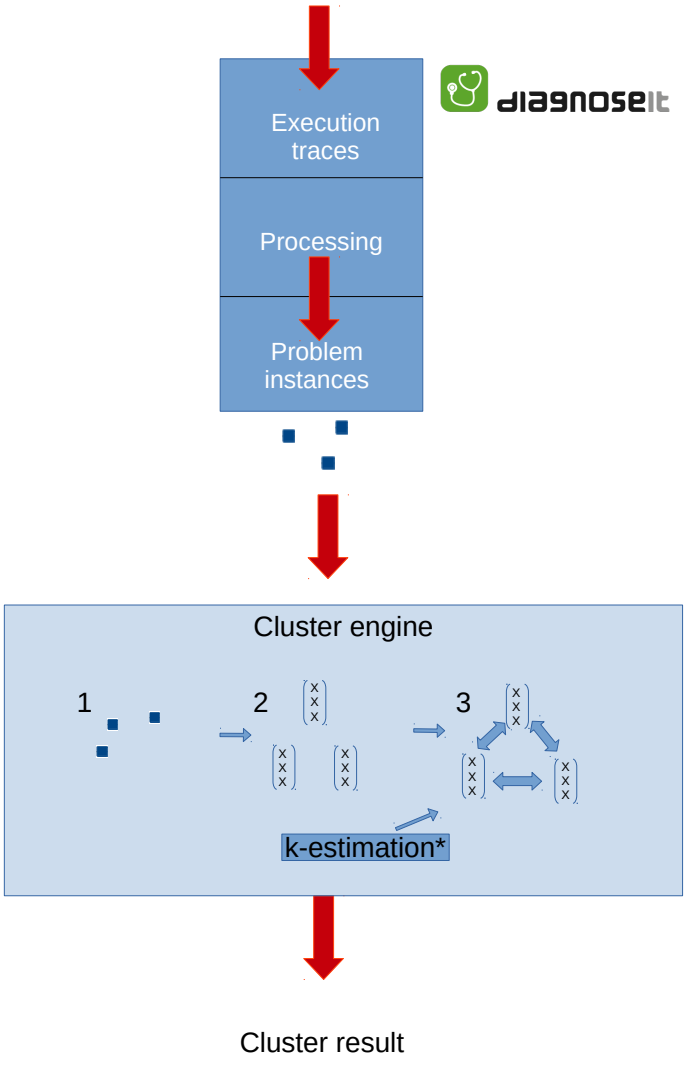


Figure 3.4. Expected Pipeline

Chapter 4

Approach of Optimization

The categorization approach categorizes the performance problems using the provided attributes for instance the exclusive response time and the business transaction id. To guarantee an optimal result, it is necessary to add a specific weight to each parameter. For each new monitoring environment, the weights have to be calibrated again, because the influence of each attribute strongly depends on the architecture and the use cases of the monitored application. This has to be done with an optimization approach which returns the weight for each attribute as result. By default, each attribute has the default weight of *1*. The weights are defined globally and can be accessed by the optimization engine and the cluster engine. Figure 4.1 shows that the optimization engine also uses the cluster engine to optimize the weights. It triggers the cluster engine with different weights and mutates the weights based on the quality of the last clustering results. After the optimization has finished, the weights will be updated and all upcoming performance problem instances will be clustered with the updated weights.

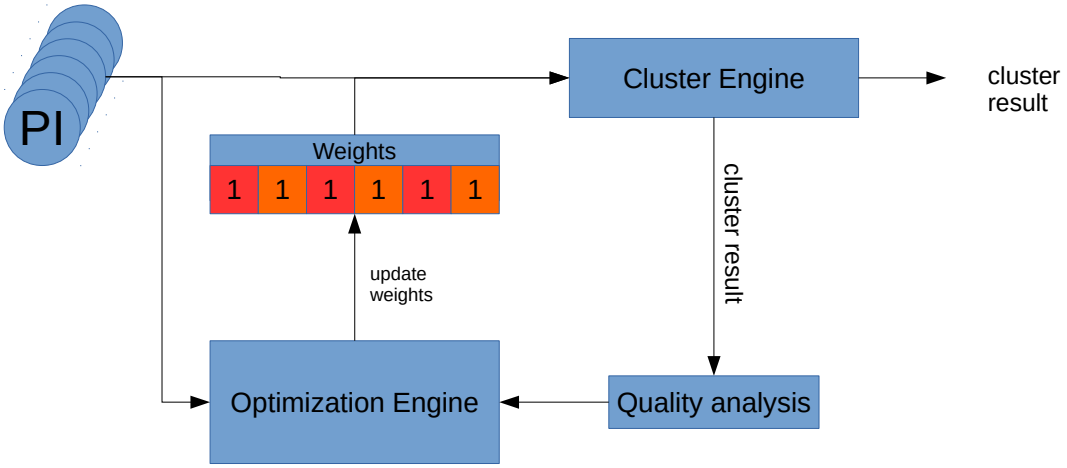


Figure 4.1. Optimization pipeline

The section is structured as follows. Section 4.1 introduces metrics which can be used to evaluate a clustering result. Section 4.2 introduces different optimization approaches.

4.1 Quality of a clustering result

In order to compare and evaluate clustering results, it is important to use a metric, which rates the quality of a clustering result. Especially the automated optimization approaches need criteria to decide, how good the current solution is (Section 2.1). There are two general ways how to evaluate the a clustering result [20]:

- Internal quality criteria: Internal quality criteria rate the clustering result only based on internal data. External data is not needed. The criteria are evaluating the similarity of all instances which are belonging to the same cluster and the difference to instances of different clusters. This approach rates the result based on algorithmic considerations.
- External quality criteria: External quality criteria compare the clustering result with an already clustered set of instances. In our case, we are using a reference cluster (Section 2.1).

In the following we would like to present you an internal and an external criteria:

4.1.1 Internal quality approach: Sum of squared errors (SSE)

The SSE approach is measuring homogeneity of each cluster by computing the squared distances of each instance to the cluster center. The complete error rate of one clustering result is represented by the sum of the SSE values of all clusters: $SSE_{Total} = SSE_1 + SSE_2 + SSE_3 + SSE_4 + SSE_5$. The SSE approach can be only used as indicator, whether a clustering result seems to be valid or not, because it only analyzes the result on a algorithmic level. The SSE cannot determine the professional quality of the clustering result.

4.1.2 Internal quality approach: Standard deviation of cluster center distances

Another internal quality criteria is the distance of the cluster centers. The more distant the cluster centers are, the better is the clustering result. However the average distance of the cluster centers does not represent the goodness of clustering result, because it could be, that one cluster is very far from a group of clusters and this distance will increase the average of the distances. Such a clustering result is not appropriate. Additionally it is important, that the clustering results are equally distributed. Thus, the standard deviation of the distances between the cluster centers is used to compare the different clustering results.

4.1.3 External quality approach: Kappa statistics

The basic idea using kappa statistic is to compare the current clustering result with existing optimal clustering results. This problem is quite similar to the problem of measuring inter-rater reliability [23]. Consider two observers are categorizing several items. In some

points, both observers agree and in some points the two observers disagree. The Kappa statistic tries the evaluate the agreement between the two optimizers compared to the expected agreement, if the two observers would randomly categorize the items. As a result, the Kappa analysis returns the kappa value, which represents the level of agreement between the two observers. This scenario can be easily transfered to the comparison of two clustering results. In our case, the two observers are the clustering results and the categories are the clusters. The analogy can be understood by the following example

In table 4.1, two clustering results with three clusters each are represented. Each cell represents the number of instances, which are member in cluster i of clustering result 1 and member in cluster j of clustering result 2. Kappa now uses the diagonal of the table, because the diagonal represents the agreement of both clustering results. The range of kappa is between 0 and 1, thus it can be also seen as percentile agreement. Usually, Kappa statistics is used in cases, where the categories of both observers are equal. In our case, it is not clear, which cluster of the first clustering result belongs to which cluster of the second clustering result. Thus, we are optimizing the Kappa value and permute the order of the clusters.

Table 4.1. Example: Usage of Kappa statistics for clustering result comparison

	Clusterresult 2			
Cluster result 1	1	2	3	j
1	3	2	1	⋮
2	2	7	2	⋮
3	2	2	5	⋮
i	⋮	⋮	⋮	⋮

The introduced metric can be also used to determine the quality of a cluster solution: We use a reference cluster (Section 2.1), which represents the optimal way of how the instances should be clustered. This method is also used in the fitness function (Section 2.1) of the optimization approaches Hill-climbing and evolutionary clustering (see [23, 18]).

4.2 Overview of optimization approaches

There are several different approaches for optimization algorithms. In the following, we introduce three. All approaches should be able to learn from former results. We analyzed the well-known optimization approaches "Hill-climbing", "Evolutionary Optimization" and additionally our own interactive approach called "Human-in-the-loop".

4.2.1 Evolutionary Optimization

Evolutionary optimization is an iterative approach. The algorithm generates weights and the corresponding clustering results in each so-called generation. Figure 4.2 shows a general pipeline of an evolutionary algorithm. The algorithm generates several clustering results. All clustering results get rated by a fitness function (Valuation). If the best rated weights fits into the predefined requirements, the weights will be accepted, otherwise, the k best weights get manipulated and modified (Reproduction) in the next iteration. The algorithm stops if a predefined fitness threshold or a predefined number of generations is reached.

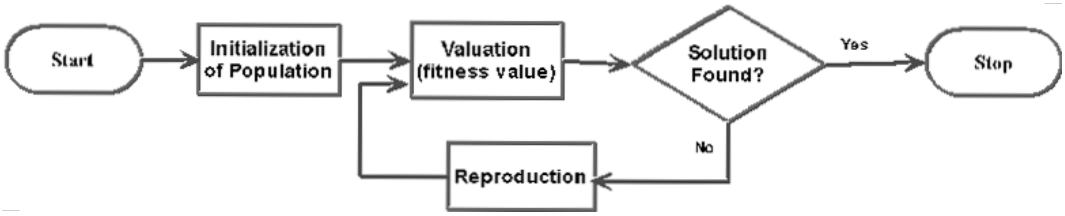


Figure 4.2. Abstract pipeline of an optimization approach [7]

4.2.2 Hill-climbing Optimization

Hill-climbing [21] is quite similar to the evolutionary approach: Each weight will be modified until the result is improving. The algorithm stops if a defined stop criteria is fulfilled. Possible stop criteria could be the number of iterations or the change of the fitness function between the iterations. Hill-climbing does not need any complex data structure. In each iteration, it just stores the best weights and the corresponding evaluation of the fitness function. If there are multiple best weights to choose from, hill-climbing randomly chooses the best set of weights.

4.2.3 Human-in-the-loop optimization

Figure 4.3 shows a concrete concept of optimization. The basic idea of the human-in-the-loop optimization is to include the expertise of the performance analyst into the optimization process. The categorization algorithm returns several solutions, which are based on different weights. Instead of using a fitness function, the approach uses the knowledge of the performance analyst to optimize the weights. In each iteration, the performance analyst has to choose the best result. Based on the choice of the performance analyst, new solutions based on permuted weights are provided. The process of optimization stops, when the performance analyst is satisfied with the current clustering result.

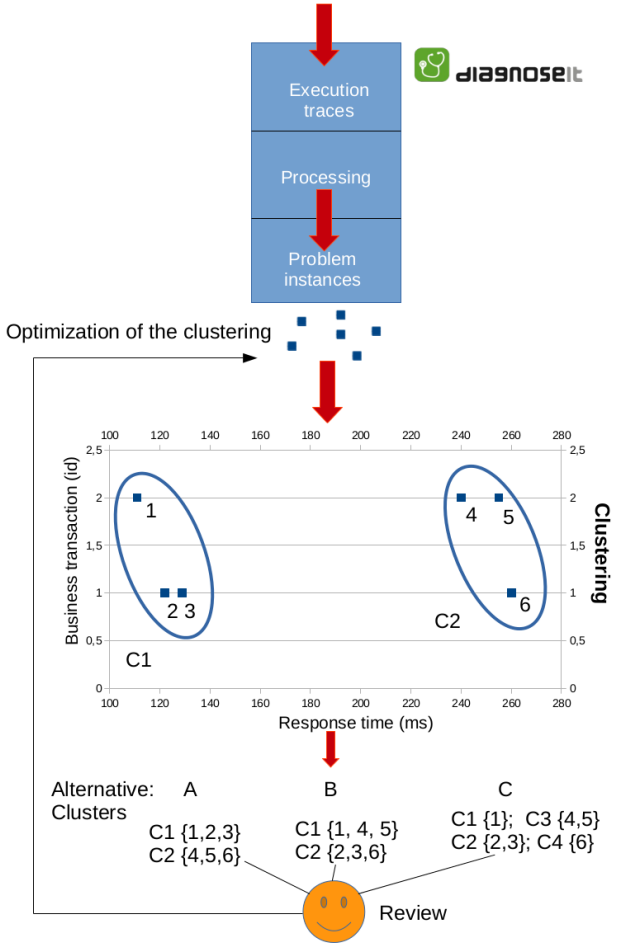


Figure 4.3. Expected Pipeline with human-in-the loop optimization

Weight manipulation

In each iteration of the human-in-the-loop optimization, a number of solutions have to be defined. The number of provided solutions should not be to high and cluster solutions should not be to similar. There are several ways to manipulate the weights. In the following, we present two approaches.

Increasing and Decreasing each value With k features, there are existing 2^k permutations. With six features, the human-in-the-loop optimization engine would have to provide 64 possible solutions, which is way to much: One performance analyst cannot choose the

best clustering result of 64 in an acceptable period of time. Thus, we use a simplified concept of optimization. In each iteration, the optimization algorithm provides $2k + 1$ solutions:

- one baseline solution, which is unweighted (compared to the previous iteration)
- weight $1 \dots k$ get increased
- weight $1 \dots k$ get decreased

Each k feature will be multiplied with a factor g. After each iteration, the factor decreases by a factor h, because otherwise the user can choose a clustering result, which is similar to a clustering result of the previous iteration. The weights of the chosen clustering result will be the baseline result of the next iteration. Thus, the user can compare the results with the old result of the previous iteration.

Compared to the complete optimization approach, the simple approach ignores some permutations. Thus, the simple approach needs more iterations, but less solutions per iteration. It is recommended to choose $0,5 \leqslant h \leqslant 1$ and g should be relative high to h, because in the first iterations, the provided clustering results should be very different. The more iterations the user executes the more similar will be the provided solutions.

Example To simplify the example, we use the following factors:
$k = 3$
$g = 3$
$h = 0.5$

$$\overrightarrow{W} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\overrightarrow{W_1} = \begin{pmatrix} 1 * g \\ 1 \\ 1 \end{pmatrix}; \overrightarrow{W_2} = \begin{pmatrix} 1 \\ 1 * g \\ 1 \end{pmatrix}; \overrightarrow{W_3} = \begin{pmatrix} 1 \\ 1 \\ 1 * g \end{pmatrix};$$

$$\overrightarrow{W_4} = \begin{pmatrix} 1 * (1/g) \\ 1 \\ 1 \end{pmatrix}; \overrightarrow{W_5} = \begin{pmatrix} 1 \\ 1 * (1/g) \\ 1 \end{pmatrix}; \overrightarrow{W_6} = \begin{pmatrix} 1 \\ 1 \\ 1 * (1/g) \end{pmatrix}; \overrightarrow{W_{Baseline}} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Iteration 1

$g = 3$

$$\overrightarrow{W_{1.1}} = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}; \overrightarrow{W_{1.2}} = \begin{pmatrix} 1 \\ 3 \\ 1 \end{pmatrix}; \overrightarrow{W_{1.3}} = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix};$$

$$\overrightarrow{W_{1.4}} = \begin{pmatrix} 0.333 \\ 1 \\ 1 \end{pmatrix}; \overrightarrow{W_{1.5}} = \begin{pmatrix} 1 \\ 0.333 \\ 1 \end{pmatrix}; \overrightarrow{W_{1.6}} = \begin{pmatrix} 1 \\ 1 \\ 0.333 \end{pmatrix}; \overrightarrow{W_{Baseline}} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$g_{new} = g_{old} * h \rightarrow g_{new} = 3 * 0.5 = 1.5$

The performance analyst has choosen the result, computed with the weights $\overrightarrow{W_{1.3}} = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}$.

Iteration 2

$g = 1.5$

$\overrightarrow{W_{2.1}} = \begin{pmatrix} 1.5 \\ 1 \\ 3 \end{pmatrix}; \overrightarrow{W_{2.2}} = \begin{pmatrix} 1 \\ 1.5 \\ 3 \end{pmatrix}; \overrightarrow{W_{2.3}} = \begin{pmatrix} 1 \\ 1 \\ 4.5 \end{pmatrix};$

$\overrightarrow{W_{2.4}} = \begin{pmatrix} 2/3 \\ 1 \\ 3 \end{pmatrix}; \overrightarrow{W_{2.5}} = \begin{pmatrix} 1 \\ 2/3 \\ 3 \end{pmatrix}; \overrightarrow{W_{2.6}} = \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix}; \overrightarrow{W_{Baseline}} = \begin{pmatrix} 1 \\ 1 \\ 3 \end{pmatrix}$

$g_{new} = g_{old} * h \rightarrow g_{new} = 1.5 * 0,5 = 0.75$

Experimental design by Plakett and Burman [16] Another way more efficient approach for evaluating many parameters is the Plakett-Burman design. The number of solutions per iteration is dependent on the number of attributes and the number of different manipulations of each attribute, which is also called level l. To check all permutations of k attributes, one basically needs l^k solutions. In our case, the level is $l = 2$, because we only have two states: Increase and decrease. With the Plackett-Burman design, the process can be optimized. With N solutions, it is possible to evaluate $\frac{N-1}{l-1}$ number of attributes. With level $l = 2$ and N solutions, we can evaluate $N - 1$ attributes The design usually will visualized by a matrix. The rows represent the solutions and the columns represent the different factors of each attribute. $" + "$ increases the factor and $" - "$ decreases the factor. It applies: $N = x * 4 \mid x \in \mathbb{N}$
The matrix is built from a reference column:

+	+	+	-	+	-	-	-

Each column is shifted -1 to the previous column. The last row represents the baseline configuration.

1	+	-	-	+	-	+	+	+
2	+	+	-	-	+	-	+	+
3	+	+	+	-	-	+	-	+
4	-	+	+	+	-	-	+	-
5	+	-	+	+	+	-	-	+
6	-	+	-	+	+	+	-	-
7	-	-	+	-	+	+	+	-
8	-	-	-	-	-	-	-	-

In our case, six different factors are enough. Thus, we only need six columns:

1	+	-	-	+	-	+
2	+	+	-	-	+	-
3	+	+	+	-	-	+
4	-	+	+	+	-	-
5	+	-	+	+	+	-
6	-	+	-	+	+	+
7	-	-	+	-	+	+
8	-	-	-	-	-	-

Comparison of manipulation approaches Whereas Increasing and Decreasing each value covers more combinations, the Plackett-Burman approach needs less generated solutions per iteration and allows a significant prediction about the influence of each attribute. All important combinations of weights are represented. For 6 attributes, we only need 8 solutions and save 5 solutions in each iteration. Thus, we implement the Plakett-Burman approach.

Mockup of the human-in-the-loop approach

The human-in-the loop approach needs a very usable user interface, which supports the performance analyst to choose between different clustering algorithms. Figure 4.4 shows a possible graphical user interface (GUI). In each iteration, the user has to choose the best result. While clicking on the "next iteration" button, the optimization algorithm generates new solutions. The user can also step back to correct the previous choice. Each solution is visualized by a list of the corresponding clusters. The used weights and the number of clusters is visible above. Figure 4.5 shows the comparison view. The user can unfold an entry of the cluster list and can analyze the cluster with the statistic information, which are provided. The button "view instances" opens a pop-up, which provides a detailed list of all consisting instances. The slider can be used to rate the two different clustering results. With the navigation bar, the user can go to the next comparison or can repeat the previous comparison. This view is opened for each pairwise comparison.

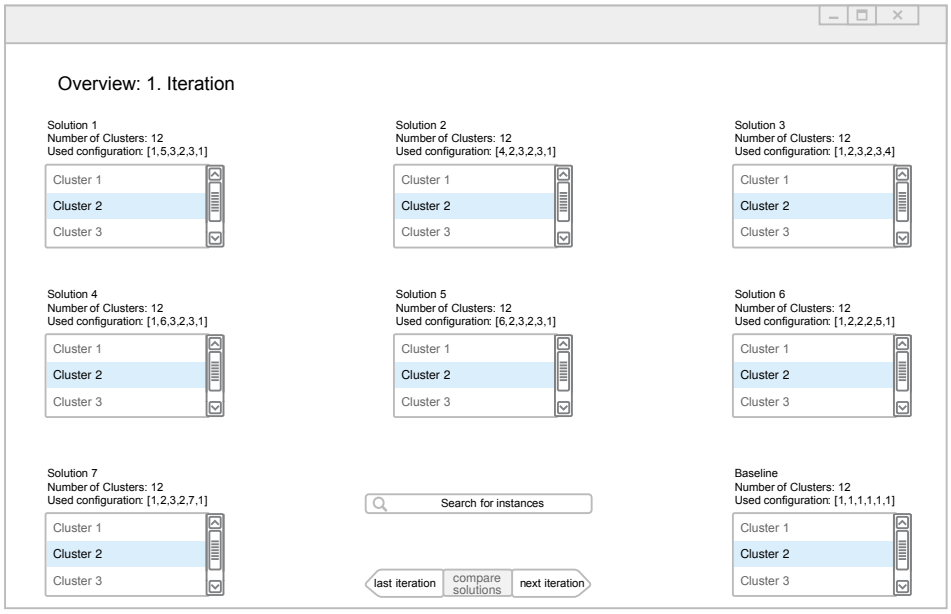


Figure 4.4. GUI Mockup: Overview

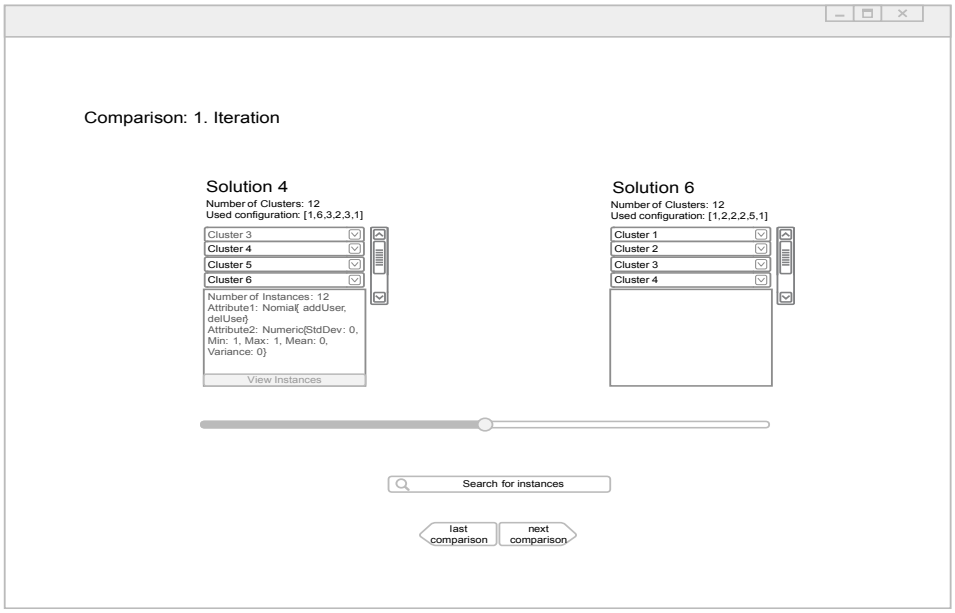


Figure 4.5. GUI Mockup: Comparison

Chapter 5

Implementation

In the following, the structure of the implementation of the cluster engine and the optimization engine is presented. In Section 5.1 the used technologies are introduced. In Section 5.2 and in Section 5.3 the data structure and the implementation structure are presented.

5.1 Used technologies

Based on our requirements (Section 1.2) the implementation is integrated into the diagnoseIT-inspectIT prototype [12]. Thus, we use Java1.8 as programming language, because inspectIT and diagnoseIT are written in Java and the monitoring currently mainly focuses on Java applications. For the clustering, we use the Cluster framework Weka (Section 2.3.2), which provides an huge amount of clustering algorithms. For optimization, we use two different engines: The JAMES Framework (Section 2.4.3) provides hill-climbing algorithm, which we use to optimize the weights of Weka. For the evolutionary approach we use the Opt4J framework, which provides us several evolutionary algorithms and a GUI.

5.2 Data structure

From the raw set of problem instances to the clustering result, we used many different data structures, because for the hierarchical clustering, Weka, only provides the clustering result as an String. Figure (Section 5.1) shows different stages with data structures for the hierarchical clustering. diagnoseIT triggers our clustering pipeline and provides a set of unsorted problem instances. The engine analyzes the problem instance and extracts all important attributes. It creates a new Weka instance for each problem instance with all extracted attributes. After hierarchical clustering, Weka provides a clustering result as String. This String has to be parsed into a tree structure, because the tree structure can be better visualized and instances can be better accessed. The tree represents several level of clusters. The cluster engine converts the tree into a clustering result list, which consists all clusters with its instances. Afterwards the user gets represented the clustering result list (Section 4.2.3).

For using k-means, we can easily access the membership of each instance and no complex data transformation is needed.

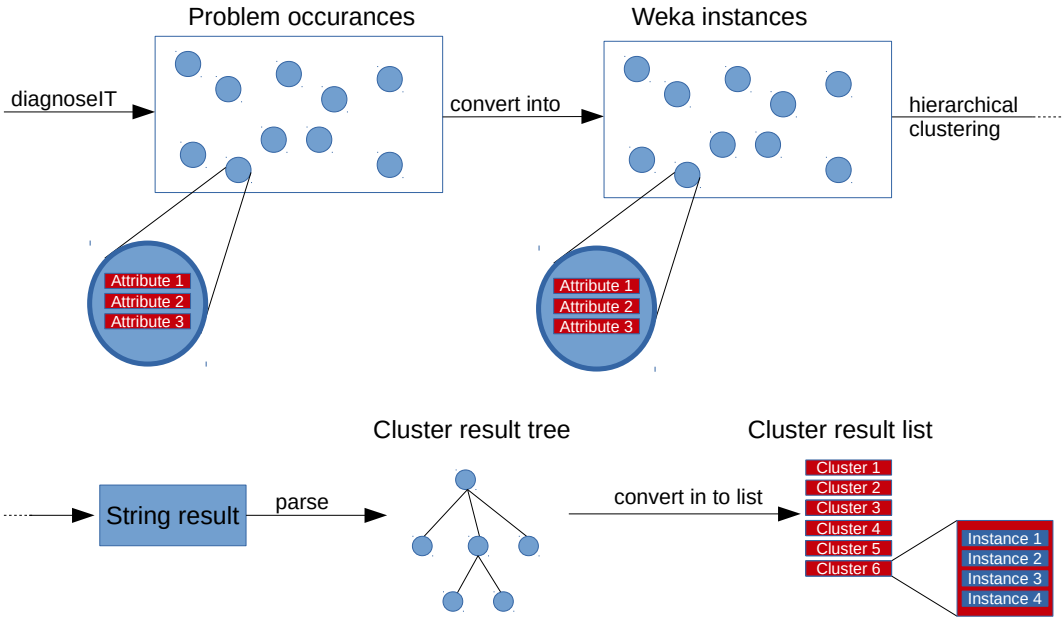


Figure 5.1. Data structure pipeline

5.2.1 Cluster result tree

The clustering result tree represents the result of the hierarchical clustering. The tree offers additional information to the user. An inner node can be interpreted as a main problem. Each level gets more specific. The tree can hold two type of nodes. A *ClusterNode* can have other ClusterNodes as children. A *ClusterLeaf* is a subclass of *ClusterNode* and can hold a list of instances which represents the smallest unit of clusters in the tree. A *ClusterLeaf* cannot hold any children. The cluster engine also provides the possibility to shorten the tree by level. The algorithm scans the tree, until it reaches the predefined level. All Leafs, which are connected directly or indirectly with the current node are added to the current node. All other ClusterNodes are deleted. Figure 5.2 show the tree transformation at level 2 (1). All ClusterLeafs, which are children of ClusterNodes with an higher level than 2 get pulled to the next ClusterNode at level 2 (2). Afterwards, all ClusterNodes with a level higher than 2 are deleted (2). As result, we get six separate clusters (3).

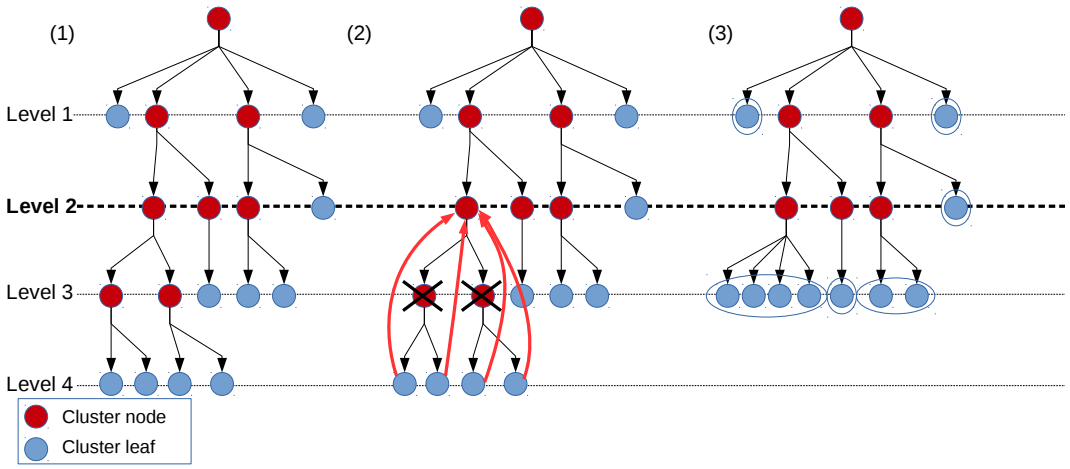


Figure 5.2. Tree transformation

5.3 Implementation structure

The following section introduces the implemented work packages. It presents their function and the interaction in between. Figure 5.3 shows the package structure.

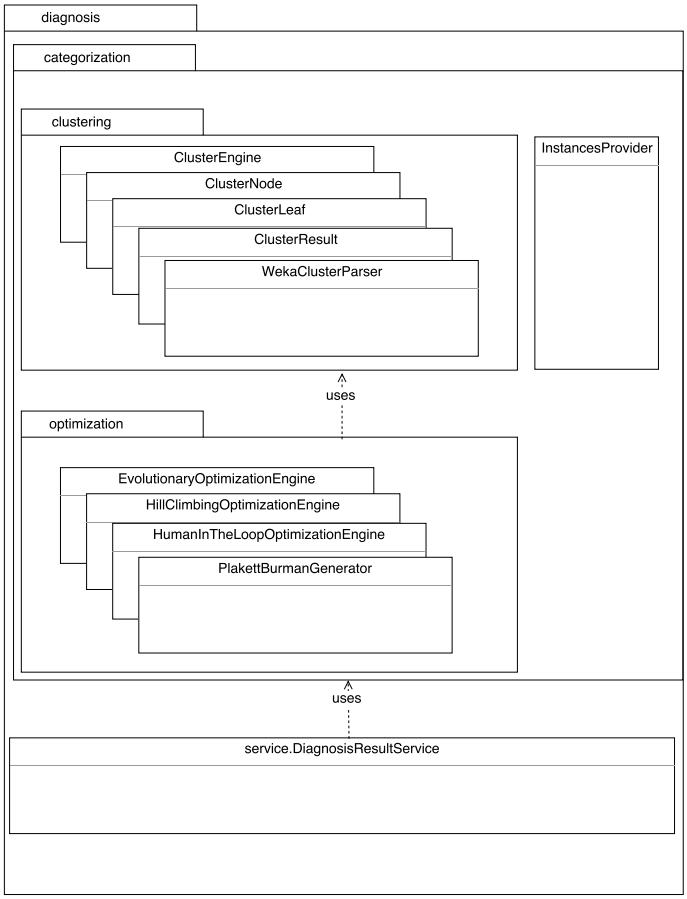


Figure 5.3. Package diagram of the prototype

5.3.1 Categorization

The categorization package consists of the packages *classification* and *optimization*. In addition, the package consists of the class *InstancesProvider*.

InstancesProvider

The class InstancesProvider provides random generated instances and the corresponding attribute list. The following are used:

- Root cause
- Problem context
- Entry Point
- Global Context
- Node Type
- Exclusive Duration

Additionally, InstancesProvider provides a set of static instances, which do not change, to guarantee an fair test environment. This class is mainly used for testing and evaluation.

5.3.2 Clustering

The clustering package is responsible for the core implementation of the clustering. The *ClusterEngine* provides the method *createClusterResult()*, which provides a list of clusters with the corresponding weights. We use the *Complete Link-HierarchicalClusterer* of the Java Framework Weka (Section 2.3.2) and the *Expected Maximization (EM)* algorithm, which optimizes the estimated number of clusters using k-means (Section 3.2.2). Both approaches are using the euclidean distance metric.

Figure 5.4 shows the output of the cluster engine. It presents each cluster with several statistics: The user gets an overview over the occurred values of each attribute. Numeric attributes are statistically evaluated. In addition, we provided the number of clusters and the weights.

```
Cluster 0
Number of Instances: 2
[Nominal] Occured values of rootCause: [4797(2)]
[Nominal] Occured values of problemContext: [5068(2)]
[Nominal] Occured values of entryPoint: [5152(2)]
[Nominal] Occured values of globalContext: [5068(2)]
[Nominal] Occured values of nodeType: [SINGLE(2)]
[Numeric] Statistics of exclusiveDuration: [StDev: 493.897116096055] [Variance:
    243934.36128800007] [Mean: 1169.106264] [Confidence interval(95%):
    {484.599784<->1853.612744}
Cluster 1
Number of Instances: 1
[Nominal] Occured values of rootCause: [5114(1)]
[Nominal] Occured values of problemContext: [4797(1)]
[Nominal] Occured values of entryPoint: [5152(1)]
[Nominal] Occured values of globalContext: [5068(1)]
[Nominal] Occured values of nodeType: [RECURSIVE(1)]
[Numeric] Statistics of exclusiveDuration: [StDev: 0.0] [Variance: 0.0] [Mean:
    1518.344264] [Confidence interval(95%): {1518.344264<->1518.344264}
Cluster 2
Number of Instances: 40
[Nominal] Occured values of rootCause: [5152(2)] [4819(5)] [5068(1)] [5114(9)]
    [4765(19)] [4811(4)]
[Nominal] Occured values of problemContext: [5152(3)] [4819(9)] [5068(19)]
    [4811(4)] [4825(5)]
[Nominal] Occured values of entryPoint: [5152(40)]
[Nominal] Occured values of globalContext: [5152(26)] [4819(2)] [5068(12)]
[Nominal] Occured values of nodeType: [SINGLE(11)] [RECURSIVE(29)]
[Numeric] Statistics of exclusiveDuration: [StDev: 18728.6668542554] [Variance:
    3.507629621376849E8] [Mean: 13046.803582550001] [Confidence interval(95%):
    {7242.729592351113<->18850.87757274889}
```

Figure 5.4. Output of the cluster engine

The cluster engine provides the following statistical information:

- Number of instances
- The occurrence of nominal attributes
- Standard deviation of numeric attributes
- Variance of numeric attributes

- Mean of numeric attributes
- Confidence interval for numeric attributes: the confidence level is 95 %

5.3.3 Optimization

The optimization package provides three implementations of different optimization approaches: Human-in-the-loop, hill-climbing and evolutionary optimization. For each optimization approach, it is necessary to translate the goal of optimizing the weights of the cluster engine into an optimization problem. All optimization engines can be executed with their main method.

Human-in-the loop

The human-in-the-loop implementation generates different results in each iteration. In the current prototype, the different clustering results are printed on the command line. The user can determine the used factor, which manipulates the weights of the cluster algorithm and can choose the best result in each iteration. The performance analyst either can use the Plakett-Burman generator or the approach of increasing and decreasing values, to generate different permutations of weights (see Section 4.2.3).

Evolutionary Optimization

The evolutionary optimization engine uses the Opt4J (Section 2.4.2) framework, to optimize the weights of the cluster engine. The optimization problem is defined by the following components:

- *Creator*: This component creates new randomized weights.
- *Problem decoder*: This component runs the cluster engine with the given manipulated weights and provides the clustering result to the *Problem evaluator*.
- *Problem evaluator*: This component rates based on the fitness function (Section 2.1).

Hill-climbing

The hill-climbing optimization engine uses the James (Section 2.4.3) framework, to optimize the weights of the cluster engine. The optimization problem is defined by the following components:

- *Solution:* This component creates new randomized weights.
- *Problem data:* This component holds some general information about the optimization problem, for instance, the number of weights.
- *Problem objective:* This component evaluates the given weights with the fitness function of the evolutionary approach.

- *Problem OptMove:* This component defines a set of weights.
- *Problem OptNeighbourhood:* This component provides new manipulated weights.

Fitness function

Hill-climbing and evolutionary optimization are using the same fitness function. The fitness function (Section 2.1) maps the quality of a given reference cluster to a numeric value. The higher the value the better is the rated result. The fitness function computes the quality by metrics of the defined quality criteria in Section 4.1.

Chapter 6

Evaluation

The current chapter evaluates the approaches of Chapter 3 and Chapter 4. Whereas Section 6.4 analyzes the scalability of the cluster engine, Section 6.5 evaluates the survey to get a reference cluster (Section 2.1). Section 6.1 compares hierarchical clustering and k-means in terms of quality and execution time. Whereas Section 6.2 evaluates the sensitivity of the weights, Section 6.3 compares hill-climbing and evolutionary optimization in terms of the potential of optimization.

6.1 Comparison of the cluster approaches

In the following, we compare the hierarchical clustering approach with the k-means clustering approach. Hereby, we compare the response time and the quality of the cluster solutions. For both categories, we use the default configuration of the clustering algorithms, which are provided by the framework. Thus, both approaches are using the same distance metric, the weights do not influence the measurement.

6.1.1 Datasets

For the comparison of k-means and hierarchical clustering, we are using two types of datasets:

Real data

With the DVD store application [1], we generated 200 real performance problem instances. The performance problems are generated by 50 generated virtual users.

Synthetic Data

We generated 3000 random performance problems, which we use to evaluate the response time of the different clustering approaches.

6.1.2 Comparison of the clustering result quality

In the following, we compare the quality of the different clustering approaches depending on the number of performance problem instances.

Research question

The following experiment should answer the following research question:

- Which clustering approach provides the better results according to the predefined quality criteria (Section 4.1)?

Hypothesis

We assume, that the sum of squared errors of the k-means will be smaller compared to the hierarchical clustering results. The higher the level of the hierarchical cluster is, the higher is the standard deviation of the distances between the cluster centers.

Experimental design

We use the sum of squared errors and the distances between the cluster centroids as metric (Section 4.1.2). In this case, we are using real performance problem instances, which are generated with the DVD store application (Section 6.1.1). Using the JMH framework [2], we executed the tests with the following number of instances: 50, 75, 100, 125, 150 and 200. We use the hierarchical clustering with different number of levels.

Evaluation of the measurements

The following graph shows the sum of squared errors depending on the number of problem instances (Section 4.1.1):

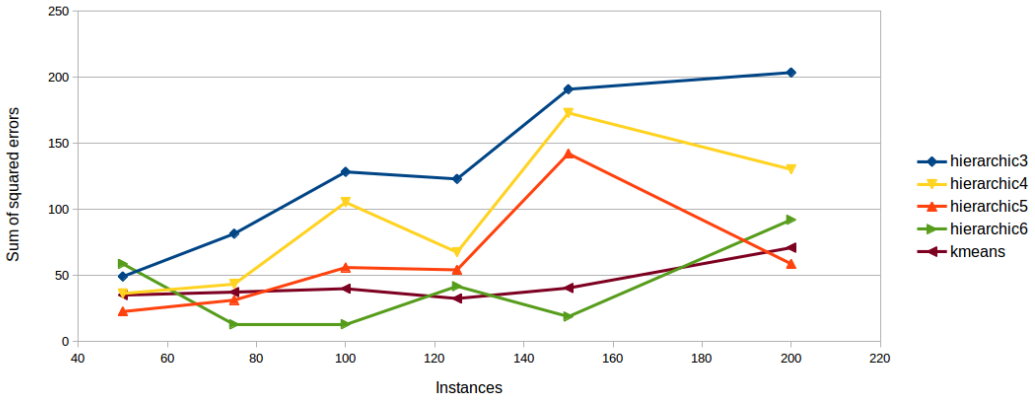


Figure 6.1. Comparison hierarchic clustering – k-means: sum of squared errors

The yellow, orange, green and blue lines are representing the hierarchical clustering with different level configurations. The purple line represents the k-means clustering. The runs which are using the level 3 and 4 do not perform well. They have a completely higher sum of squared errors. The runs which are using a the level 5 and 6 and the k-means clustering perform best. However, the distance between the cluster centroids is another metric which has to considered. The more similar the distances between the cluster centroids are, the more stable and the better is the clustering result (Section 4.1.2). Figure 6.2 shows, that, as assumed in the hypothesis, the runs of the hierarchical clustering using the levels 5 and 6 provide a much higher standard deviation of distances between the cluster centroids than the other configurations including k-means. In the run with 125 instances, the hierarchical clustering result with level 5 and 6 have a spike. The standard deviation is 8,2 for the hierarchic clustering using level 5 and 10,5 for the hierarchic clustering. This effect is caused by a clustering result, where one cluster has a very high distance to the other clusters. Such a standard deviation indicates a bad quality of the clustering result. Figure 6.3 shows the standard deviation in a zoomed view: Compared to the hierarchical clusterers the k-means clusterer stays almost linear and has a lower distributions of clusters, than the hierarchical clusters with the level 4, 5 and 6.

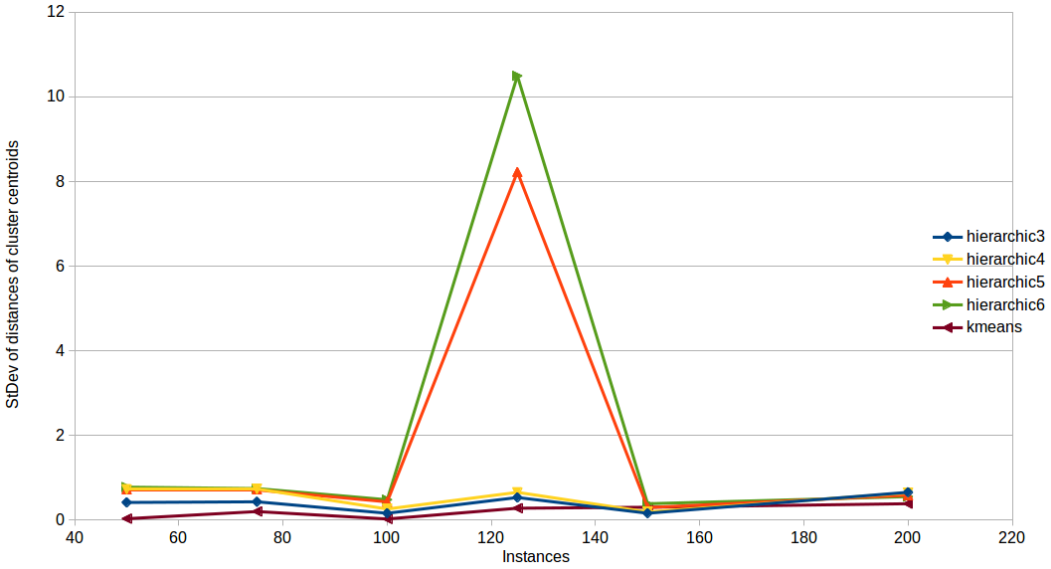


Figure 6.2. Comparison hierarchic clustering – k-means: number of clusters

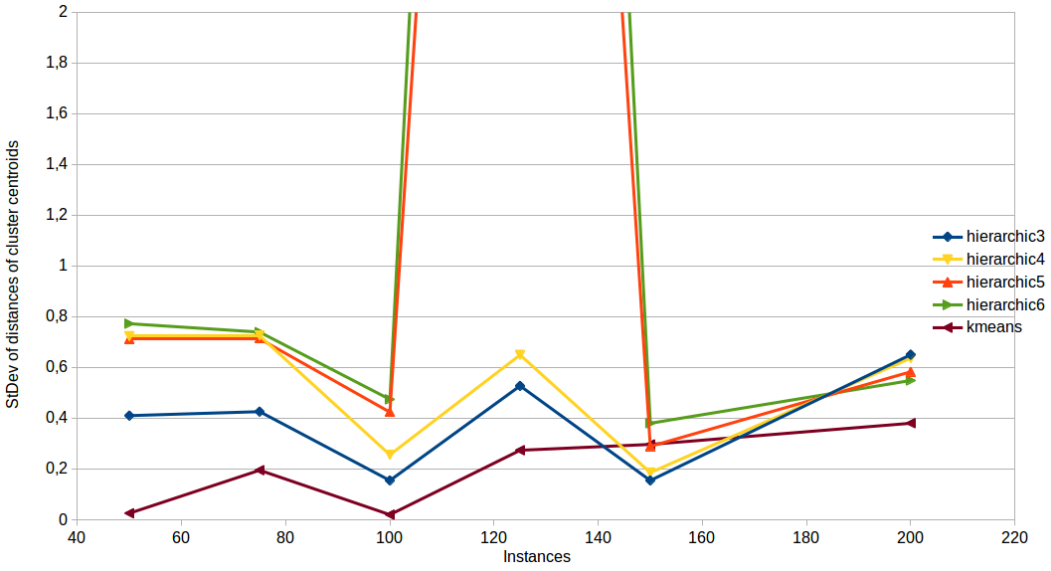


Figure 6.3. Zoomed view: Comparison hierarchic clustering – k-means: number of clusters

6.1.3 Evaluation of the response time

In the following, we compare the response time of k-means and hierarchical clustering depending on the number of performance problem instances which are provided by diagnoseIT.

Experimental design

We use the synthetic dataset (Section 6.1.1) and measure the execution time with the JMH framework [2]. We execute the JMH tests with the different number of instances: 10, 50, 100, 500, 1000, 1500, 2000, 2500 and 3000. We have chosen this range, because it provides the possibility to see trends, and make predictions for more instances. Additionally, we execute the hierarchical clustering with different level configurations: We choose the level 3, 4, 5, and 6, because in previous tests, those levels provided the most promising result. If we choose a level, which is higher than six, the clusters become to small and do not present single problem categories anymore. We run the cluster engine with 45 different configurations.

Hypothesis

We assume, that the k-means algorithm will be much faster than the hierarchical clusterer. The level will not have much influence on the clustering result.

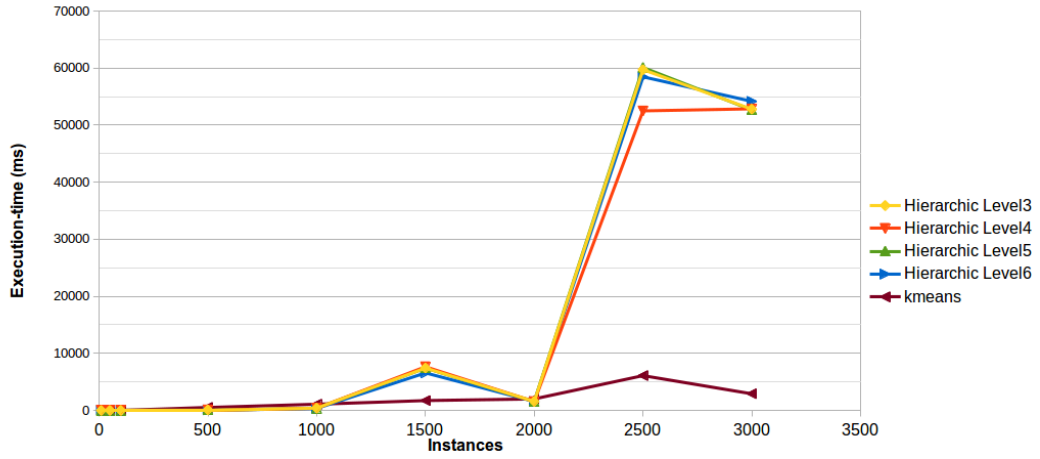


Figure 6.4. Comparison hierarchic clustering – k-means: execution time

Evaluation of the measurements

Figure 6.4 shows the response time of hierarchical clustering and k-means depending on the number of instances. The yellow, orange, green and blue lines represents the execution time of the hierarchical clustering with the levels 3, 4, 5 and 6. The purple line represents the execution time of the k-means. Until 1000 instances the execution time does not differ significantly. From 2000 instances, one can notice a significant rise of the hierarchic cluster execution times, meanwhile the k-means measurement stays more-less linear. As assumed in the hypothesis, the chosen level for the hierarchical clustering does not have any influence on the execution time.

Preconclusion Because the quality of the best hierarchical runs and the k-means differs not much and the standard deviation of the distances between the clusters for the hierarchical clustering algorithms is to high and the execution time of k-means clusterer is linear, we recommend to use the k-means algorithm in combination with k-estimation. However there might be also usecases, where the hierarchical structure of the problem categories (Section 2.1) might be useful. For further evaluation, we use the k-means algorithm.

6.2 Sensitivity analysis of the weights

In the following, we analyze the influence of each weight on the clustering results. We use the real dataset (Section 6.1.1). We use the One-at-a-time sensitivity measure method [10], because it is a very fast approach and it provides a sensitivity ranking of all attributes: We parameterize each weight in logarithmic steps: The values 0.001, 0.01, 0.1, 1, 10, 100 and 1000 are used, because they allow to detect trends, which are used to get a ranking of weights later on. All other attribute weights stay 1 by default. Thus, we have 42 single runs. In our case, each performance problem instance consists of the six following attributes:

- rootCause
- problem context
- entrypoint
- node type
- exclusive duration

6.2.1 Research questions

The sensitivity analysis should answer the following questions:

- Do the weights have a significant influence on the clustering result?
- Which attributes are more important?

- Which attributes are less important?

6.2.2 Hypothesis

The weights will have an influence on the clustering result. We assume, that the parameterization of the exclusive response time will have a great influence on the clustering result. The parameterization of the entrypoint will be very influencing, because all 200 problem instances have the same entrypoint. The cluster algorithm will probably ignore the entrypoint attribute.

6.2.3 Evaluation of the measurements

Table 6.1 shows the standard deviation of the sum of squared errors. The standard deviation of each attributes indicates the influence on the clustering result.

Table 6.1. Standard deviation of each attribute

Attribute	Standard deviation
root cause	3,726066611
problem Context	5,8492347018
entrypoint	0
global context	6,9926191778
node type	4,5394491354
Exclusive Duration	24,6326823402

Whereas the exclusive duration has a high influence on the clustering result, the entrypoint does not have any influence for the clustering result. Business transaction, global context and node type have a small influence on the clustering result. The results indicates, that the exclusive response time mainly is responsible for the cluster separation. In the case of the DVD store application, the entrypoint attribute is not necessary, because the weight of the entrypoint attribute does not have any influence on the clustering result. The reason for this is, that all 200 instances consist of the same entrypoint. Thus, k-means ignores the attribute. Figure 6.5 shows the sum of squared errors of the clustering result depending on the different weights. Figure 6.6 shows the standard deviation of the distances between the cluster centroids. The value using the weight 1 represents the baseline sum of squared errors in Figure 6.5 and the baseline standard deviation of the centroid distance in Figure 6.6. Except for the root cause attribute, there are many similarities between the Figure 6.7 and 6.5. If the sum of squared errors increases the standard deviation also increases. The measurements indicates a proportionality between the squared sum of errors and the standard deviation of the cluster center distances. However, as seen in Figure 6.6, the root cause attribute does not follow this pattern. If the root cause is weighted with a value, which is lower than 1, the standard deviation rises

up to 260, 12. Additionally, it is remarkable, that on the one hand, the exclusive response time has a big influence on the clustering result, but should be weighted lower than 1. The exclusive response time is the only numeric attribute which is subjected by a noise. Thus, it is recommended to weaken the weights. Otherwise, the standard deviation raises up, as seen in Figure 6.6.

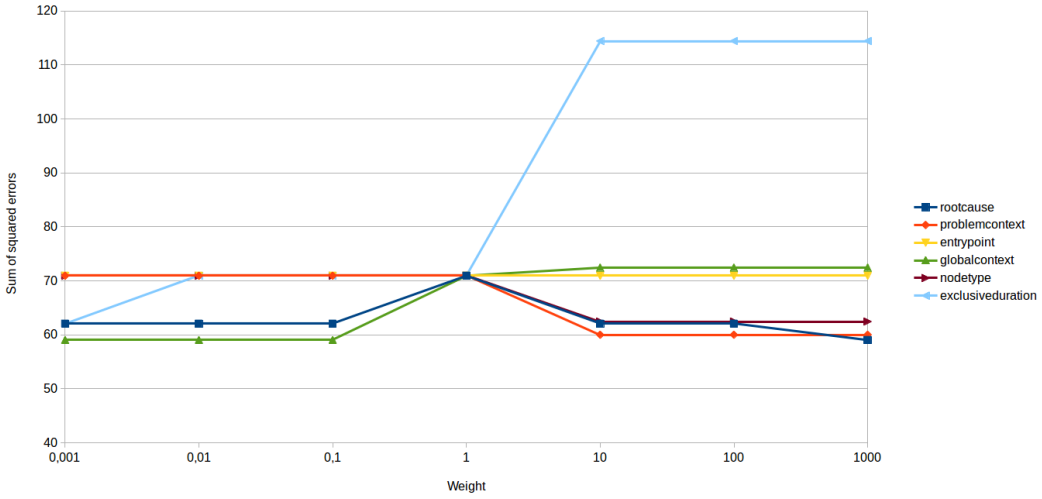


Figure 6.5. SSE depending on weights

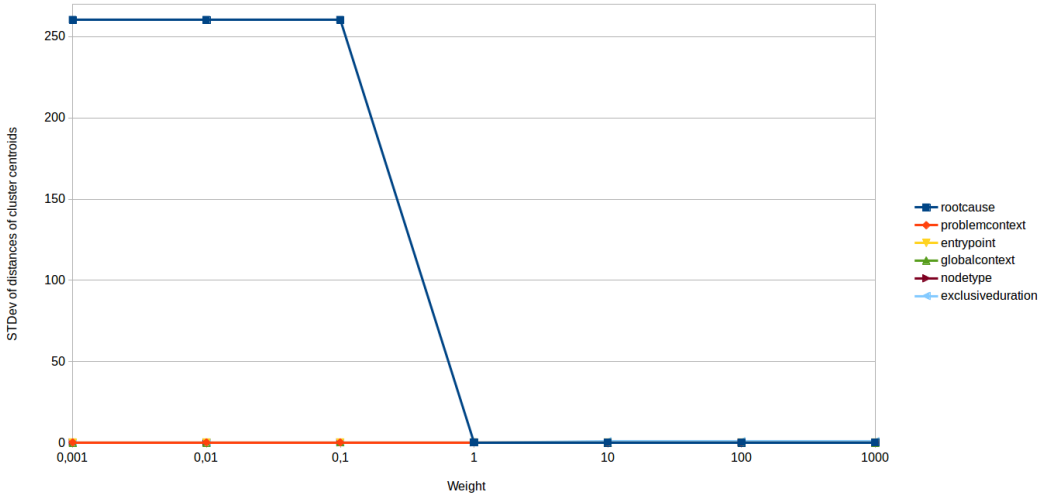


Figure 6.6. Standard deviation of centroid distances depending different weights

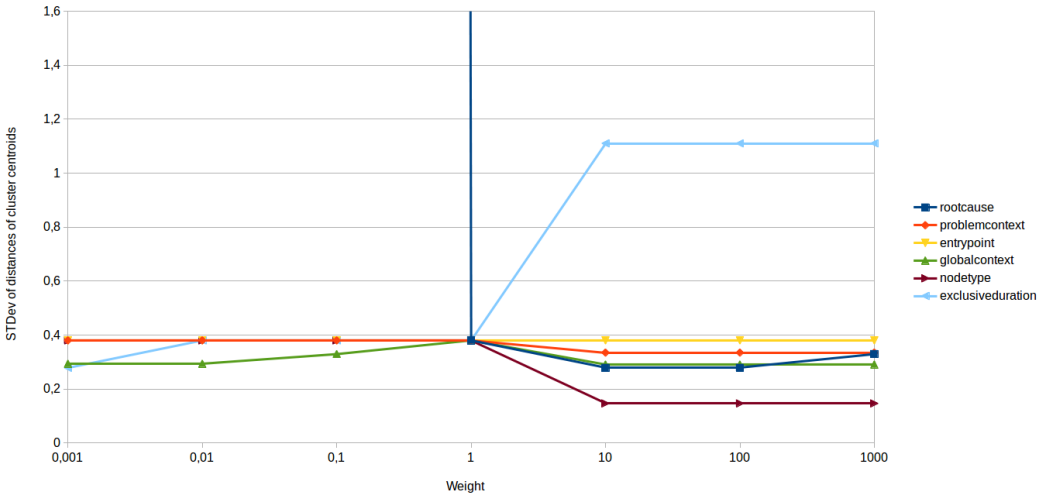


Figure 6.7. Zoomed View: Standard deviation of centroid distances depending different weights

Based on the presented measurements, we recommend to use the following weights. We only provide a trend compared to the default weights, because the weights can differ much depending on the problem instances:

Table 6.2. Recommended weights

Attribute	Weight range
root cause	>1
problem context	>1
entrypoint	not needed
global context	< 1
node type	> 1
Exclusive Duration	< 1

However, the recommended values are only representative for the DVD store application. To get more representative recommendations, it is necessary to analyze more different applications. We assume, that the optimal weights strongly depend on the analyzed application, because different use cases will occur different results.

6.3 Evaluation of optimization approaches

Thus, weights have an influence on the quality of clustering results, there is a potential for optimization. The weights have to be optimized different for each application, because the

influence of each attribute changes for each measured application. In the following we will evaluate automated optimization approaches.

6.3.1 Evaluation of the automated optimization

We analyze the two implemented automated optimization approaches hill-climbing and evolutionary optimization.

Evolutionary optimization Evolutionary optimization is an iterative optimization approach, which generates a number of weights in each iteration. The size of the generated solutions is limited by the population size. In the context of evolutionary optimization, a iteration is called generation. Each weight is rated by a fitness function (Section 2.1). A number of best weights are used and mutated in the next generation.

Hill-climbing Hill-climbing is an iterative optimization approach, which generates a number of weights in each iteration. After each weight is rated by the fitness function, the best weight is used in the next iteration.

Break condition Iterative optimization algorithms usually can have to types of break conditions:

- *fitness based*: The optimization algorithm stops, if a predefined fitness threshold is reached.
- *iteration based*: The optimization algorithm runs until a predefined number of iterations is reached.

In our experiment we decided to use the iteration based break condition. It is not possible to define a threshold, because the goal of this experiment is to evaluate, how far the cluster algorithm can be optimized. Additionally, we want to evaluate the influence of iterations on the quality of the resulting optimized cluster.

Fitness function In order to rate the weights, a fitness function has to be defined. Initially it was planned to compare a reference cluster with the returning clustering result using the current weights. However the results of the reference cluster study were to different to merge them into one reference cluster. In the following we discuss possible alternatives:

- Choose one reference cluster of the study.
- Choose an own reference cluster which is based on the results of the study.
- Use internal quality criteria to detect the fitness of the clustering result.

We decided to use the internal quality criteria (Section 4.1.2, 4.1.1), because the results are too diverse to decide for one specific reference cluster, let alone creating an own reference cluster which is based on the study results. The fitness function uses the Sum of

Squared Errors (SSE) and the standard deviation of the distances between the cluster centers to compute the fitness: $\boxed{fitness_{currentWeight} = SSE * StandartDeviation_{DistancesOfClusterCenters}}$

Parameterization We use the real data set (Section 6.1.1) as input data. Both approaches return optimized weights for the clustering and terminate after a predefined number of iterations. In the context of evolutionary optimization, the iterations are called generations. Additionally, the evolutionary optimization implementation also provides to configure the population size of each generation. Because we could not provide a representative reference cluster, we used internal quality criteria (Section 4.1.2, 4.1.1).

Research questions The experiment should answer the following questions:

- Taking into account the different configurations, which approach provides the best optimized weights?
- Which influence have the parameters *generation size* and *population size* on the quality of the weights?
- Do the optimized weights provide a significant better result than the default weights?

Metrics To answer the research questions it is necessary to use a metric, which determines the quality of the clustering result. We use the following internal quality criterias:

- Sum of squared errors (Section 4.1.1): Represents the accumulated distances of all members of a cluster to the cluster centers. The less the distances are, the higher is the quality of the clustering result.
- Distances between cluster centers (Section 4.1.2): The more distant and equally distributed the cluster centers are, the more stable an reliable is the clustering result.

Experimental design

Both algorithms are executed with the following number of generations: 10, 100, 200, 500, 1000, 2000 and 3000. Additionally, we execute the evolutionary optimization with the population sizes 10, 200, 200, 500 and 1000. The different configurations provide us the possibility to detect trends. All runs will be executed on the same set of problem instances, which are already used in Section 6.2. Thus, it is possible to compare the results of the optimization algorithms with the results of the sensitivity analysis. The resulting optimized weights are used to execute the cluster engine with the same problem instances. Afterwards, we analyze the provided clustering result by internal quality criteria (Section 4.1.2).

Hypothesis

We expect, that the evolutionary algorithm as a more complex approach will score better than hill-climbing. Furthermore, we assume, that a increase of generations and populations will increase the quality of the optimized weights and they will provide a significant better clustering result than the default weights.

Evaluation of the measurements

Figure 6.8 shows the sum of squared errors of the clustering results using the optimized weights of hill-climbing and evolutionary optimization. The light blue line represents the hill-climbing approach, the blue, orange, yellow, green and purple line represent the evolutionary approach with different population sizes. All evolutionary runs almost provide weights, which invoke clustering results with a similar quality. This indicates, that the population size and the generation size does not have any influence on SSE of a clustering result. The SSE of hill-climbing is much higher than the SSE of evolutionary optimization and after 200 iterations, it almost stays linear to 253. Table 6.3 shows, that from 200 iterations, hill-climbing only returns optimized weights which provides three clusters. However, Table 6.5 shows, that the optimized weights of evolutionary optimization provides clustering results with six and seven clusters. This explains why the SSE of Hill-climbing is much higher, because by increasing the clusters, the error rate decreases. Thus, the SSE is not only determining for the quality of a clustering result and the used weights.

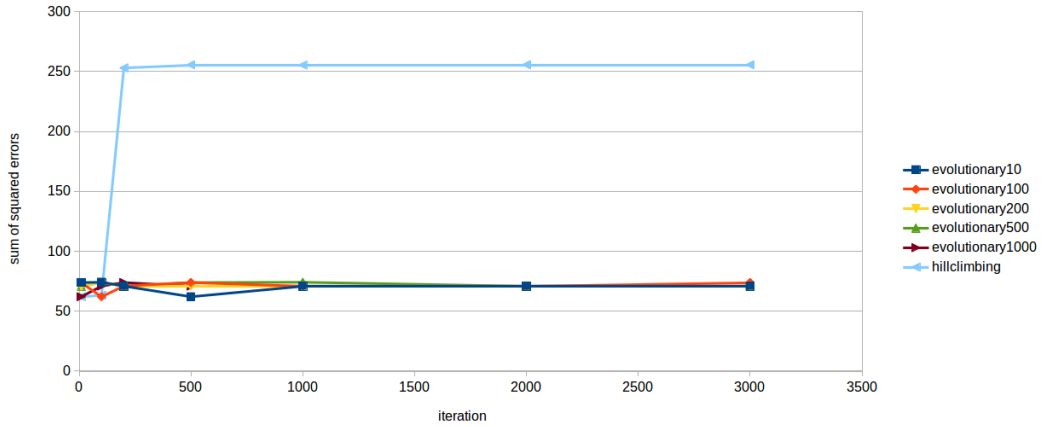


Figure 6.8. Sum of squared errors of the clustering results using the optimized weights of hill-climbing and evolutionary optimization

In the following we focus on the distances of the cluster centers. Figure 6.9 shows the standard deviation of the distances between cluster centers of the clustering results

using the optimized configurations of hill-climbing and evolutionary optimization. The standard deviation indicates a completely different valuation of the algorithms. Whereas the standard deviation of hill-climbing stays equal after a generation size of 200, the standard deviation of the evolutionary approach varies very much. Unfortunately there is no trend recognizable. The curves of the evolutionary approach more-less hop up and down. Table 6.6 shows, that, the standard deviation for different population sizes is quite similar. For instance a majority of the clustering results have a standard deviation of distances of 234, 343. This indicates that, the population does not have a significant influence on the quality of the optimized weights. The relative high standard deviation is caused by the huge difference between min and max. The shortest distance between two cluster centers is 38,526 and the longest distance is 767,886. It is also remarkable, that the smallest distance in the focused clustering result is significant higher than in other runs of the evolutionary clustering or in the runs of hill-climbing as shown in Table 6.4.

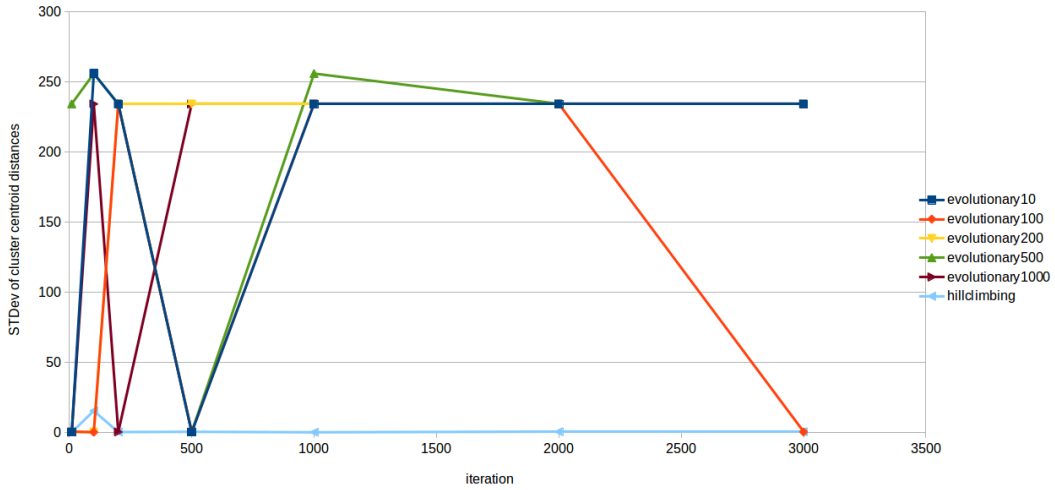


Figure 6.9. Standard deviation of the distances between cluster centers

Figure 6.10 shows Figure 6.9 in a zoomed view. The dashed line represents the clustering result with default weights. Thus it appears that the standard deviation of the hill-climbing after a generation size of 200 stays between 0 and 1 and it is always lower than the standard deviation of the evolutionary approaches. However the standard deviation of hill-climbing is quite similar to the baseline and it indicates, that the hill-climbing optimization does not improve the clustering result significantly.

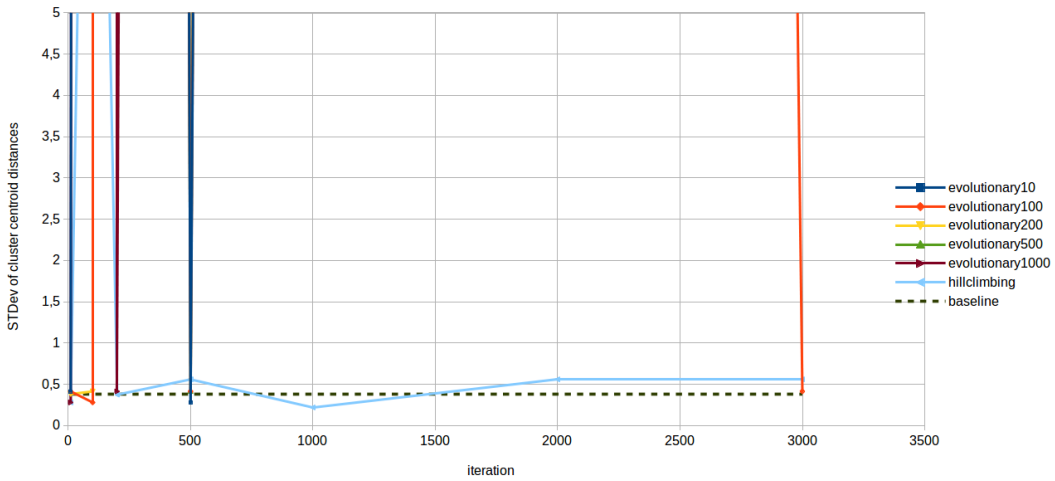


Figure 6.10. Zoomed view: Standard deviation of the distances between cluster centers

Table 6.5 and Table 6.3 show the resulting optimized weights of hill-climbing and evolutionary optimization. The weights are normed to the average of 1, thus they are comparable. In Table 6.3 it is remarkable that many weights are set to a very low value. It is interesting to note that the runs which provide a good result, does not follow the recommendations of the sensitivity analysis.

Preconclusion

Against all expectations, hill-climbing performs better than evolutionary optimization an provides good and stable results after 200 iterations. Whereas hill-climbing has a better standard deviation of cluster center distances, evolutionary optimization has a lower sum of squared errors. Compared to the measured baseline of sensitivity analysis both algorithms does not improve the values significantly. We assume, that the results would perform better, if the optimizers would also have some knowledge about the monitored application, for instance in the form of a reference cluster. To validate this assumption, it should be necessary to execute the same experiment using a reference cluster (Section 4.1) and carry out a user study with the human-in-the-loop approach. The study should

elaborate the improvement of the human-in-the-loop approach and the influence of the user on the optimized weights. However it is very expensive to create a reference cluster out of raw performance problem instances. Thus we recommend to use the human-in-the loop approach, which optimizes the weights iteratively, without the need of any reference cluster.

Table 6.3. Hill-climbing: Optimized Weights depending on the number of iterations (N: Number of clusters)

Iteration	N	Weights					
10	6	1,272	0,282	2,600	0,179	1,655	0,012
100	8	4,345	0,068	0,496	8,505E-5	0,198	0,893
200	3	1,366	0,000	0,635	3,665	0,320	0,013
500	3	5,744	1,837E-7	2,371E-7	1,795E-7	0,237	0,019
1000	3	1,975	0,205	0,059	3,760	0,000	0,000
2000	3	7,097E-9	5,291E-15	1,107E-5	3,749	2,251	1,229E-9
3000	3	1,395E-13	1,957E-10	6,964E-13	2,261E-5	8,341E-24	6,000

Table 6.4. HillClimbing: Internal quality statistics depending on number of iterations (SSE: Sum of Squared Errors, AvDist: Average of the centroid distances, MinDist: Shortest distance of clustering result, MaxDist: Biggest distance of clustering result, STDevDist: Standard Deviation of all distances)

Iteration	N	SSE	AvDist	MinDist	MaxDist	STDevDist
10	6	62,065	1,440	0,539	1,744	0,279
100	8	63,217	20,255	1,416	63,766	15,501
200	3	253,012	1,391	1,000	1,743	0,373
500	3	255,577	0,678	0,033	1,001	0,558
1000	3	255,355	1,305	1,055	1,447	0,218
2000	3	255,577	0,678	0,033	1,001	0,558
3000	3	255,577	0,678	0,033	1,001	0,558

Table 6.5. Evolutionary Optimization: Optimized Weights depending on the generation size and population size (N: Number of clusters)

Generation	Population	N	Weights					
10	10	6	1,203	0,852	1,667	0,940	0,637	0,701
100	10	7	1,451	0,693	0,014	1,437	1,036	1,368
200	10	6	1,289	0,738	0,487	1,145	1,333	1,008
500	10	6	1,525	1,101	0,306	1,104	1,449	0,515
1000	10	6	1,209	1,048	0,958	1,105	1,281	0,399
2000	10	6	1,437	0,838	0,658	1,218	0,876	0,974
3000	10	6	1,406	0,390	1,420	1,363	1,161	0,260
10	100	6	2,019	0,551	0,035	1,971	0,954	0,470
100	100	6	0,979	0,824	1,033	0,873	1,042	1,249
200	100	6	1,223	0,893	1,015	1,207	0,848	0,814
500	100	6	1,631	1,268	0,849	1,334	0,871	0,047
1000	100	6	1,387	1,046	0,183	1,212	0,797	1,376
2000	100	6	1,252	0,919	0,851	1,006	1,268	0,705
3000	100	6	1,773	1,294	0,036	1,488	1,126	0,284
10	200	6	1,200	0,448	0,856	1,198	0,964	1,334
100	200	6	1,270	0,965	1,409	1,105	0,752	0,499
200	200	6	1,646	1,097	0,058	1,321	1,446	0,432
500	200	6	1,120	0,782	1,020	0,980	1,058	1,039
1000	200	6	1,304	1,077	1,321	1,127	0,897	0,274
2000	200	6	1,237	1,005	0,931	1,029	1,134	0,663
3000	200	6	1,448	1,154	0,000	1,205	1,475	0,719
10	500	6	1,108	0,980	1,318	1,041	0,845	0,708
100	500	7	1,273	0,991	0,019	1,228	0,934	1,555
200	500	6	1,820	0,651	0,073	1,735	0,675	1,046
500	500	6	1,332	0,998	0,584	1,082	0,748	1,256
1000	500	7	1,120	0,778	0,490	1,058	1,101	1,452
2000	500	6	1,404	0,955	0,517	1,141	1,315	0,668
3000	500	6	1,283	1,069	1,079	1,153	1,358	0,058
10	1000	6	1,001	0,983	1,021	0,988	0,460	1,548
100	1000	6	1,152	0,831	0,688	0,917	1,211	1,202
200	1000	6	1,461	1,202	1,251	1,203	0,852	0,032
500	1000	6	1,369	1,117	0,393	1,245	1,386	0,489
1000	1000	6	1,340	0,981	1,134	1,163	1,171	0,211
2000	1000	6	1,261	0,940	0,705	0,960	1,218	0,917
3000	1000	6	1,336	0,904	1,334	1,103	0,734	0,589

Table 6.6. Evolutionary Optimization: Internal quality statistics depending on the populations size and generation size (Gen: generation size, Pop: population size, SSE: Sum of Squared Errors, AvDist: Average of the centroid distances, MinDist: Shortest distance of clustering result, MaxDist: Biggest distance of clustering result, STDevDist: Standard Deviation of all distances)

Gen	Pop	N	SSE	AvDist	MinDist	MaxDist	STDevDist
10	10	6	73,943	1,283	0,045	1,487	0,413
100	10	7	74,166	427,049	18,561	955,035	256,042
200	10	6	70,954	306,312	38,526	767,886	234,343
500	10	6	62,065	1,440	0,539	1,744	0,279
1000	10	6	70,954	306,312	38,526	767,886	234,343
2000	10	6	70,954	306,312	38,526	767,886	234,343
3000	10	6	70,954	306,312	38,526	767,886	234,343
10	100	6	73,943	1,283	0,045	1,487	0,413
100	100	6	62,065	1,440	0,539	1,744	0,279
200	100	6	70,954	306,312	38,526	767,886	234,343
500	100	6	73,943	1,283	0,045	1,487	0,413
1000	100	6	70,954	306,312	38,526	767,886	234,343
2000	100	6	70,954	306,312	38,526	767,886	234,343
3000	100	6	73,943	1,283	0,045	1,487	0,413
10	200	6	70,954	1,550	0,540	2,031	0,380
100	200	6	73,943	1,283	0,045	1,487	0,413
200	200	6	70,954	306,312	38,526	767,886	234,343
500	200	6	70,954	306,312	38,526	767,886	234,343
1000	200	6	70,954	306,312	38,526	767,886	234,343
2000	200	6	70,954	306,312	38,526	767,886	234,343
3000	200	6	70,954	306,312	38,526	767,886	234,343
10	500	6	70,954	306,312	38,526	767,886	234,343
100	500	7	74,166	427,049	18,561	955,035	256,042
200	500	6	70,954	306,312	38,526	767,886	234,343
500	500	6	73,943	1,283	0,045	1,487	0,413
1000	500	7	74,166	427,049	18,561	955,035	256,042
2000	500	6	70,954	306,312	38,526	767,886	234,343
3000	500	6	70,954	306,312	38,526	767,886	234,343
10	1000	6	62,065	1,440	0,539	1,744	0,279
100	1000	6	70,954	306,312	38,526	767,886	234,343
200	1000	6	73,943	1,283	0,045	1,487	0,413
500	1000	6	70,954	306,312	38,526	767,886	234,343
1000	1000	6	70,954	306,312	38,526	767,886	234,343
2000	1000	6	70,954	306,312	38,526	767,886	234,343
3000	1000	6	70,954	306,312	38,526	767,886	234,343

6.4 Scalability of the cluster engine

During the implementation phase, it was noticeable, that with a high number of problem instances, the hierarchical clustering has a scalability problem. For instance, if the cluster engine runs on more than 10000 problem instances, the cluster engine does not terminate correctly, it is stop by a heap space error. The following experiment analyzes the root cause of the occurred error.

6.4.1 Research question

The experiment should answer the following question:

- Which component of the implementation causes the Java Heap Space error?

6.4.2 Hypothesis

We assume, that the Java Heap Space error is caused by the Weka framework.

6.4.3 Experimental design

The cluster engine using hierarchical clustering is executed with 10000 random generated performance problem instances. The number of 10000 performance problem instances guarantees, that the JVM runs into a Java Heap Space error. Meanwhile, a profiler, called VisualVM analyzes the current objects of the heap.

6.4.4 Evaluation of the experiment

With a number of 10000 performance problem instances, the current implementation fails caused by a Java Heap Space error. There are to much objects in the heap. With the profiler VisualVM, we analyzed the heap usage: 99,6% of all objects are HierarchicalClusterer-tuples (see Figure 6.11).

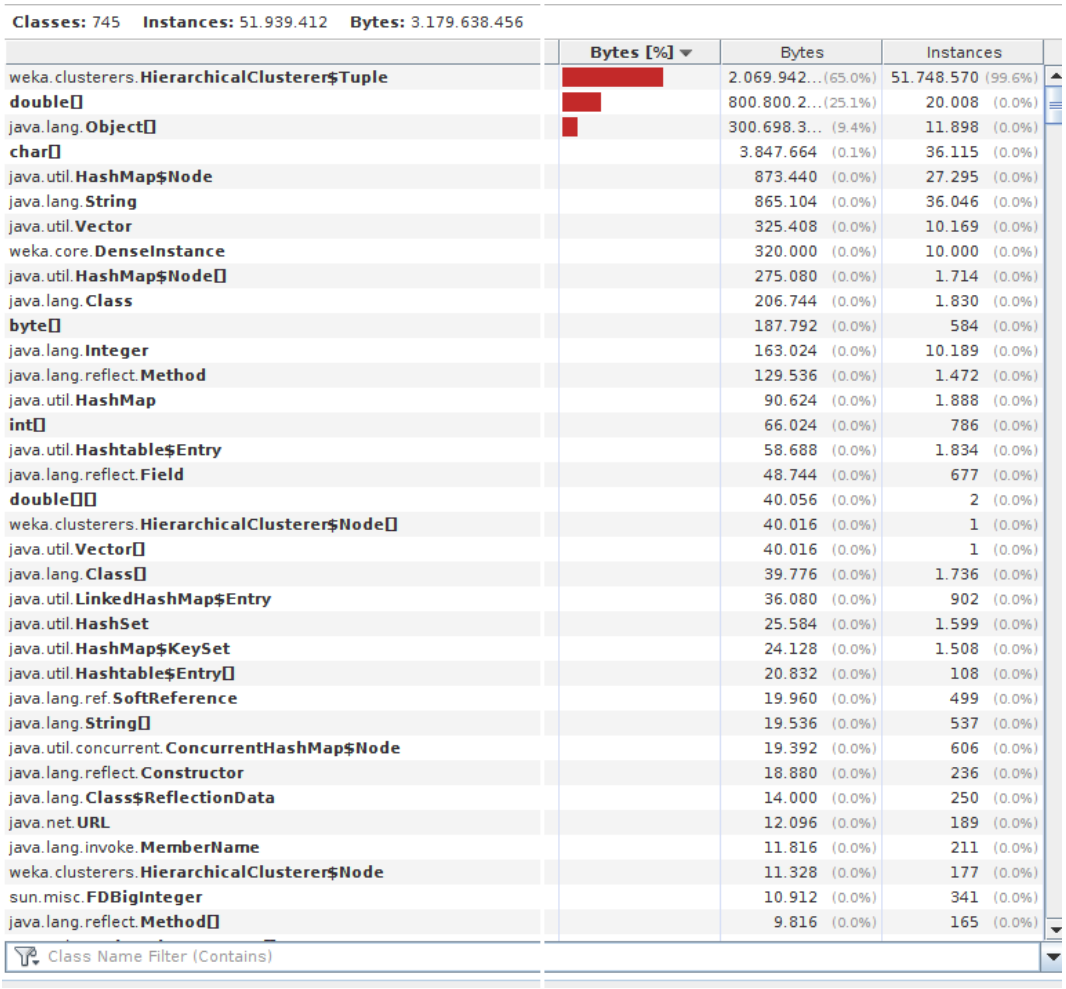


Figure 6.11. Heap usage of the ClusterEngine

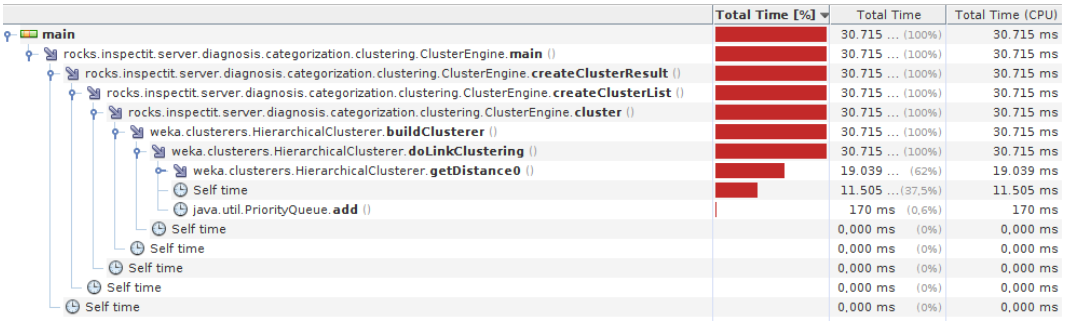


Figure 6.12. CPU usage of the ClusterEngine

The Weka framework, which is responsible for the cluster algorithm, creates the objects. In addition, we analyzed the CPU usage of the ClusterEngine and we noticed, that the method *getDistance0()* of the Cluster result uses 35% of the whole CPU time (See Figure 6.12).

As conclusion, the Weka framework limits the scalability of the hierarchical clustering. The code implemented as part of the thesis does not cause the scalability problem.

6.5 Survey reference cluster

To determine the external quality of a cluster, we need an reference cluster (Section 2.1). We compare one single generated cluster solution with the reference cluster and rate the current solution. The reference cluster has to be generated manually. Thus, we started a survey to find a reference, which is clustered by APM experts.

6.5.1 Goal

The goal of the survey is to elaborate an optimal reference clustering, which is made up of single clustering results of each participant. Thus, the reference cluster is built based on the experience of the APM experts. The APM experts instinctively put on a weight on the different attributes and focusing on the more important attributes. Our aim is to conserve the experience of the APM experts in the reference cluster.

6.5.2 Research question

Apart from elaborate a reference cluster, the study should answer the following research question:

- Is there a correlation between the duration of the experiment and the number of clusters?
- Is there a correlation between the years of experience and the number of clusters?

6.5.3 Hypothesis

We assume, that the number of experienced years influences the number of clusters. In addition we assume, that the more time a participant spends on the experiment the more clusters will have the clustering result.

6.5.4 Experimental design

Instead of performance problem instances, we created twenty invocation sequences and asked ten APM experts to cluster them by hand, because most of the APM experts are more familiar to deal with invocation sequences. We created the invocations sequences with the DVD store application [1]. Each invocation sequence consists of one or more performance problem instances. The DVD store application provides the possibility to switch on performance problems. Four different scenarios are provided. Each participant was free to choose the number of clusters. It was possible to put each invocation sequence into a single cluster. We informed each participant, that the estimated time will be about half an hour. That guaranteed, that each participant conforms with the estimated time and the participants are not stressed. With a standard deviation of 9,25 min the results are comparable.

Control questions

To classify results, each APM expert had to answer to following questions:

- How many years are you working in the field of Application Performance Management?
- How much minutes did you spent to execute the task?
- In which environment do you work?
- What is your role in this environment?

The questions are used to evaluate the different clustering results. Based on the answers of the questions, we can categorize the results due to the experience, the effort and the environment of the participants [11].

6.5.5 Evaluation of the results

Based on the answers of the control questions, we made a statistical evaluation. The similarity of the results and the background of the participants are analyzed. On an average, the participants needed 27,6 minutes for the task and already worked for 5,4 years in the context of application performance management.

Environment of the participants

It stands out, that most of the participants are working in the context of consulting. Over 44 percent are also working in the context of industry and university.

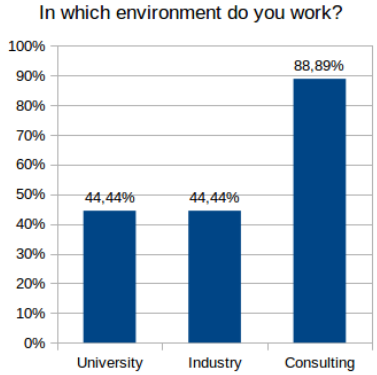


Figure 6.13. Environment deviation of the participants

Role of the participants

In evidence, most of the participants are working as performance engineer, which confirms, that mostly experts participated at this study. 44,44 % of the participants are also working as researcher.

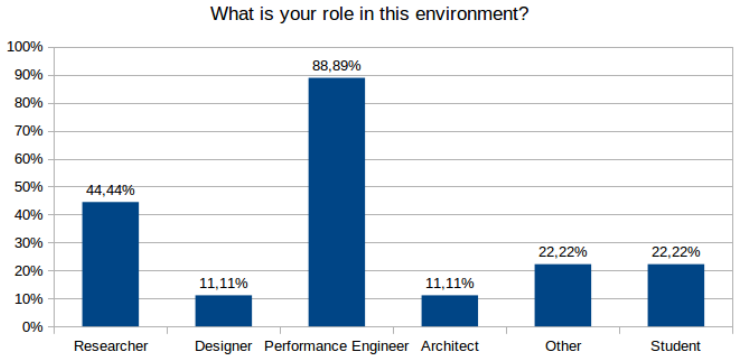


Figure 6.14. The role deviation of the participants

Number of Clusters

It is noticeable, that there is a huge diversity of results. One third of the participants detected three clusters and another third detected four clusters.

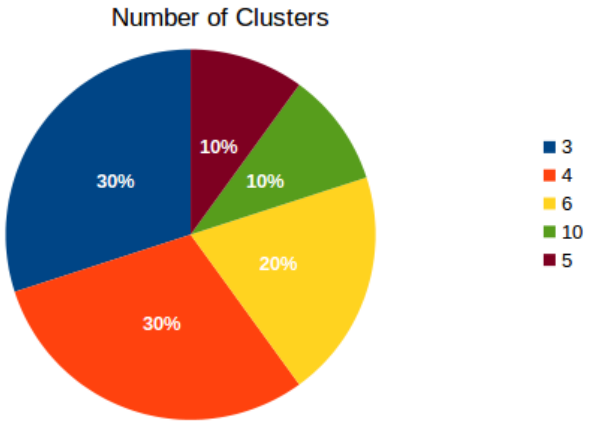


Figure 6.15. The role deviation of the participants

Correlation between the number of clusters and the duration of the experiment

Figure 6.16 shows the number of clusters depending on the duration of the experiment. It is obvious that there is no correlation between the number of clusters and the duration of the experiment. This indicates, that the hypothesis is disproved.

Correlation between the number of clusters and the years of experience

Figure 6.17 shows the number of clusters depending on the years of experience of the participants. It is obvious that there is no correlation between the number of clusters and the years. Thus, the hypothesis cannot be approved.

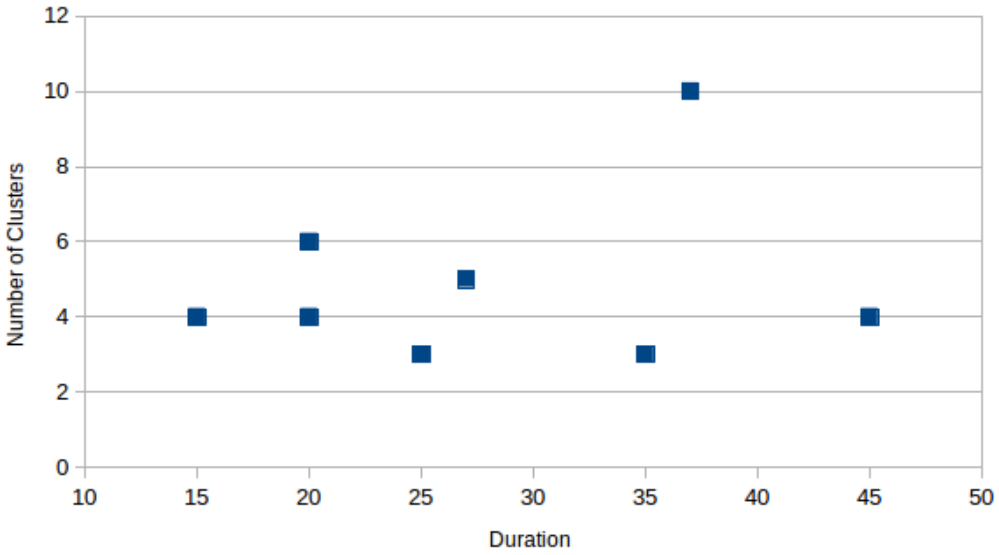


Figure 6.16. Number of clusters depending on the duration of the experiment

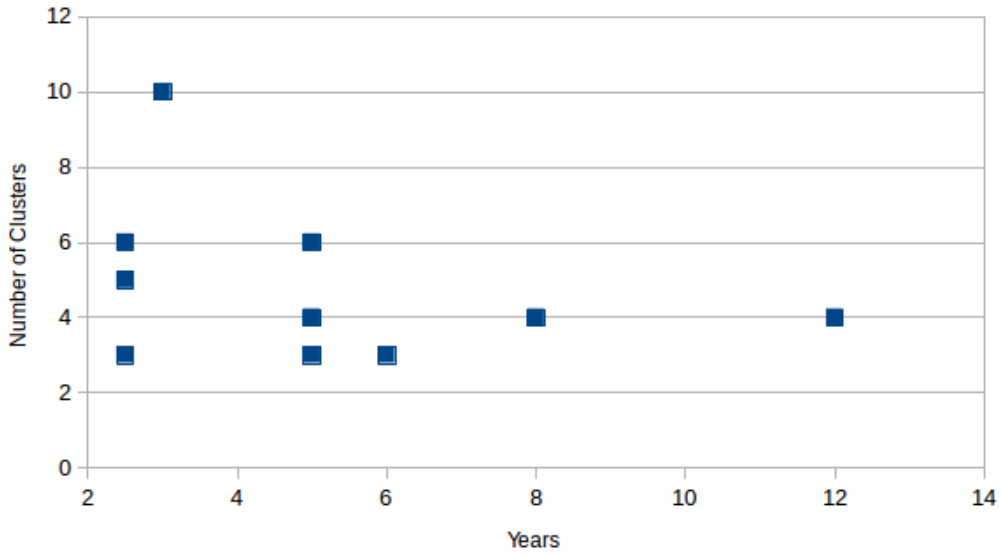


Figure 6.17. Number of clusters depending on the years of experience of the participants

Merging the clustering results

The goal of the survey was to find a representative reference cluster for the used performance problem instances. However, against all our expectations, the results are very diverse and even results with the same number of clusters have only a few similarities. The participants used a different detail degree and weighted the provided information different. Based on the external quality function (Section 4.1) we modeled the following graph:

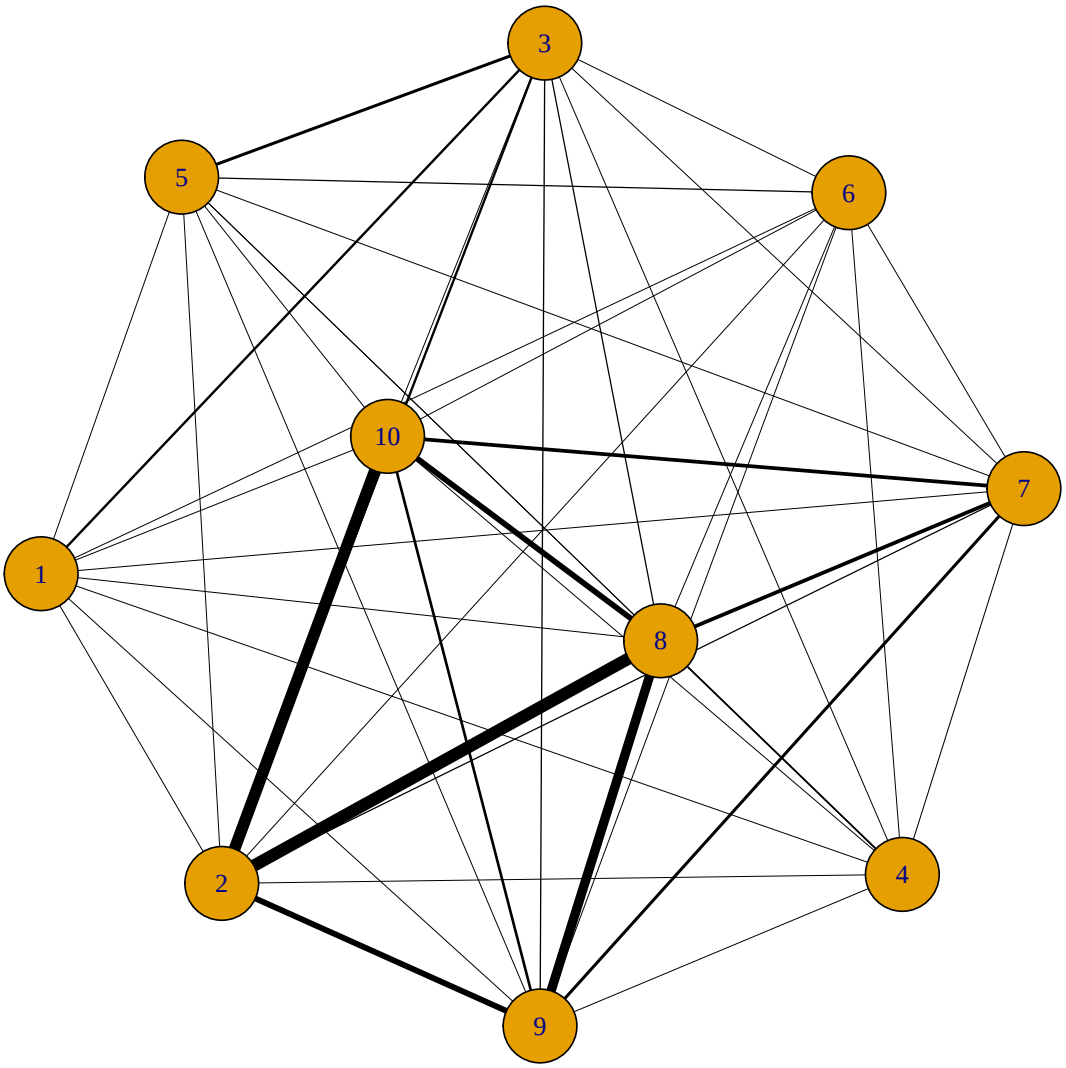


Figure 6.18. Similarity graph of reference clustering results

Each node represents a clustering result of a participant. Similar clustering results are closer together. The more similar the results are, the thicker are the edges in between. As represented in the graph the participants 2, 8, 9 and 10 submitted solutions, which are closer together than other solutions. However it is not possible to merge the most similar solutions, because they are not close enough to guarantee a meaningful reference cluster. For further evaluation, we use the internal quality methods (Section 4.1.2) to measure the quality of a reference cluster. The survey indicates, that it is recommended to use an optimization approach which is not dependent on a reference cluster.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The goal of this thesis was to elaborate and evaluate a concept of automated categorization of performance problem instances. The instances are provided by *diagnoseIT*. The automated categorization is implemented with a cluster algorithm. The cluster algorithm categorizes the performance problem instances based on the attribute of each single problem instance. Instances, which have similarities, will be in the same cluster. As a consequence, the performance analyst gets a proper overview about the problem categories of the monitored software.

In the second part of the thesis we focused on the optimization of the cluster approach. Each attribute of a performance problem instance can be weighted. Three different approaches were elaborated. The goal of the optimization approach was to provide optimized weights, which are appropriate for the present monitored application. In the evaluation, we compared hierarchical clustering with k-means clustering and determined, that k-means is better than hierarchical clustering in terms of accuracy and performance. However, hierarchical clustering provides a hierarchical structure of the problem categories, which can be very useful in some cases.

It turned out, that there is potential for optimization, however in our experiments the tested automated optimization approaches hill-climbing and evolutionary optimization did not provide significant better results than the default weights using internal quality criteria (Section 4.1.1). For a better optimization, it is recommended to include some knowledge about the monitored system into the optimization process.

7.2 Future Work

Within this thesis we could not answer all questions and test all possibilities. The following topics might be interesting to explore in depth:

- New Attributes: Beside the attributes we are using currently, there are many other properties of an execution trace, which can be transformed into an attribute. For instance, it could be worth it to compare the tree structure of the corresponding execution trace.
- Iterative Optimization: Instead of an optimization engine, which has to be triggered

manually, the cluster engine optimizes itself iteratively.

- Accuracy of the human clustered reference cluster: It could be interesting to compare the different cluster solutions of different performance analysts. How stable is a result of the performance analysts and which properties of the problem instances have a influence on the heterogeneity of the results.
- GUI: visual inspection of the cluster tree: The tree structure, which we use to process the data can be also presented to the user. With the tree, we provide more information about the occurred problems and similarities of different problems can be seen much faster.

Bibliography

[1] DVDStore. https://inspectit-performance.atlassian.net/wiki/display/DVDStore. Accessed: 2016-10-25.

[2] JMH Benchmarking. http://openjdk.java.net/projects/code-tools/jmh/. Accessed: 2016-10-25.

[3] Rapidminer. http://docs.rapidminer.com/studio/getting-started/. Accessed: 2016-10-25.

[4] diagnoseIT Research Project. https://diagnoseit.github.io/. Accessed: 2016-10-25.

[5] inspectIT. http://www.inspectit.rocks/, . Accessed: 2016-10-25.

[6] inspectITarchitecture. https://inspectit-performance.atlassian.net/wiki/display/DOC16?preview=/5018625/5018663/inspectit-overview.png, . Accessed: 2016-10-25.

[7] A. Abraham and L. Jain. *Evolutionary multiobjective optimization*. Springer, 2005.

[8] H. De Beukelaer, G. F. Davenport, G. De Meyer, and V. Fack. James: A modern object-oriented java framework for discrete optimization using local search metaheuristics. In *4th International symposium and 26th National conference on Operational Research*, pages 134–138. Hellenic Operational Research Society, 2015.

[9] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009. ISSN 1931-0145.

[10] D. Hamby. A review of techniques for parameter sensitivity analysis of environmental models. *Environmental monitoring and assessment*, 32(2):135–154, 1994.

[11] C. Heger. *An Approach for Guiding Developers to Performance and Scalability Solutions*. PhD thesis, Karlsruhe, Karlsruher Institut für Technologie (KIT), Diss., 2015, 2015.

[12] C. Heger, A. van Hoorn, D. Okanović, S. Siegl, and A. Wert. Expert-guided automatic diagnosis of performance problems in enterprise applications. 2016.

[13] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM computing surveys (CSUR)*, 31(3):264–323, 1999.

[14] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich. Opt4j: A modular framework for meta-heuristic optimization. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1723–1730, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0557-0.

[15] D. Okanović, A. van Hoorn, C. Heger, A. Wert, and S. Siegl. Towards performance tooling interoperability: An open format for representing execution traces. In *European Workshop on Performance Engineering*, pages 94–108. Springer, 2016.

[16] R. L. Plackett and J. P. Burman. The design of optimum multifactorial experiments. *Biometrika*, 33(4):305–325, 1946.

[17] K. Project. Kieker user guide. Research report, April 2015. URL `http://eprints.uni-kiel.de/16537/`.

[18] C. Reilly, C. Wang, and M. Rutherford. A rapid method for the comparison of cluster analyses. *Statistica Sinica*, pages 19–33, 2005.

[19] J. Renegar. Is it possible to know a problem instance is ill-posed?: some foundations for a general theory of condition numbers. *Journal of Complexity*, 10(1):1–56, 1994.

[20] L. Rokach and O. Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[21] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[22] C. U. Smith and L. G. Williams. Performance solutions: a practical guide to creating responsive, scalable software. 2001.

[23] A. J. Viera, J. M. Garrett, et al. Understanding interobserver agreement: the kappa statistic. *Fam Med*, 37(5):360–363, 2005.

Appendices

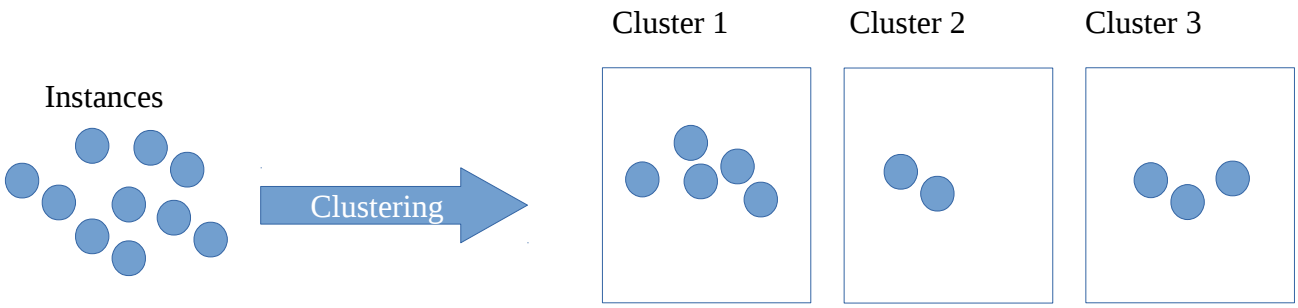
.1 Reference cluster study

This section shows the corresponding document of the study for getting a reference cluster. The document introduces the topic of clustering performance problem instances and guide the participant through the GUI of the inspectIT client. It defines the task and describes the conditions. Afterwards, the participant is asked to answer questions.

Survey – Reference cluster

Dear participant,

Thank you for joining this survey. The survey consists of a task and four questions. Before you execute the task, please read the following sections as they will provide you the background and rational of the survey. The survey is conducted as part of my Bachelor Thesis on categorizing performance problem instances. Your responses are used to evaluate different optimization approaches.



The goal of the survey is to derive a reference cluster for given set of performance problem instances. The reference cluster is used to determine the goodness of the cluster solutions computed by the clustering algorithms under test. Your task is to cluster 20 invocation sequences provided by inspectIT. The remainder of this document guides you through the clustering tasks. In case of any questions, please contact me through tobias.angerstein@novatec-gmbh.de.

Task:

Please measure the time how long it takes to cluster the instances! If you have finished the task, please sent your results to tobias.angerstein@novatec-gmbh.de including the PDF with the answered questions.

Step 1:

Download the correct inspectIT client for your operating-system. Please only use the versions, provided by the following link:

Password: apm
https://cloud.novatec-gmbh.de/index.php/s/X2RHRYeA802R2EN

Step 2:

Unzip the folder and run the “inspectIT” binary.

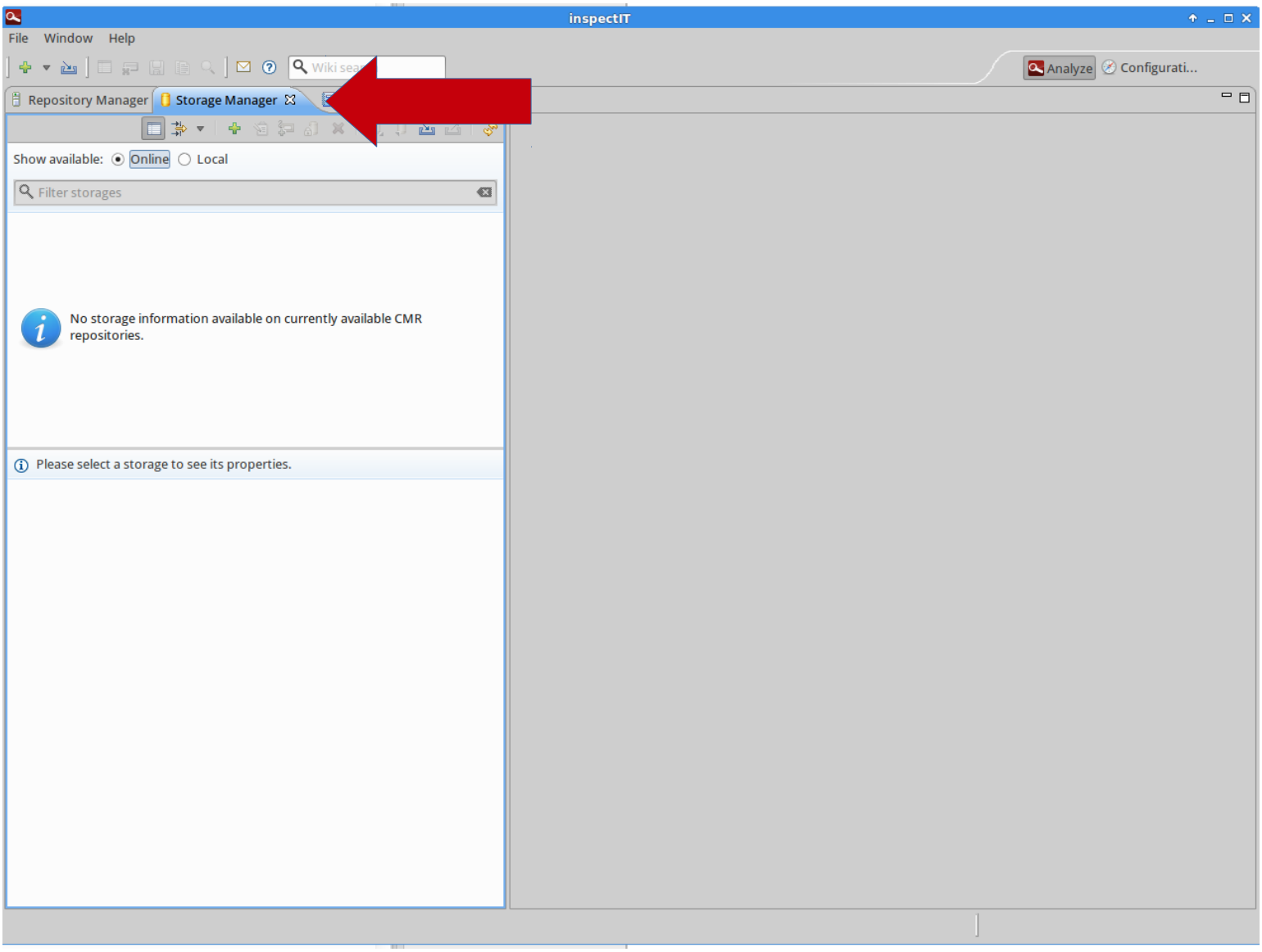
Step 3:

Download the storage file “*InvocationSequences.itds*”.

Password: apm
https://cloud.novatec-gmbh.de/index.php/s/X2RHRYeA802R2EN

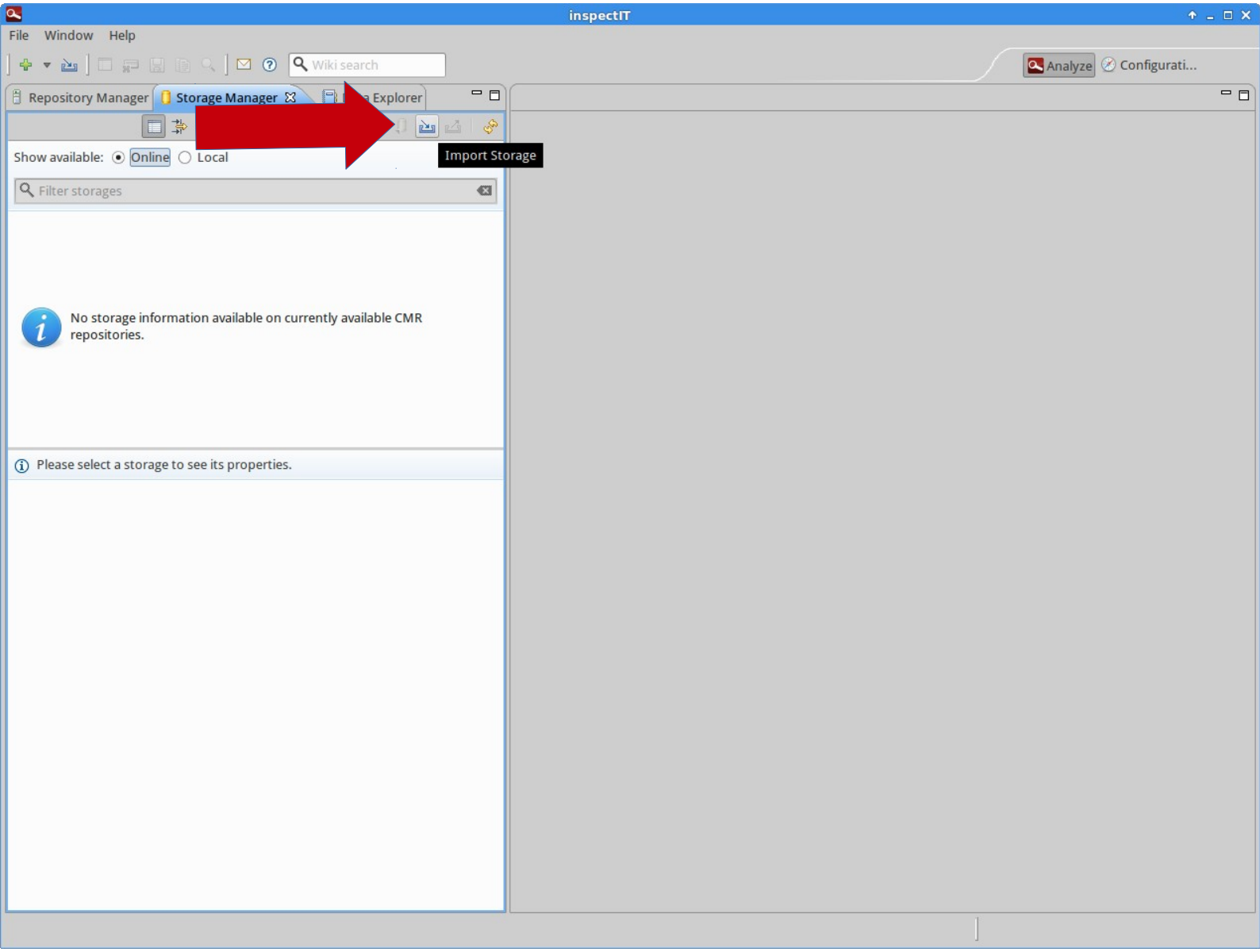
Step 4:

Switch to the inspectIT client and goto the tab “Storage Manager”:



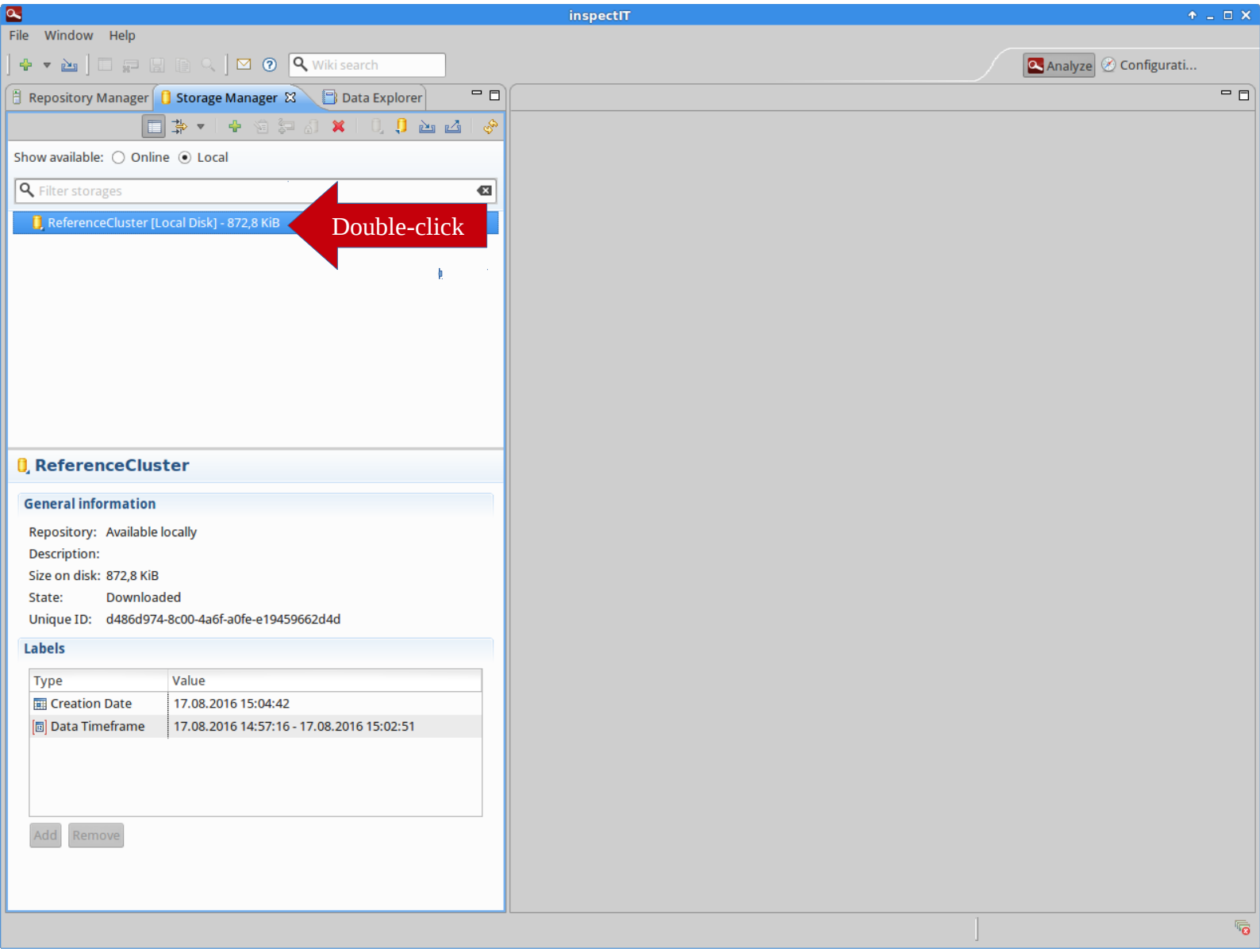
Step 5:

Click on “Import Storage” and choose the file “*InvocationSequences.itds*”.



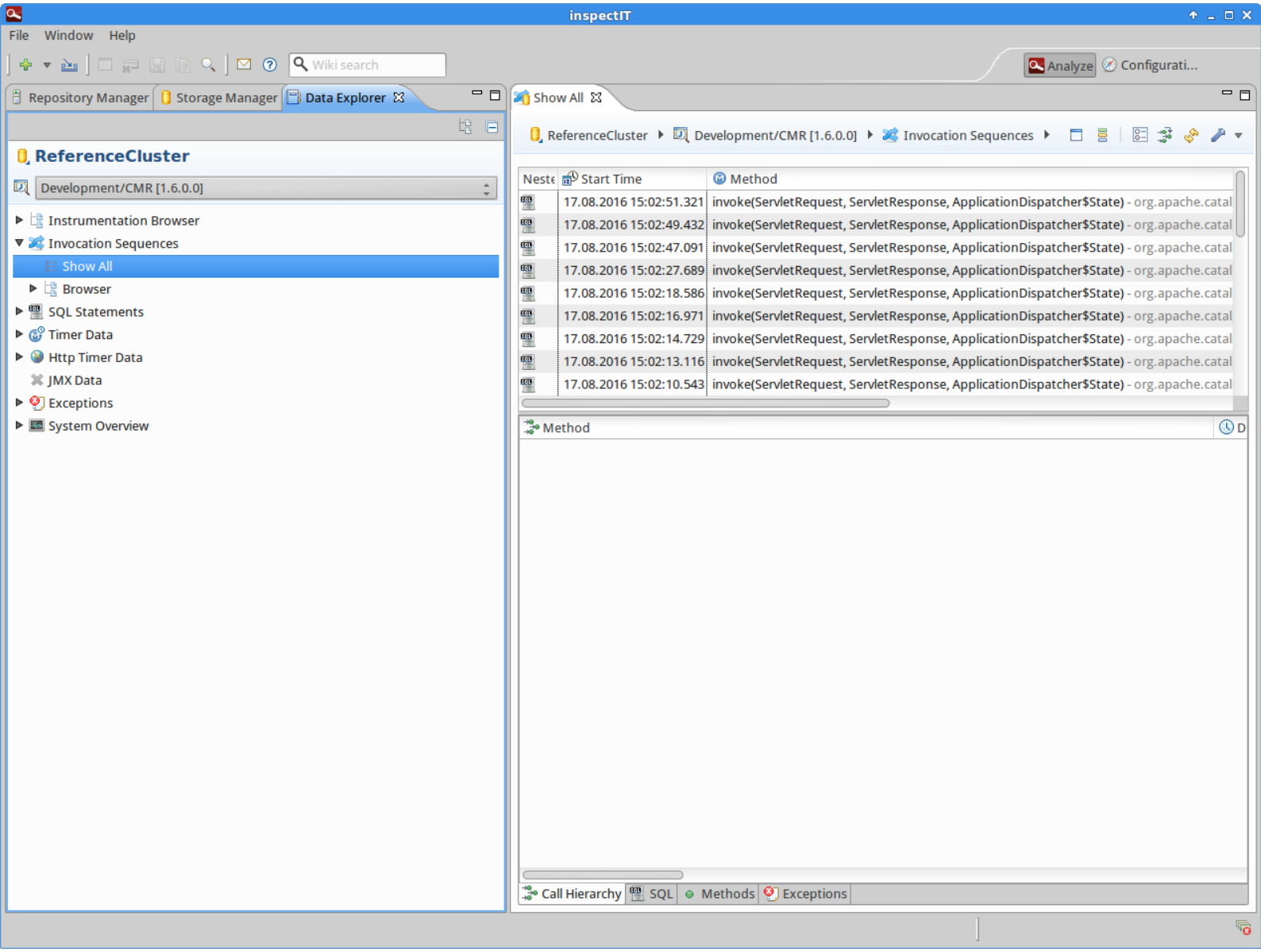
Step 6:

If the import was successful, the Storage is visible. After double-clicking on the ReferenceCluster the measured data is visible.



Step 7:

Open the tab “Invocation Sequences” and choose “Show all”. You will see 20 invocation sequences.



Step 8: Categorization

Inspect each invocation sequence and try to find similarities and differences. Your task is to categorize the invocation sequences. Sequences with similar parameters should be in the same cluster. For instance, you can use the call tree and the response times to compare the different invocation sequences.
You are free to create any possible solution, the number of clusters is not limited.

Please insert your result in the attached excel sheet and use the timestamp of an invocation sequence as reference. Please choose the right timestamp in the drop down menu of the excel sheet.

Questions:

1. How many years are you working in the field of Application Performance Management?

2. How much minutes did you spent to execute the task?

3. In which environment do you work? (Multiple answers are possible)

- [ ] University
- [ ] Industry
- [ ] Consulting
- [ ] Other:

4. What is your role in this environment? (Multiple answers are possible)

- [ ] Manager
- [ ] Software Engineer/Developer
- [ ] Quality Engineer/Tester
- [ ] Researcher
- [ ] Designer
- [ ] Performance Engineer
- [ ] Architect
- [ ] Other:

Thanks for your participation!

Declaration

I declare that this thesis is the solely effort of the author. I did not use any other sources and references than the listed ones. I have marked all contained direct or indirect statements from other sources as such. Neither this work nor significant parts of it were part of another review process. I did not publish this work partially or completely yet. The electronic copy is consistent with all submitted copies.

---

place, date, signature