ΣΥΝΟΨΕΙΣ ΤΗΣ ΒΙΟΓΡΑΦΙΑΣ ΕΞΕΛΙΣΣΟΜΕΝΩΝ ΣΧΗΜΑΤΩΝ ΒΑΣΕΩΝ ΔΕΔΟΜΕΝΩΝ

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Μηχανικών Η/Υ και Πληροφορικής
Εξεταστική Επιτροπή

από τον

Γιάχο Θεοφάνη

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Οκτώβριος 2015

ACKNOWLEDMENTS

I would like to express my gratitude to my supervisor, Associate Professor Panos Vassiliadis for his immense guidance and assistance in every difficulty that I met in this Thesis.

I am also grateful to my family that it is alongside me in every step of my life. Also I could not forget my friends and fellow students Maria Spai, Michael Kolozoff and Kostas Noulis for all the great moments that we had. Special thanks to my friends Vasilis Spais and Dimitris Moustos too.

To those people that I have met in my life,
and helped me to be a better man…

TABLE OF CONTENTS

	Page
ACKNOWLEDMENTS	i
TABLE OF CONTENTS	ii
LIST OF TABLES	iv
LIST OF FIGURES	v
ABSTRACT	vii
ΕΚΤΕΤΑΜΕΝΗ ΠΕΡΙΛΗΨΗ ΣΤΑ ΕΛΛΗΝΙΚΑ	ix
CHAPTER 1. Introduction	1
1.1 Thesis Scope	1
1.2 Thesis Structure	4
CHAPTER 2. Fundamentals	7
2.1 Definitions of main concepts	7
2.2 Identified Changes per Transition of each Relation	8
2.3 Matrix Representation of Database Evolution	11
2.4 Visual representation of a history of a database	12
CHAPTER 3. Problem Specification	17
3.1 Segmentation of the history into phases	17
3.2 Clustering of tables into groups	18
3.3 Zoom into a specific point of the overview	18
3.4 Filter the overview	19
3.5 Details on demand	19
CHAPTER 4. Creating an overview of the history of a schema	21
4.1 Computing a segmentation of the history into phases	21
4.1.1 Parameters	23
4.1.2 Distance Function	23
4.1.3 Assessment of the method	25
4.2 Grouping tables into clusters	35
4.2.1 Parameters	37
4.2.2 Distance Function	37
4.2.3 Assessment of the method	39
4.3 Zoom into a specific point of overview	61
4.4 Filter the overview of the history of a relational database schema	63
4.5 Details on demand	66
CHAPTER 5. Software aspects of our solution – the PPL tool	71
5.1 Design and Analysis	71
5.1.1 The data Package	72
5.1.2 The phaseAnalyzer Package	77
5.1.3 The tableClustering package	82

5.1.4 The gui package 85
5.2 Implementation 87
5.2.1 Programming Tools and IDEs 87
5.2.2 Selected Classes 88
5.2.3 PPL Tool Screenshots 92
CHAPTER 6. Related work 101
6.1 Empirical Studies of software and schema evolution 101
6.2 Timeseries Segmentation 104
6.3 Data Visualization 106
CHAPTER 7. conclusions and open issues 109
REFERENCES 111
APPENDIX 113
Metrics of change for the database level 113
Assessment of phase extraction 114
Assessment of table clustering 149

LIST OF TABLES

Table	Page
Table 2-1: Types of changes that occur to a relation R, during a transition t	12
Table 4-1: Explanation of the distance function	24
Table 4-2: Number of wins for different sets of parameters	28
Table 4-3: Explanation of the distance function	38
Table 4-4: Results for wb: 0.333 , wd: 0.333 , wc: 0.333	44
Table 4-5: Results for wb:0.0 wd:1.0 wc:0.0	45
Table 4-6: Results for wb:0.0 wd:0.5 wc:0.5	46
Table 4-7: Results for wb:0.0 wd:0.0 wc:1.0	47
Table 4-8: Results for wb:0.5 wd:0.5 wc:0.0	48
Table 4-9: Results for wb:0.5 wd:0.0 wc:0.5	49
Table 4-10: Results for wb:1.0 wd:0.0 wc:0.0	50
Table 4-11: Average F-Measure	51
Table 4-12: Atlas results	52
Table 4-13: bioSQL results	52
Table 4-14: Coppermine results	53
Table 4-15: phpBB results	53
Table 4-16: Atlas Dataset Results	56
Table 4-17: Coppermine Dataset	57
Table 4-18: bioSQL Dataset	57
Table 4-19: Ensembl Dataset	58
Table 4-20: mwiki Dataset	58
Table 4-21: Opencart Dataset	59
Table 4-22: phpBB Dataset	59
Table 4-23: typo3 Dataset	60
Table 4-24: Number of wins for different sets of parameters	61
Table A- 1: Assessment of phase extraction for Atlas	114
Table A- 2: Assessment of phase extraction for bioSQL	119
Table A- 3: Assessment of phase extraction for Ensembl	124
Table A- 4: Assessment of phase extraction for mediaWiki	129
Table A- 5: Assessment of phase extraction for Opencart	134
Table A- 6: Assessment of phase extraction for phpBB	139
Table A- 7: Assessment of phase extraction for Typo3	144
Table A- 8: Assessment of table clustering for Atlas	149
Table A- 9: Assessment of table clustering for phpBB	156
Table A- 10: Assessment of table clustering for Coppermine	163

LIST OF FIGURES

Figure	Page
Figure 2.1: A part of Ensembl's PPL diagram	14
Figure 4.1: Atlas Dataset	26
Figure 4.2: bioSQL Dataset	26
Figure 4.3: Coppermine Dataset	26
Figure 4.4: Ensembl Dataset	26
Figure 4.5: mediaWiki Dataset	27
Figure 4.6: Opencart Dataset	27
Figure 4.7: phpBB Dataset	27
Figure 4.8: typo3 Dataset	27
Figure 4.9: (δtime, δc) for (WC:0.0, WT:1.0, PPC:OFF, PPT:OFF)	29
Figure 4.10: (δtime, δc) for (WC:0.0, WT:1.0, PPC:ON, PPT:OFF)	30
Figure 4.11: (δtime, δc) for (WC:0.0, WT:1.0, PPC:OFF, PPT:ON)	30
Figure 4.12: (δtime, δc) for (WC:0.0, WT:1.0, PPC:ON, PPT:ON)	31
Figure 4.13: (δtime, δc) for (WC:0.5, WT:0.5, PPC:OFF, PPT:OFF)	31
Figure 4.14: (δtime, δc) for (WC:0.5, WT:0.5, PPC:ON, PPT:OFF)	32
Figure 4.15: (δtime, δc) for (WC:0.5, WT:0.5, PPC:OFF, PPT:ON)	32
Figure 4.16: (δtime, δc) for (WC:0.5, WT:0.5, PPC:ON, PPT:ON)	33
Figure 4.17: (δtime, δc) for (WC:1.0, WT:0.0, PPC:OFF, PPT:OFF)	33
Figure 4.18: (δtime, δc) for (WC:1.0, WT:0.0, PPC:ON, PPT:OFF)	34
Figure 4.19: (δtime, δc) for (WC:1.0, WT:0.0, PPC:OFF, PPT:ON)	34
Figure 4.20: (δtime, δc) for (WC:1.0, WT:0.0, PPC:ON, PPT:ON)	35
Figure 4.21: bioSQL dataset with classes	43
Figure 4.22: Zoom into a specific point of overview (1)	62
Figure 4.23: Zoom into a specific point of overview (2)	63
Figure 4.24: Filter by a specific phase	64
Figure 4.25: Filter by a specific table	65
Figure 4.26: Filter by specific clusters	66
Figure 4.27: Details on demand (1)	67
Figure 4.28: Details on demand (2)	68
Figure 4.29: Details on demand (3)	69
Figure 4.30: Details on demand (4)	70
Figure 5.1: data UML Diagram	72
Figure 5.2: dataProccessing UML diagram	73
Figure 5.3: dataPPL UML diagram	75
Figure 5.4: phaseAnalyzer UML diagram	77
Figure 5.5: parser UML diagram	78
Figure 5.6: commons UML diagram	79

Figure 5.7: analysis UML diagram 81
Figure 5.8: tableClustering UML diagram 82
Figure 5.9: commons UML diagram 83
Figure 5.10: analysis UML diagram 84
Figure 5.11: gui UML diagram 85
Figure 5.12: Synopsis of Atlas 93
Figure 5.13: PLD of Atlas 93
Figure 5.14: Synopsis of bioSQL 94
Figure 5.15: PLD of bioSQL 94
Figure 5.16: Synopsis of Coppermine 95
Figure 5.17: PLD of Coppermine 95
Figure 5.18: Synopsis of Ensembl 96
Figure 5.19: A part of PLD of Ensembl 96
Figure 5.20: Synopsis of mediaWiki 97
Figure 5.21: A part of PLD of mediaWiki 97
Figure 5.22: Synopsis of Opencart 98
Figure 5.23: PLD of Opencart 98
Figure 5.24: Synopsis of phpBB 99
Figure 5.25: PLD of phpBB 99
Figure 5.26: Synopsis of Typo3 100
Figure 5.27: PLD of Typo3 100

ABSTRACT

Theofanis Giachos
MSc, Department of Computer Science and Engineering
University of Ioannina, Greece
October 2015
Biography synopses for evolving relational database schemata.
Supervisor: Panos Vassiliadis

Studying the evolution of database schemata is of great importance as a change in the schema of the database (e.g., the deletion of a table or attribute) can impact semantically and syntactically the entire ecosystem of applications that are built on top of the database. The study of schema evolution entails extracting schema versions and their delta changes from software repositories, subsequently leading to the extraction of patterns and regularities. The history of a typical database can consists of hundreds of transitions from version to version and includes a potentially large number of tables that change during the life of the database.

In order to study the evolution of databases, we have devised the Parallel Lives Diagram, a detailed 2D visual representation that graphically visualizes the life of tables in parallel lines. However, this representation has to fit into a computer screen -at best- or into a much smaller window at worst. Therefore, in this Thesis, we construct a synopsis of the entire life of a database in which the set of transitions is segmented into phases and the set of tables is organized in clusters. For this purpose, we have designed two algorithms for the construction of this synopsis. The first of our algorithms segments the entire set of transitions into phases according to their similarity in terms of activity. The second of our algorithms, extracts clusters from the entire set of the tables of a database by taking into account their similarity in terms of their creation and destruction

time points as well as their change activity. Both of these algorithms are assessed with different methods to evaluate the effectiveness of the grouping methods.

Our algorithms are part of a tool that we have developed, called Plutarch's Parallel Lives, that visualizes the entire life of a database in a multi-view, interactive fashion. A central feature of the tool is the visualization of the aforementioned (a) Parallel Lives Diagram, and (b) the synopsis of the lifetime of the database. Apart from these central constructs, the tool is also supplied with features such as drilling into specific points, filtering according to various criteria or providing the user with details on demand.

ΕΚΤΕΤΑΜΕΝΗ ΠΕΡΙΛΗΨΗ ΣΤΑ ΕΛΛΗΝΙΚΑ

Γιάχος Θεοφάνης του Κωνσταντίνου και της Γεωργίας
ΜΔΕ, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων
Οκτώβριος 2015
Συνόψεις της βιογραφίας εξελισσόμενων σχημάτων βάσεων δεδομένων. Επιβλέποντας: Παναγιώτης Βασιλειάδης

Η μελέτη της εξέλιξης σχημάτων βάσεων δεδομένων είναι μεγάλης σημασίας, καθώς μία αλλαγή στο σχήμα της βάσης (π.χ., η διαγραφή ενός πίνακα ή γνωρίσματος) μπορεί να έχει επιπτώσεις τόσο σημασιολογικά όσο και συντακτικά στο υπόλοιπο οικοσύστημα εφαρμογών το οποίο είναι χτισμένο επί της βάσης. Η μελέτη της εξέλιξης ενός σχήματος συνεπάγεται την εξαγωγή επιμέρους εκδόσεων της βάσης και των μεταξύ τους διαφορών από αποθετήρια λογισμικού, αλλά και την εξαγωγή προτύπων από αυτά. Η ιστορία μιας τυπικής βάσης δεδομένων μπορεί να αποτελείται από εκατοντάδες μεταβάσεις από έκδοση σε έκδοση και περιλαμβάνει ένα διόλου ευκαταφρόνητο αριθμό από πίνακες οι οποίοι αλλάζουν κατά την διάρκεια της ζωής της.

Για να μελετήσουμε την εξέλιξη των βάσεων, επινοήσαμε το «Διάγραμμα Παράλληλων Ζωών», μία λεπτομερή διδιάστατη οπτική αναπαράσταση η οποία οπτικοποποιεί γραφικά την ζωή των πινάκων σε παράλληλες γραμμές. Ωστόσο, η αναπαράσταση αυτή είναι αναγκαίο να χωρέσει σε μία οθόνη υπολογιστή στην καλύτερη περίπτωση ή σε ένα πολύ μικρότερο παράθυρο στη χειρότερη. Επομένως, στη συγκεκριμένη διατριβή, κατασκευάζουμε μία σύνοψη της ζωής μιας βάσης δεδομένων, στην οποία το σύνολο των μεταβάσεων της από έκδοση σε έκδοση τμηματοποιείται σε φάσεις και το σύνολο των πινάκων της οργανώνεται σε συστάδες. Για τον σκοπό αυτό, σχεδιάσαμε δύο αλγορίθμους για την κατασκευή της σύνοψης. Ο πρώτος από τους δύο αλγόριθμους τμηματοποιεί το σύνολο των μεταβάσεων της βάσης σε φάσεις, σύμφωνα με την

ομοιότητά τους σε σχέση με την δραστηριότητά τους. Ο δεύτερος από τους αλγορίθμους μας, σχετίζεται με την εξαγωγή συστάδων από το σύνολο των πινάκων της βάσης, λαμβάνοντας υπόψη την ομοιότητα τους όσον αφορά τόσο τα σημεία της δημιουργίας και καταστροφής τους όσο και τις αλλαγές τους. Και οι δύο αυτοί αλγόριθμοι αξιολογούνται με διαφορετικές μεθόδους για να κριθεί η αποτελεσματικότητα των μεθόδων ομαδοποίησης.

Οι αλγόριθμοί μας αποτελούν κομμάτι ενός εργαλείου που έχουμε αναπτύξει και ονομάζεται “Παράλληλοι Βίοι του Πλούταρχου” και έχει ως αντικείμενο την οπτικοποίηση ολόκληρης της ζωής μιας βάσης δεδομένων με την βοήθεια τεχνολογιών πολλαπλής όψης και διαδραστικότητας. Ένα βασικό χαρακτηριστικό του εργαλείου είναι η οπτικοποίηση (α) του «Διαγράμματος Παράλληλων Ζωών», και (β) της σύνοψης της ζωής της βάσης. Εκτός από την υποστήριξη αυτών των δύο βασικών χαρακτηριστικών, το εργαλείο είναι επίσης εφοδιασμένο με λειτουργίες όπως η εμβάθυνση σε συγκεκριμένα σημεία της σύνοψης, το φιλτράρισμά της σύμφωνα με διάφορα κριτήρια αλλά και η παροχή στον χρήστη λεπτομερειών ύστερα από αίτησή του.

CHAPTER 1. INTRODUCTION

1.1 Thesis Scope
1.2 Thesis Structure

1.1 Thesis Scope

Studying the evolution of database schemata is of great importance as a change in the schema of the database (e.g., the deletion of a table or attribute) can impact semantically and syntactically the entire ecosystem of applications that are built on top of the database. The study of schema evolution entails extracting schema versions and their delta changes from software repositories, subsequently leading to the extraction of patterns and regularities. The history of a typical database can consist of hundreds of transitions from version to version and includes a potentially large number of tables that change during the life of the database.

One of the main tools to study schema evolution is the visual inspection of the history of the schema. This can allow the scientists to construct research hypotheses as well as to drill into the details of inspected phenomena and try to understand what has happened at particular points in time. However, such an effort is constrained by our means of visual representation. Mainly, such a representation is targeted for a two-dimensional representation target in a computer screen or a printed paper. The space available in these representation media is simply too small for encompassing the hundreds of transitions from one version to another and the hundreds of tables involved in such a history, at the

same time. In other words, if we want to represent the space of transitions x tables in a two dimensional canvas, it is practically impossible, even for the typical case to put in every detail in the diagram, in a way that is humanly exploitable.

A solution to the problem is to follow the traditional method for handling this kind of problems [Shne96] and start with an overview of the detailed representation. This becomes even more useful if this overview is enriched with features such as zooming into specific point of the history, or filtering by different elements, etc. *Our solution to the problem starts with the main idea of creating a synopsis of the history of the schema evolution, where the number of transitions is abstracted by a limited set of phases and the number of tables is represented by a limited number of table clusters. Then, we can represent this synopsis as a 2D diagram, with phases at the x-axis and clusters at the y-axis, with the details of change in the contents of this 2D space.*

The first challenge that we have addressed involves fitting the timeline of transitions into a fixed space on screen. This requires extracting phases of the timeline. The related work includes a small number of works on this issue. So, our take to the problem was to introduce a hierarchical agglomerative clustering algorithm that merges the most similar transitions according to the time that have been committed and their heartbeat of changes into one phase. As a result, in the end we can have a desired number of phases, each of which encompasses subsequent and similar transitions.

The main advantage of a hierarchical agglomerative clustering algorithm is that it can always give us the best quality of clustering for a given distance function on the data that will be called to cluster. The main disadvantage of this type of algorithms is that for large datasets, it comes with reduced performance and speed, as all the algorithms of this family have to compute a distance for each pair of data points, and progressively merge smaller clusters into bigger one. However, within the scope of our problem, the datasets involved contain only some hundreds of data points, which is not prohibitive in terms of performance.

To make the algorithm work we introduce a distance function that computes the distance between two contiguous transitions. This function depends on two parts. The first part has to do with the time distance between the involved transitions, whereas the second part has to do with the difference between the numbers of changes that have been committed to these transitions. The more closely two transitions are with regards to time and changes, the most similar they are considered.

Second, we address a respective challenge to reduce the large number of tables of a database that have to be fitted into a fixed height of window. As already mentioned, we extract clusters out of the entire set of the tables. Our solution adopts the same approach as with transitions and we introduce another hierarchical agglomerative clustering algorithm that creates a desired number of clusters. Within each cluster, the desideratum is to maximize the similarity of the contained tables. In this case, the distance function considers three parameters that can characterize a table: The first parameter involves the birth dates of the involved tables; the second parameter is concerns the death dates of these tables; the last parameter has to do with the number of changes of a table. Combining them, we can find similar tables that were born and removed in close dates and also have a similar number of changes.

We have combined our abovementioned contribution in a tool that is called Plutarch's Parallel Lives that allows the interactive exploration of the history of schema. The tool gets the history of the database, i.e., a temporally sorted list of schemata. Firstly, the tool produces a detailed visualization of the life of the database, called Parallel Lives Diagram that contains the transitions of the database in its x-axis and the tables in its y-axis. Each row of this visualization contains the entire life of a table. Secondly, the tool extracts an overview for this visualization, which has the phases that are produced by the phase extraction algorithm in its x-axis and the clusters that have been extracted by the clustering extraction algorithm in its y-axis. This overview has been designed to be interactive and can provide the user with features like zooming into specific points, filters according to specific criteria and details on demand. Plutarch's

Parallel Lives has been designed to be easily extended due to its architecture that includes all the basic principles of object-oriented programming such as abstract coupling via interfaces, factories, and similar mechanisms.

In summary, we can list our contributions as follows:

- We have designed a phase extraction algorithm for the transitions of the entire life of a database that segments this life into phases according to the similarity of the transitions.
- We have designed an algorithm for cluster extraction over the entire set of the tables that are contained by a database, which divides the tables into clusters according to the similarity of their lives and in particular the similarity of their birth, death and change activity.
- We have assessed our algorithms with a principled method. We have used two different methods to assess our phase extraction algorithm. At the same time, we have also used both internal and external validity evaluation techniques to assess our clustering algorithm.
- We have designed and implemented a tool called Plutarch's Parallel Lives, in short, PPL [PPL15], that visualizes the entire life of a database. When a dataset is given as input to Plutarch's Parallel Lives, PPL produces an overview of the life of the respective database. Continuing, the tool allows the user to zoom into specific points of the overview, to filter it according to various features and to get information about details of the transitions and tables.

1.2 Thesis Structure

The structure of this thesis is as follows. In Chapter 2, we give the definitions of the main concepts of the thesis. In Chapter 3, we present the problem formulations and intuition on how we could solve them. In Chapter 4, we present the solutions that we designed for these problems and how we assessed these solutions. In Chapter 5, the interested reader can find details about the software that we have developed, such as the design and analysis of the Plutarch's Parallel Lives, some coding of the most significant classes and some screenshots of how it

performs. In Chapter 6, there is an overview of the related work that we have studied. Finally, in Chapter 7, we conclude our results and offer roads for future work.

CHAPTER 2. FUNDAMENTALS

2.1 Definitions of main concepts
2.2 Identified Changes per Transition of each Relation
2.3 Matrix Representation of Database Evolution
2.4 Visual representation of a history of a database

2.1 Definitions of main concepts

In this subsection, we start by giving the definitions and terminology for the concepts of dataset, version, transition, and revision.

Schema Version, or simply, Version: A snapshot of the database schema, committed to the public repository that hosts the different versions of the system. To facilitate our deliberations, we assign an artificial version id to each version, in the form of an auto-incrementing integer with step one. Thus, version ID's are isomorphic to a contiguous subset of the set of positive integers. Whenever possible, we also assign a timestamp to the version, which is the commit time to the public repository.

Synonymous term: Commit.

Dataset: A sequence of versions, respecting the order by which they appear in the repository that hosts the project to which the database belongs. In our case,

we have monitored only the versions committed to the trunk (master development branch) of the project to which the database belongs.

Transition: The fact that the database schema has been migrated from version v_i to version v_j, $i < j$. We refer to v_i as the source version of the transition and to v_j as the target version of the transition. We denote such a transition by an arrow from the source towards the target of the transition, e.g., $v_i \rightarrow v_j$. Throughout all our deliberations, we employ the term old to refer to properties of the source version and the term new to refer to properties of the target version of a transition. Each transition includes a set of changes to the DB schema. We discuss these kinds of changes in the following subsection.

Revision: A transition between two sequential versions, i.e., from version v_i to version v_{i+1}. Each transition incurs a set of differences to the database schema. We measure the alteration of tables and attributes in a manner that will be clearly defined right away. To simplify expressions, we frequently use the term "version" instead of "transition that leads to this version". So, for example, if we say the "new number of relations for version v_{i+1}", we refer to the number of tables of v_{i+1}, after a transition from v_i has taken place.

In the rest of our deliberations, unless otherwise specified, the term transition refers to a revision. Apart from an old and a new version, each revision has a timestamp, which signifies the date that the target, new version was made public on the public repository from which it was retrieved. For convenience, we also assign revisions with id's which are consecutive integers.

2.2 Identified Changes per Transition of each Relation

The evolution history of each database schema can be thought of as (a) a sequence of versions, but also as (b) a sequence of revisions. For each relation of the database schema, and for each revision, we identify the set of changes that have occurred.

Specifically, for each transition, for each relation, we can identify the following data:

Old Attributes: The set of attributes of the relation at the source, old version of the transition.

New Attributes: The set of attributes of the relation at the target, new version of the transition.

Attributes Inserted: The set of attribute names inserted in the relation in the new version of the transition. Attribute insertions can be of two types: *attributes inserted at table formation*, which are the attributes with which the table is born (i.e., the relation did not exist in the source version of the transition) and a*ttributes inserted to an existing relation*, of the relation existed in the old version of the transition.

Note that the two above subcategories of "Attributes inserted" are mutually exclusive for the case of a specific relation in a specific transition: either the relation is in its "birth" transition, in which case we have attributes inserted at table formation, or the transition takes place after the relation's birth, in which case we have tables injected in an existing relation.

Note also that although this is a mutually exclusive situation for a specific relation in a specific transition, the two metrics are can have non-zero values simultaneously when we study the evolution at the database level (meaning that, in the context of a specific transition, a new table can be created and another table can be updated at the same time).

Attributes Deleted: The set of attribute names deleted from the relation during the transition from the old to the new version. Attribute deletions can be of two types: (a) *attributes deleted at table removal*, which are the attributes that existed in relation in the source version of the transition, while, at the same time, the

relation does not exist in the target version of the transition, and, (b) a*ttributes deleted from a surviving relation*, for a relation that continues to exist in the new version of the transition.

The comments of the previous category concerning the disjointness of the two subcategories apply here too, in a direct manner.

Attributes with Type Alternations: The set of attributes whose data type changed during a transition.

Attributes involved in Key Alternations: The set of attributes that reversed their status concerning their participation to the primary key of the relation between the old and the new version of the transition. Specifically, these are the attributes that either became primary keys in the new version (while they were not in the old version) or stopped being part of the primary key in the new version while being part of the primary key in the old version.

The above sets can be treated (a) directly as sets, but, most commonly, (b) via the cardinality of this set. In other words, to measure the amount of change, we measure the size of each of these sets. In Table 2-1, we summarize the notation for the different sets and measures of change.

Having the respective measures, we can also measure the *growth* and alteration of a relation, as well as its total change. Specifically, we employ the following measures for the change of a relation during a transition:

Attribute Alternations: The sum of Attribute Type Alterations and Attribute Key Alterations.

Schema Growth: The difference between the cardinalities of the new and old attributes of a transition, i.e., *Number of New Attributes – Number of OldAttributes* (attn: not the absolute, but the actual value of the subtraction).

Total Change: the sum of absolute values of Attribute Alterations and Schema Growth.

2.3 Matrix Representation of Database Evolution

We define the *history of a database schema between revisions s*(tart) *and e*(nd) as a sequence of contiguous revisions: $\mathbf{H} = \{ t_s, t_{s+1}, \ldots, t_{e-1}, t_e\}$. Assuming that our knowledge for the life of a schema spans from revision 1 to revision *m*, we will use the simplified term *history* or *entire history of a database schema* to refer to the history between revisions 1 and *m*. The *diachronic schema of the database* is the union of all relation names appearing in the schema of the database throughout its entire history; we denote it as $\mathbf{S}_H = \{R_1, \ldots, R_n\}$.

To measure evolution, we adopt a convenient representation of the history of a database schema as a two-dimensional matrix, with one row per relation and one column per transition. The content of each cell of the matrix is a tuple with all the measures that correspond to the specific transition of the specific table that act as coordinates of the cell. We refer to this matrix as the *CART Matrix* (Change Analysis per Relation and Transition).

$$CART[R, t] = [A_t^{old}(R), A_t^{new}(R), I_t(R), D_t(R), T_t(R), K_t(R), U_t(R), g_t(R), M_t(R), Ch_t(R)]$$

Then, we can define the projection of the CART Matrix per (a) measure, if we are interested in only one measure, (b) aggregate measures per transition, where we aggregate all the measures for all relations for each transition, or (c) aggregate measures per relation, where we can aggregate the change measures for each relation over all the transitions. We can employ several aggregate functions for the two marginal aggregate measures (e.g., *sum* to measure total change, *count* to measure occurrences of change, *avg* to measure average change, *max* to see peaks in change, etc).

Table 2-1: Types of changes that occur to a relation R, during a transition t

Set of attributes involved in the change		*Measures of change for a relation R during a transition t*	
Old Attributes	$\mathbf{A}_t^{old}(R)$	*Number of Old Attributes*	$A_t^{old}(R)$
New Attributes	$\mathbf{A}_t^{new}(R)$	*Number of New Attributes*	$A_t^{new}(R)$
Attributes Inserted	$\mathbf{I}_t(R)$	*Attribute Insertions*	$I_t(R)$
Attributes Deleted	$\mathbf{D}_t(R)$	*Attribute Deletions*	$D_t(R)$
Attributes with Type Alterations	$\mathbf{T}_t(R)$	*Attribute Type Alterations*	$T_t(R)$
Attributes in Key Alterations	$\mathbf{K}_t(R)$	*Attribute Key Alterations*	$K_t(R)$
		Schema Growth	$g_t(R) = A_t^{new}(R) - A_t^{old}(R)$
		Attribute Alterations	$U_t(R) = T_t(R) + K_t(R)$
		Schema Modifications	$M_t(R) = I_t(R) + D_t(R)$
		Total Change	$Ch_t(R) = \|U_t(R)\| + \| M_t(R) \|$

2.4 Visual representation of a history of a database

We have developed a visualization tool that equips us with visual aids to study the history of a database.

We depict the matrix in its well-known, two dimensional rectilinear grid[1] format, having relations for rows and transitions for columns.

[1] See http://en.wikipedia.org/wiki/Regular_grid for the definition of the term

We define the *Parallel (Table) Lives Diagram of a database schema* as a two dimensional rectilinear grid having all the revisions of the schema's history as columns and all the relations of the diachronic schema as its rows. Each cell *PLD*[*i*,j] represents the changes undergone and the status of the relation at row *i* during the revision *j*.

Specifically, we employ the following visual notation:

- The blue cells (mildly grey in black and white) correspond to transitions where some form of change occurred to the respective table.
- Dark cells denote that the table was not part of the database at that time.
- Green solid cells (lightly colored in black and white) denote zero change.
- In Figure 2.1 there is a PPL diagram that has been extracted by Plutarch's Parallel Lives and contains a small part of Ensembl's database life. In its x-axis there is a part of the transitions that have been committed to the database. In y-axis we can see a part of the entire set of the tables of the database. Combining these with the above bullets we can realize that each row of the below figure contains the life of a table of a database including when it was born, when it died or when it was changed.

Whereas the ordering of the transitions is fixed and isomorphic to their timestamps, the ordering of relations can vary. We will discuss issues of relation ordering and grouping in Chapter 3.

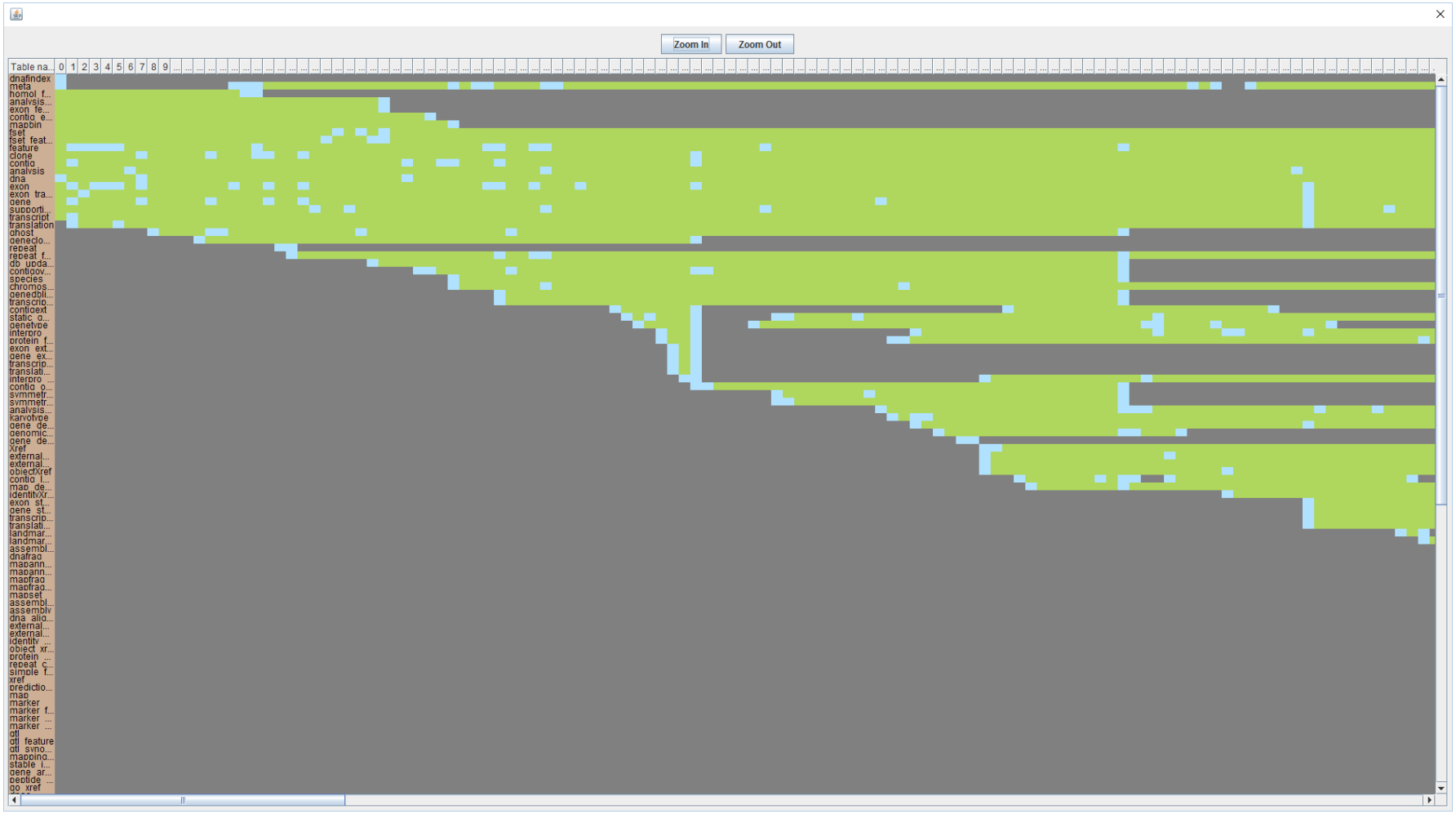


Figure 2.1: A part of Ensembl's PPL diagram

In the context of the Parallel Lives Diagram, we will refer to the line that corresponds to a specific relation *R* as the *Biography Line* of *R*. This is also why we will refer to the Parallel Lives Diagram as the *Parallel Biographies Diagram*, as an alternative terminology.

Intuition on the problem:

Although intuitive enough, the diagram of Fig. 2.1 suffers from the limitations of the two-dimensional display media that we use for showing it (on screen and on paper that is). Clearly, the available space that screens and paper-sheets can offer, requires an excessive shrinking of the Parallel Lives Diagram in order to fit within the available area. This makes the visual inspection process ineffective as crucial details are unobservable.

So the idea came from the mantra that Shneiderman underlines in his article at 1996 [Shne96], which is

Overview first, zoom and filter, details on demand.

We thought that the substitution of the extra-detailed diagram of Figure 2.1 with an overview that could be filtered and zoomed in and to provide us with helpful details on demand could be the ideal approach of our problem.

So in Chapter 3, we formalize these concepts as well as our solution to the interactive exploration process.

CHAPTER 3. PROBLEM SPECIFICATION

3.1 Segmentation of the history into phases
3.2 Clustering of tables into groups
3.3 Filter the overview
3.4 Details on demand

To address the problem of effectively representing the PLD in the limited space of 2D representation media, we need to solve two problems. The first problem deals with zooming out on transitions, and replacing them by phases, and the second one deals with zooming out on relations and replacing them by relation groups. More specifically we desire a number of phases that fits into a part of the width of the screen and also we desire a number of clusters that fits into a part of the height of the screen.

3.1 Segmentation of the history into phases

The idea is that we want to zoom-out on the time/version axis. So, we need to group transitions to *phases*, i.e., partition the set of transitions to disjoint groups of consecutive transitions, such that each phases is "homogeneous" internally (and disjoint from its neighbors).

The formulation of the problem is as follows:

Given the evolution history of a database schema,

group transitions into phases

such that the transitions of each phase share similar

3.2 Clustering of tables into groups

The idea is that we want to zoom-out on the vertical axis with the tables (in case the relations are too many). The idea here is that we partition the set of relations into disjoint subsets or else *clusters*. Each cluster has relations with similar lives i.e., lives with similar start, death and heartbeat of changes. This way we can zoom-out over the vertical dimension of the Parallel Lives Diagram (i.e., if the relations are too many, we can group them in a number of clusters that fits our visual space).

The formulation of the problem is as follows:

Given the evolution history of a database schema,

group relations into groups of relations with similar lives

such that the relations of each group share similar

3.3 Zoom into a specific point of the overview

If we have zoomed out the history of a relational database schema, there are many times that we would like to drill down more on what was happened to a specific point of this overview. For example, if we have a matrix in which the x-axis contains the phases that have been extracted and the y-axis contains the

tables of the database or the clusters that have been created how we could zoom into a specific cell of this table?

3.4 Filter the overview

Sometimes there is the desire to isolate a component of an overview including its elements to compare for example how similar are the elements from which it consists of. More precisely, maybe we would like to show up only the facts of a specific phase, or the behavior of the tables of one specific cluster, or even the life of a unique table. Then we would have to filter the overview according to these features.

3.5 Details on demand

According to the format of the PLD and the selection on it, we would like to have the ability to get some details on demand. For example, if the PLD contains in its x-axis the phases that have been extracted and in its y-axis the clusters that have been created by database's tables what details we could get about a cell of PLD?

To conclude, the tool that we will develop has to implement all of the above. Moreover, Plutarch's Parallel Lives must have the ability to create an overview for the importing dataset. This overview will be a combination of segmenting the entire set of transitions into phases according to their distance and sharing the entire set of tables into clusters according to their similarity. Furthermore, this overview has to be interactive to provide the user with the ability to zoom into a specific point, or with the ability to filter the overview according to the feature that he desires. Finally, it has to supply user with details on his demand about specific elements of the overview.

CHAPTER 4. CREATING AN OVERVIEW OF THE HISTORY OF A SCHEMA

4.1 Computing a segmentation of the history into phases
4.2 Grouping tables into clusters
4.3 Zoom into a specific point of overview
4.4 Filter the overview of the history of a relational database schema
4.5 Details on demand

In this subsection, we address the problems that were referenced in chapter 3. For the goal of extracting phases from the entire history of a database, we designed and implemented a Phasic Extractor. For the purpose of clustering of the tables, a Clustering Extractor was designed and implemented. Subsequently, we are going to analyze their main algorithms, the parameters that are needed and were explored and the distance metrics that were implemented for both of them. Finally, the assessment of these two methods completes the section.

4.1 Computing a segmentation of the history into phases

The Phasic Extractor gets as input the entire history of a database (and some extraction parameters that will be explained more in the sequel), and it produces a segmentation of the history into phases. More specifically, the Phasic Extractor parses the input (.csv file each line of which contains details about a transition from an older version of the database to a newer) and constructs one phase for

each transition. When the parsing will have been finished the Phasic Extractor will have as many phases as the number of the transitions that have been committed to the database. Next to this, is the execution of an agglomerative clustering algorithm that attempts to merge the most similar phases according to a distance metric. The new merged phases are given as input to the agglomerative clustering algorithm recursively, until the desired number of phases will have been extracted. At the end, database will be segmented according to the result of the recursive-clustering algorithm.

Algorithm: The Phasic Extractor algorithm

Input: A list of schema transitions H = { t_s, t_{s+1}, …, t_{e-1}, t_e}, the desired number of phases *k*, the weight to assign to time w_t, the desired weight to assign to changes w_c , the choice if we want the data to be preprocessed according to the time *prePocessingTime,* the choice if we want the data to be preprocessed according to the changes *prePocessingChanges*.

Output: A partition of H, P = {p_1...p_k}

variable *numPhases=e*, counter of the number of phases.

Begin

1. P={p_1,…p_e} s.t. p_i={t_i} ∀ i ∈ s…e
2. while(numPhases>k){
    a. **for** each pair of phases ph_i, ph_{i+1}, $1 \leq i \leq n$
        i. compute δ(ph_i, ph_{i+1})
    b. Merge the most similar phases, p_a and p_{a+1} into a new phase p'
    c. P = {p1,…, p_{a-1}, p, p_{a+1}, …, p_m}
    d. numPhases --

    }
3. Return P;

End

4.1.1 Parameters

One of the extraction parameters that are used by Phasic Extractor is associated with the desired number of phases that we wish to segment the history. The weight that we want to assign to the time and changes between transitions is another parameter. Finally the last part of the extracted parameters is the execution or not of some preprocessing methods which are associated with the time or changes between transitions. Transitions that have time distance less than three days or transitions that have zero number of changes are merged into a phase each, before Phasic Extractor begin processing the data.

- **Desired number of segments (k)**: refers to the number of phases that we would like to be extracted.
- **Pre-Processing Changes (PPC)**: refers to the preprocessing of the data from the aspect of changes (ON if the data has been preprocessed, OFF otherwise).
- **Pre-Processing Time (PPT)**: refers to the preprocessing of the data from the aspect of time (ON if the data has been preprocessed, OFF otherwise).
- **Weight Change (WC)**: refers to the weight of changes (0.5 normal weight, 0 if changes is not taken into account).
- **Weight Time (WT)**: refers to the weight of time (0.5 normal weight, 0 if time is not taken into account).

4.1.2 Distance Function

Next we present the distance function that gives as outcome how similar are two phases. This distance on its normal form that both time and changes have a non-zero weight depends on both to the time that the compared phases differ with regard to the date that have been committed and to the difference of the changes that have been occurred to each other too. Here is the definition of this distance function:

For two phases p_i, p_{i+1}:

$$\delta(p_i, p_{i+1}) = w^T \times \delta^T(p_i, p_{i+1}) + w^C \times \delta^C(p_i, p_{i+1})$$

Table 4-1: Explanation of the distance function

Symbolism	Description
$\delta(p_i, p_{i+1})$	Denotes the term of the Distance Function between phases
w^T	Denotes the weight that we want to assign to the time distance
$\delta^T(p_i, p_{i+1})$	Denotes the distance between the two phases with respect to the time. Actually is the distance between the death date of the p_i phase and the birth date of the p_{i+1} phase
w^C	Denotes the weight that we want to assign to the change distance
$\delta^C(p_i, p_{i+1})$	Denotes the distance between the number of changes of the p_i phase in relation to the number of changes of the p_{i+1} phase. Actually is the distance between the number of changes of the last transition of phase p_i and the first transition of phase p_{i+1}

4.1.3 Assessment of the method

4.1.3.1 Assessment via divergence from the mean

The first assessment method that we use to evaluate the quality of the phases that the Phasic Extractor extracts, is connected with the following formula:

$$E_{pn} = \left(\sum_{\forall phase\ ph_i} \sum_{\forall transition\ e_j \in ph_i} \left| \mu_i - e_j \right|^p \right)^{1/p}$$

where μ_i is the average number of changes of each phase and e_j is the number of changes of each transition of the phase. Typically p is equal to one or two. In our evaluation p was set to one. This formula could give us a good evaluation of our method, because it depicts how similar are the elements that are included by each phase according to the events that happened into this. The most similar they are, the smaller value of E_{pn} will give as a result.

So, for this method the goal is to find which set of extraction parameters give us the best results something that means the smallest E_{pn} and finally investigate a "winner" set of them, which is the set that performs better at most.

The datasets that were used by Phasic Extractor were eight different datasets from open-source databases such as Atlas, bioSQL, Coppermine, Ensembl, mediaWiki, Opencart, phpBB and typo3 with all possible weights of time and change and either with preprocessing or not and here is how the extracted phases were assessed by the first method for all of them.

	PPC:OFF PPT:OFF	PPC:ON PPT:OFF	PPC:OFF PPT:ON	PPC:ON PPT:ON
WC=0.0 WT=1.0	898.38	907.51	898.38	907.51
WC=0.5 WT=0.5	877.94	891.98	840.24	855.17
WC=1.0 WT=0.0	912.11	912.11	859.56	859.56

Figure 4.1: Atlas Dataset

	PPC:OFF PPT:OFF	PPC:ON PPT:OFF	PPC:OFF PPT:ON	PPC:ON PPT:ON
WC=0.0 WT=1.0	380.15	381.22	380.15	381.22
WC=0.5 WT=0.5	253.84	254.62	375.37	347.37
WC=1.0 WT=0.0	206.54	206.54	325.82	325.82

Figure 4.2: bioSQL Dataset

	PPC:OFF PPT:OFF	PPC:ON PPT:OFF	PPC:OFF PPT:ON	PPC:ON PPT:ON
WC=0.0 WT=1.0	136.45	130.74	136.45	130.74
WC=0.5 WT=0.5	112.54	121.16	130.86	135.71
WC=1.0 WT=0.0	108.29	135.39	138.20	134.35

Figure 4.3: Coppermine Dataset

	PPC:OFF PPT:OFF	PPC:ON PPT:OFF	PPC:OFF PPT:ON	PPC:ON PPT:ON
WC=0.0 WT=1.0	4111.28	4115.63	4111.28	4115.63
WC=0.5 WT=0.5	4081.30	4097.89	4155.04	4083.44
WC=1.0 WT=0.0	3737.57	4044.81	4124.37	3935.95

Figure 4.4: Ensembl Dataset

	PPC:OFF PPT:OFF	PPC:ON PPT:OFF	PPC:OFF PPT:ON	PPC:ON PPT:ON
WC=0.0 WT=1.0	1052.28	1052.28	1052.28	1052.28
WC=0.5 WT=0.5	1025.91	1042.27	1030.86	1053.47
WC=1.0 WT=0.0	920.34	920.34	1061.43	1047.30

Figure 4.5: mediaWiki Dataset

	PPC:OFF PPT:OFF	PPC:ON PPT:OFF	PPC:OFF PPT:ON	PPC:ON PPT:ON
WC=0.0 WT=1.0	3390.19	3381.58	3390.19	3381.58
WC=0.5 WT=0.5	1297.10	1294.76	2733.91	2731.19
WC=1.0 WT=0.0	837.30	837.30	2745.29	2743.91

Figure 4.6: Opencart Dataset

	PPC:OFF PPT:OFF	PPC:ON PPT:OFF	PPC:OFF PPT:ON	PPC:ON PPT:ON
WC=0.0 WT=1.0	870.53	880.23	870.53	880.23
WC=0.5 WT=0.5	861.10	941.45	853.23	791.49
WC=1.0 WT=0.0	843.11	843.11	953.27	872.68

Figure 4.7: phpBB Dataset

	PPC:OFF PPT:OFF	PPC:ON PPT:OFF	PPC:OFF PPT:ON	PPC:ON PPT:ON
WC=0.0 WT=1.0	648.59	644.33	648.59	644.33
WC=0.5 WT=0.5	658.19	664.04	664.39	485.49
WC=1.0 WT=0.0	486.84	486.84	477.48	438.35

Figure 4.8: typo3 Dataset

Table 4-2: Number of wins for different sets of parameters

	PPC: OFF PPT: OFF	PPC: ON PPT: OFF	PPC: OFF PPT: ON	PPC: ON PPT: ON
WC = 0.0 WT = 1.0	-	-	-	-
WC = 0.5 WT = 0.5	-	-	1	1
WC = 1.0 WT = 0.0	**5**	3	-	1

We say that a parameter configuration wins each time it produces the lowest Epn in one of the assessments of the Figures 4.1 – 4.8. Winners are depicted in green in all these figures. In Table 4-2 we show how many times the different sets of parameter configurations "win". Ultimately, the "winner" configuration of parameters is the one that (a) the data was not preprocessed neither from the aspect of time nor of change and (b) the time was not taken into account too (0.0 time weight). Second came the set of those parameters that the data was preprocessed according to changes preprocessing and the time was not taken into account again. So, from the results we can say that the time has not an important role in phasic analysis as concerning to these specific datasets and the changes have.

4.1.3.2 Assessment via spread in the time x change space

The second assessment method can be described as follows:
For each pair of phases ph_i and ph_{i+1} we have to compute the term δ^{time} as it has been defined previously. Another term that has to be computed is the term δ_{change} which is also has been defined previously. When these two terms have been computed for the whole set of pairs we can represent our results with the scatter plot format.

In Figures 4.9-4.20 we depict the results of this assessment method with Coppermine dataset as given input. Note that x-axis is referred to the active time

distance (distance in days) and the y-axis is associated with the active changes distance. We depict the actual (non-normalized) values for both the time and the change distance. If both distances had been normalized the image of the charts would be the same and only the values of the numbers would have been different. So, we can go on with the description of the tables that are following and with the results of this assessment method.

- **Phases**: the first column refers to the pair of the source phase and the destination phase.
- **δtime**: the second column refers to the active time distance from one phase to another.
- **δc (δchange)**: the third column refers to the active change distance from one phase to another.
- The last column represents the **scatter plot** of the pairs of the second and third column.
- Concerning the image part of the following Figures, the x-axis of all the scatterplots is *δtime* and the y-axis of the scatterplots is *δchange*.

WC: 0.0	**WT**: 1.0
PPC:OFF	**PPT**:OFF

Phases	**δtime**	**δc**
0@1	110.50	0
1@2	84.73	4
2@3	90.14	0
3@4	86.63	3
4@5	128.53	2
5@6	132.30	0
6@7	214.35	0
7@8	133.67	0
8@9	80.63	0

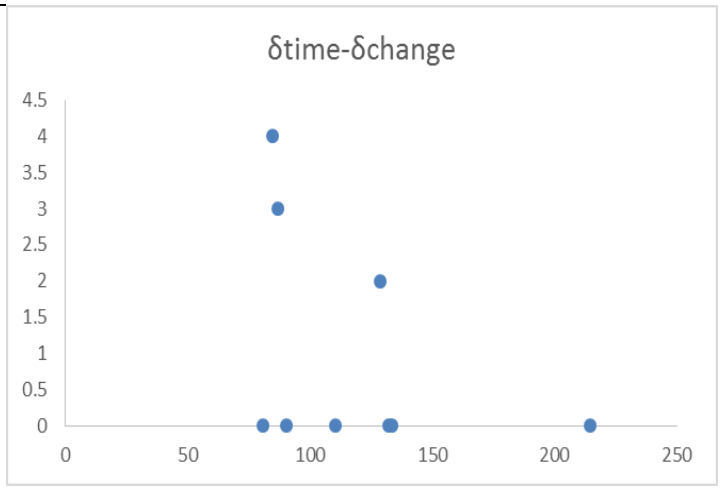


Figure 4.9: (δtime, δc) for (WC:0.0, WT:1.0, PPC:OFF, PPT:OFF)

WC: 0.0	**WT**: 1.0
PPC:ON	**PPT**:OFF

Phases	**δtime**	**δc**
0@1	66.91	5
1@2	84.73	4
2@3	80.07	1
3@4	86.63	3
4@5	128.53	2
5@6	63.76	7
6@7	66.46	1
7@8	62.45	1
8@9	56.50	1

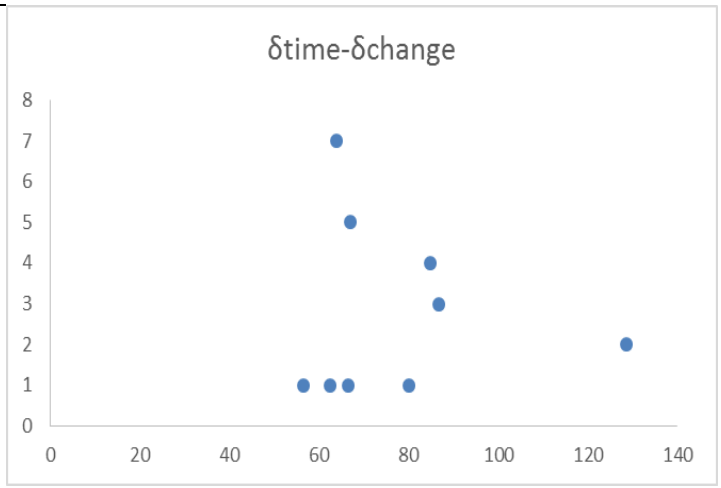


Figure 4.10: (δtime, δc) for (WC:0.0, WT:1.0, PPC:ON, PPT:OFF)

WC: 0.0	**WT**: 1.0
PPC:OFF	**PPT**:ON

Phases	**δtime**	**δc**
0@1	110.50	0
1@2	84.73	4
2@3	90.14	0
3@4	86.63	3
4@5	128.53	2
5@6	132.30	0
6@7	214.35	0
7@8	133.67	0
8@9	80.63	0

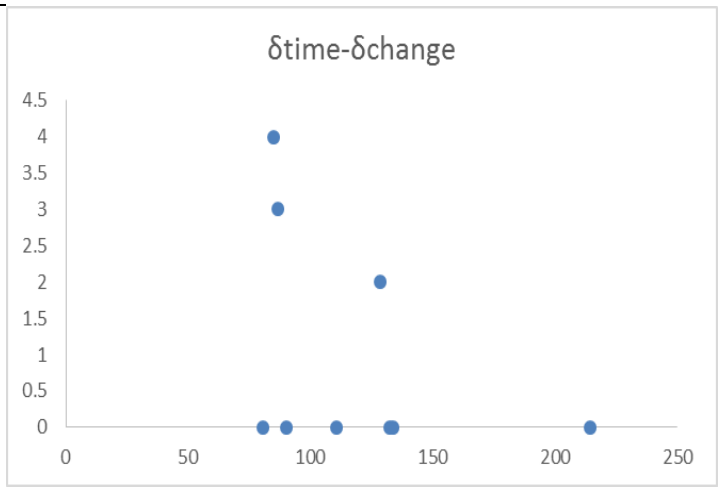


Figure 4.11: (δtime, δc) for (WC:0.0, WT:1.0, PPC:OFF, PPT:ON)

WC: 0.0	**WT**: 1.0
PPC:ON	**PPT**:ON

Phases	**δtime**	**δc**
0@1	66.91	5
1@2	84.73	4
2@3	80.07	1
3@4	86.63	3
4@5	128.53	2
5@6	63.76	7
6@7	66.46	1
7@8	62.45	1
8@9	56.50	1

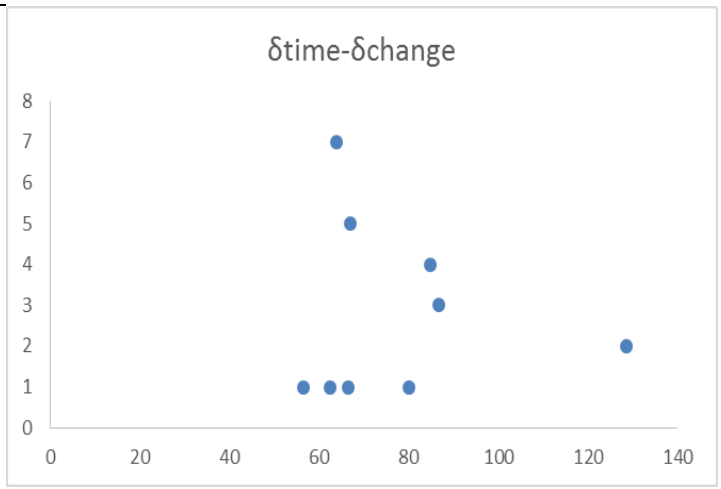


Figure 4.12: (δtime, δc) for (WC:0.0, WT:1.0, PPC:ON, PPT:ON)

WC: 0.5	**WT**: 0.5
PPC:OFF	**PPT**:OFF

Phases	**δtime**	**δc**
0@1	1.83	1
1@2	0.14	8
2@3	27.56	1
3@4	26.90	14
4@5	2.04	16
5@6	27.39	3
6@7	16.04	8
7@8	5.99	1
8@9	6.99	4

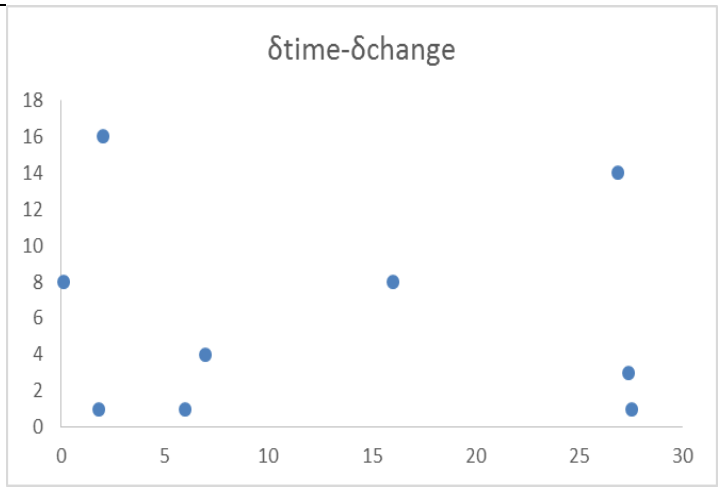


Figure 4.13: (δtime, δc) for (WC:0.5, WT:0.5, PPC:OFF, PPT:OFF)

WC: 0.5	**WT**: 0.5
PPC:ON	**PPT**:OFF

Phases	δtime	δc
0@1	66.91	5
1@2	5.66	1
2@3	2.04	16
3@4	27.39	3
4@5	84.73	4
5@6	43.09	1
6@7	16.04	8
7@8	5.99	1
8@9	6.99	4

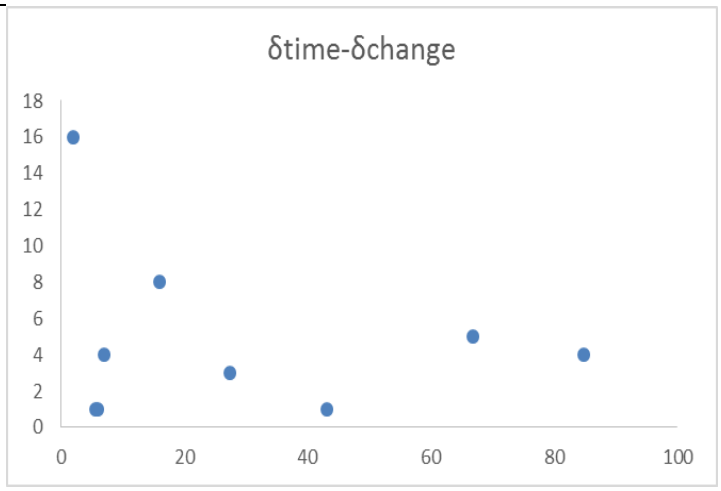


Figure 4.14: (δtime, δc) for (WC:0.5, WT:0.5, PPC:ON, PPT:OFF)

WC: 0.5	**WT**: 0.5
PPC:OFF	**PPT**:ON

Phases	δtime	δc
0@1	21.06	1
1@2	9.47	7
2@3	24.45	0
3@4	27.39	3
4@5	84.73	4
5@6	86.63	3
6@7	16.04	8
7@8	5.99	1
8@9	214.35	0

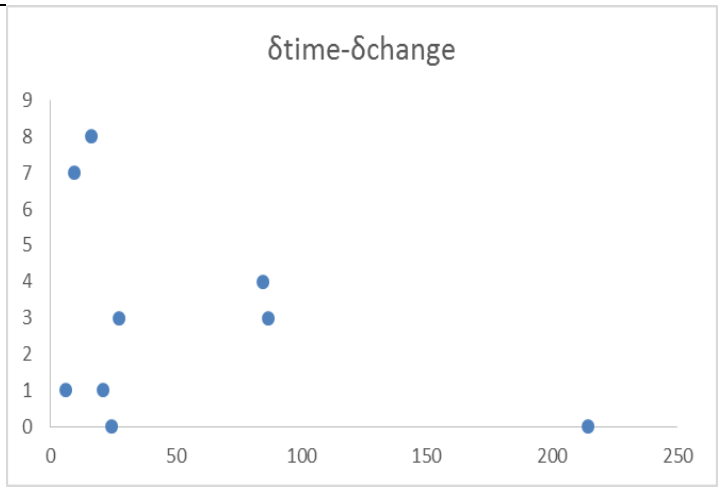


Figure 4.15: (δtime, δc) for (WC:0.5, WT:0.5, PPC:OFF, PPT:ON)

WC: 0.5	**WT**: 0.5
PPC:ON	**PPT**:ON

Phases	**δtime**	**δc**
0@1	21.06	1
1@2	5.66	1
2@3	84.73	4
3@4	43.09	1
4@5	86.63	3
5@6	16.04	8
6@7	5.99	1
7@8	4.11	2
8@9	66.46	1

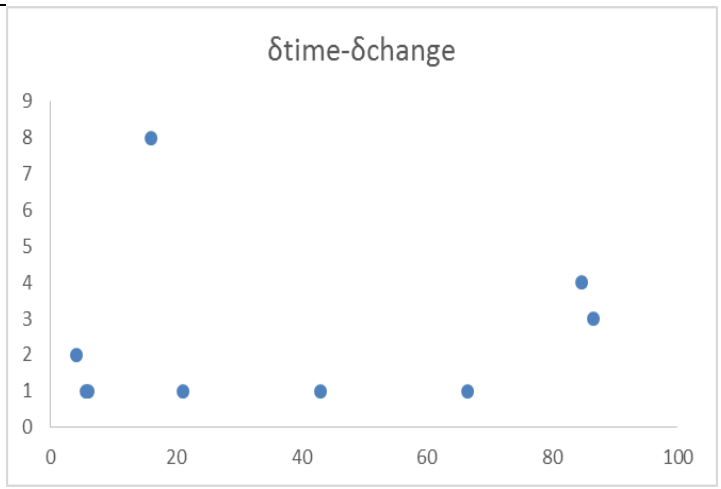


Figure 4.16: (δtime, δc) for (WC:0.5, WT:0.5, PPC:ON, PPT:ON)

WC: 1.0	**WT**: 0.0
PPC:OFF	**PPT**:OFF

Phases	**δtime**	**δc**
0@1	0.72	7
1@2	9.47	7
2@3	26.90	14
3@4	2.04	16
4@5	27.39	3
5@6	47.45	2
6@7	0.55	2
7@8	0.03	2
8@9	6.99	4

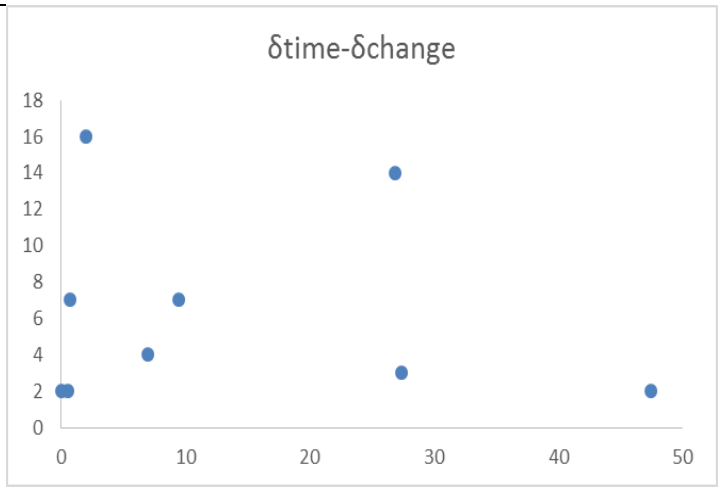


Figure 4.17: (δtime, δc) for (WC:1.0, WT:0.0, PPC:OFF, PPT:OFF)

WC: 1.0	**WT**: 0.0
PPC:ON	**PPT**:OFF

Phases	**δtime**	**δc**
0@1	0.72	7
1@2	9.47	7
2@3	1.83	1
3@4	0.14	8
4@5	2.04	16
5@6	27.39	3
6@7	47.45	2
7@8	0.55	2
8@9	0.03	2

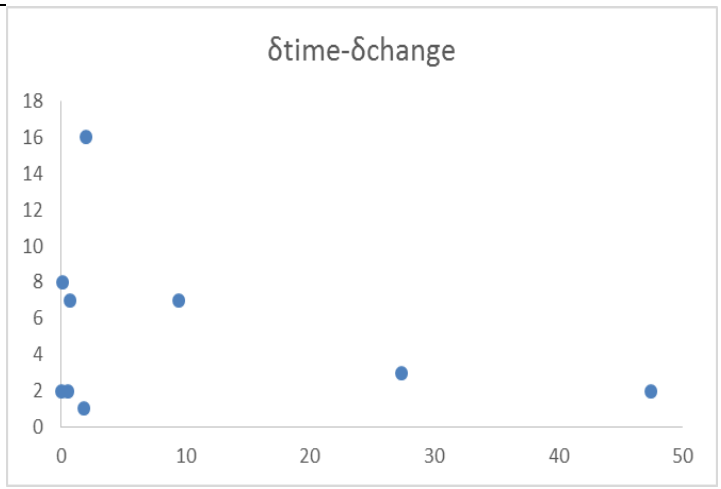


Figure 4.18: (δtime, δc) for (WC:1.0, WT:0.0, PPC:ON, PPT:OFF)

WC: 1.0	**WT**: 0.0
PPC:OFF	**PPT**:ON

Phases	**δtime**	**δc**
0@1	12.34	0
1@2	21.06	1
2@3	9.47	7
3@4	24.45	0
4@5	27.39	3
5@6	47.45	2
6@7	5.99	1
7@8	4.11	2
8@9	66.46	1

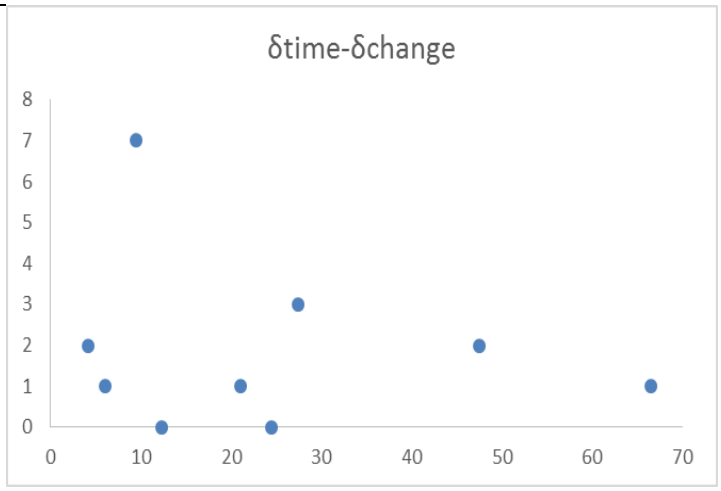


Figure 4.19: (δtime, δc) for (WC:1.0, WT:0.0, PPC:OFF, PPT:ON)

WC: 1.0	**WT**: 0.0
PPC:ON	**PPT**:ON

Phases	**δtime**	**δc**
0@1	21.06	1
1@2	9.47	7
2@3	5.66	1
3@4	84.73	4
4@5	47.45	2
5@6	16.04	8
6@7	5.99	1
7@8	4.11	2
8@9	66.46	1

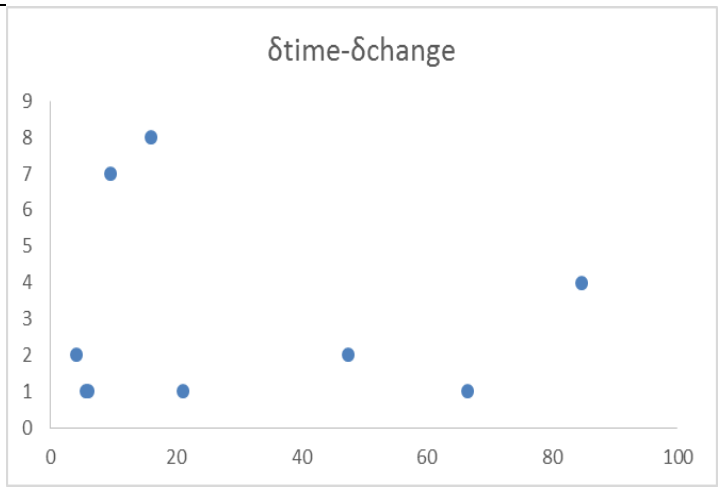


Figure 4.20: (δtime, δc) for (WC:1.0, WT:0.0, PPC:ON, PPT:ON)

Observe that the results vary depending on the values of the extraction parameters in combination with the "morphology" of the dataset. So there are some plots that show that the changes had more weightiness in the phasic analysis and the points are close to the y-axis ignoring the time factor. On the other hand there are some others that are close enough to the x-axis, something that shows that the change factor had not an important role to the phasic analysis such as the time factor. Finally there are some plots that the points are spreading to the whole chart something that evokes that both of the two factors play similar role to the extraction of the phases.

All the results from all the datasets can be found in Appendix.

4.2 Grouping tables into clusters

In this subsection we solve the problem of grouping the tables of a database into a cluster to shrink the y-axis of a PLD. We use an agglomerative clustering

algorithm which we call Clustering Extractor because this type of algorithms gives the best result if there are not performance limitations. The Clustering Extractor gets as input an object that contains the whole set of the tables that have been appeared during the life of the database and a set of parameters (will be analyzed in the following subsection) and gives as output a desired number of clusters. More specifically, the initial step of the algorithm constructs one cluster for each table, so initially we have as many clusters as tables. The next step has to do with the computation of the similarity for each pair of the entire set of clusters according to a distance function that will be analyzed in the sequel. Then, the most similar pair of compared clusters will be merged into one common. This procedure is repeated until the desired number of clusters has been created. At the end of this process, the initial set of database's tables will have been partitioned into a set of clusters, that each of them will contain a much smaller number of tables that have "common" lives.

Algorithm: The Clustering Extractor algorithm
Input: The entire set of the database's tables T {tab_1, ... , tab_n}, the desired number of clusters *k*, the weight to assign to birth date $w_{b,}$, the weight to assign to death date $w_{d,}$, the weight to assign to heartbeat of the changes date w_c
Output: A partition of T, C={c_1, … , c_k}
variable *numClusters=n*, counter of the number of clusters
Begin

1. C={c_1, … , c_n} s.t. c_i = {tab_i} ∀ i ∈ 1…n
2. while(numClusters>k){
    a. **for** each pair of clusters c_i, c_{i+1}, $1 \leq i \leq n$
        i. Compute the δ(c_i,c_{i+1})
    b. Merge the most similar clusters, c_a and c_{a+1} into a new cluster c'
    c. C = {c_1,…, c_{a-1}, c, c_{a+1}, …, c_m}
    d. numClusters –-
   }
3. Return C;

End

4.2.1 Parameters

The first parameter of Clustering Extractor is related with the desired number of the clusters that we would like to be created. The next two parameters have to do with the weight that we want to assign to the distance between the birth or death date of the compared clusters respectively. As date we consider the transition ID for both cases. The last parameter that is needed for the clustering extraction is associated with the weight of the distance between changes that have been committed to each cluster.

- **Desired number of clusters (k)**: refers to the number of clusters that we would like to be created.
- **Birth Weight (BW)**: refers to the weight of the distance between birth dates of compared clusters.
- **Death Weight (DW)**: refers to the weight of the distance between death dates of compared clusters.
- **Change Weight (CW)**: refers to the weight of the distance between the changes of compared clusters.

4.2.2 Distance Function

The distance function of the clustering analysis contains terms that are associated with the distances between two clusters as regards birth date, death date and number of changes between two compared clusters and their assigned weights. This distance metric is reflected to the following formula, the values of which are normalized:

$$\delta(cluster_A, cluster_B) = w_b * |\delta_{birth}(c_A, c_B)| + w_d * |\delta_{death}(c_A, c_B)| + w_c * |\delta_{change}(c_A, c_B)|$$

Table 4-3: Explanation of the distance function

Term	Description	Formula
$\delta(cluster_A, cluster_B)$	Total distance between two clusters	
w_b	The weight that will be assigned to the distance that is related with the birth date	
$\delta_{birth}(c_A, c_B)$	The distance between birth dates of the two compared clusters	Plain $c_A.birth - c_B.birth$ Normalized $\frac{\delta_{birth}(c_A, c_B)}{DB\ duration}$
w_d	The weight that will be assigned to the distance that is related with the death date	
$\delta_{death}(c_A, c_B)$	The distance between death dates of the two compared clusters	Plain $\begin{cases} \emptyset, \textbf{\textit{if both alive}} \\ c_A.death - c_B.death, \textbf{\textit{else}} \end{cases}$ [2] Normalized $\frac{\delta_{death}(c_A, c_B)}{DB\ duration}$
w_c	The weight that will be assigned to the distance that is related with the total changes	

[2] If one of the compared clusters is still alive, then its death date is set to the max transition of the database history +1.

$\delta_{change}(c_A, c_B)$	The distance between the total changes that have been committed to the two compared clusters	Plain $c_A.changes - c_B changes$ Normalized $\frac{\lvert Ch(A)\rvert - \lvert Ch(B)\rvert}{\lvert Ch(A)\rvert + \lvert Ch(B)\rvert}$ where Ch is the total number of changes

4.2.3 Assessment of the method

In this subsection we will discuss about clustering validity and we will try to evaluate our clustering technique. In general, there are two main categories for clustering validity, the internal evaluation and the external evaluation [TaSK05]. The first one refers to methods that do not need external knowledge and can measure the quality of the clusters that have been produced only with the information that they keep and which was used from the clustering algorithm. Otherwise, external evaluation needs external knowledge, i.e., data have to be classified before the evaluation process, by explicit tracing of human knowledge on the issue.

4.2.3.1 External Evaluation

For this type of clustering validity there is a large amount of methods that have been used previously. We decided to choose the most common of them, which are Entropy, Precision, Recall and F-measure. Our basic source for a more comprehensive studying on these metrics was [TaSK05].

Entropy: Entropy is defined as the degree to which each cluster consists of objects of a single class. Moreover, for each cluster *j* we compute p_{ij}, which is the

probability that a member of cluster *i* belongs to class *j*. This probability is given by the following formula:

$$p_{ij} = \frac{m_{ij}}{m_i}$$

where m_i is the number of objects in cluster *i* and m_{ij} is the number of objects of class *j* in cluster *i*.

So the total entropy of each cluster *i* is calculated by the following formula:

$$e_i = -\sum_{j=1}^{L} p_{ij} \log_2 p_{ij}$$

where *L* is the number of classes.

In this point, we can define the total entropy of a set of clusters, as the sum of the entropies of each cluster weighted by the size of each cluster:

$$e = \sum_{i=1}^{K} \frac{m_i}{m} e_i$$

where *K* is the number of clusters and *m* is the total number of data points.

This metric substantially give us the purity of the clustering. The less objects of different classes exist to a cluster the better and so on the smaller value of entropy. So we want to achieve as soon as smaller values of entropy.

Precision: Precision is defined as the fraction of a cluster that consists of objects of a specified class. Precision of a cluster *i* with respect to class *j* is:

$$precision(i, j) = p_{ij}$$

Recall: Recall depicts the extent to which a cluster contains all the objects of a specified class. The recall of cluster *i* with respect to class *j* is:

$$recall(i,j) = \frac{m_{ij}}{m_j}$$

where m_j is the number of objects in class *j*.

F-measure: F-measure consists of both precision and recall and measures the extent to which a cluster contains only objects of a particular class and all objects of that class. The F-measure of cluster *i* with respect to class *j* is calculated by this formula:

$$F(i,j) = \frac{2 \times precision(i,j) \times recall(i,j)}{precision(i,j) + recall(i,j)}$$

So, as we have defined the basic metrics that have been used to evaluate our clustering algorithm we have to say some words about the procedure to classify the tables clusters that then would be evaluated. We studied four different datasets (Atlas, bioSQL, Coppermine, phpBB) and we tried to classify their tables.

The source of our classification procedure was the PLD (Parallel Live Diagram). The most obvious criteria of the PLD are when a table is born (birth date) and when a table died and not as much the count of changes of each table. So, the classification is based on more on the first two criteria rather than the third one.

Now we can cite our results for four different datasets (Atlas, bioSQL, Coppermine, phpBB). Firstly, because of the large amount of data and space that are needed to show up precision, recall and F-measure results for all the datasets in combination with our desire not to overflow reader we will analyze only the bioSQL dataset and the results for the rest will be placed in Appendix. Secondly we will present and discuss all the results that are referred to entropies.

At this point, we will study the bioSQL dataset a little more comprehensively. First of all, in Figure 4.21 there is a preview of the classification of the clusters which was extracted from our tool (PPL tool). In bioSQL there are totally 45 tables and we constructed four classes to classify these tables (red, yellow, black and purple). For some tables, is obvious in which class they belong, but for some others it is a little more complicated. The "red class" consists of tables that were born at the beginning of the life of bioSQL database but they died early too and it is represented by letter "R" in fig. 4.21. The "yellow class" contains tables that were born in the middle of the life of the database but died a few versions later too and it is represented by letter "Y". The "purple class" has tables that were born after the middle of the database's life and live until the end too and it is represented by letter "P". Finally, the most complicated class, the "black class" consists initially of tables that were born at the beginning of the life of the database and lived throughout its whole life too and it is represented by letter "B".

There are some tables that is not obvious how to be classified. For example, *seqfeature location* table was classified to "black" class and not to the "red" as someone could expect. To understand this we have to have a look on a prior state of classifying procedure when neither *seqfeature location* nor *cache_corba_support* had been classified. In some state of the classification process and after the obvious tables had been classified it was the turn of the *cache_corba_support* table to be classified. *Cache_corba_support* has the same distance with respect to birth date either to "red class" or to "black class". With respect to the death date *cache_corba_support*'s distance to the "black class" is 14 versions, whereas between the "red class" whose death date is the death date of *remote_seaqfeature_name* is 16. So *cache_corba_support* was classified to "black class". Then it was the turn of *seqfeature_location*, but now it is obvious that this table is more similar with *cache_corba_support* that belongs to the "black class" rather than the *remote_seaqfeature_name* that belongs to the "red class". So, *seqfeature_location* was classified to the "black class". This is the logic behind the classification process.

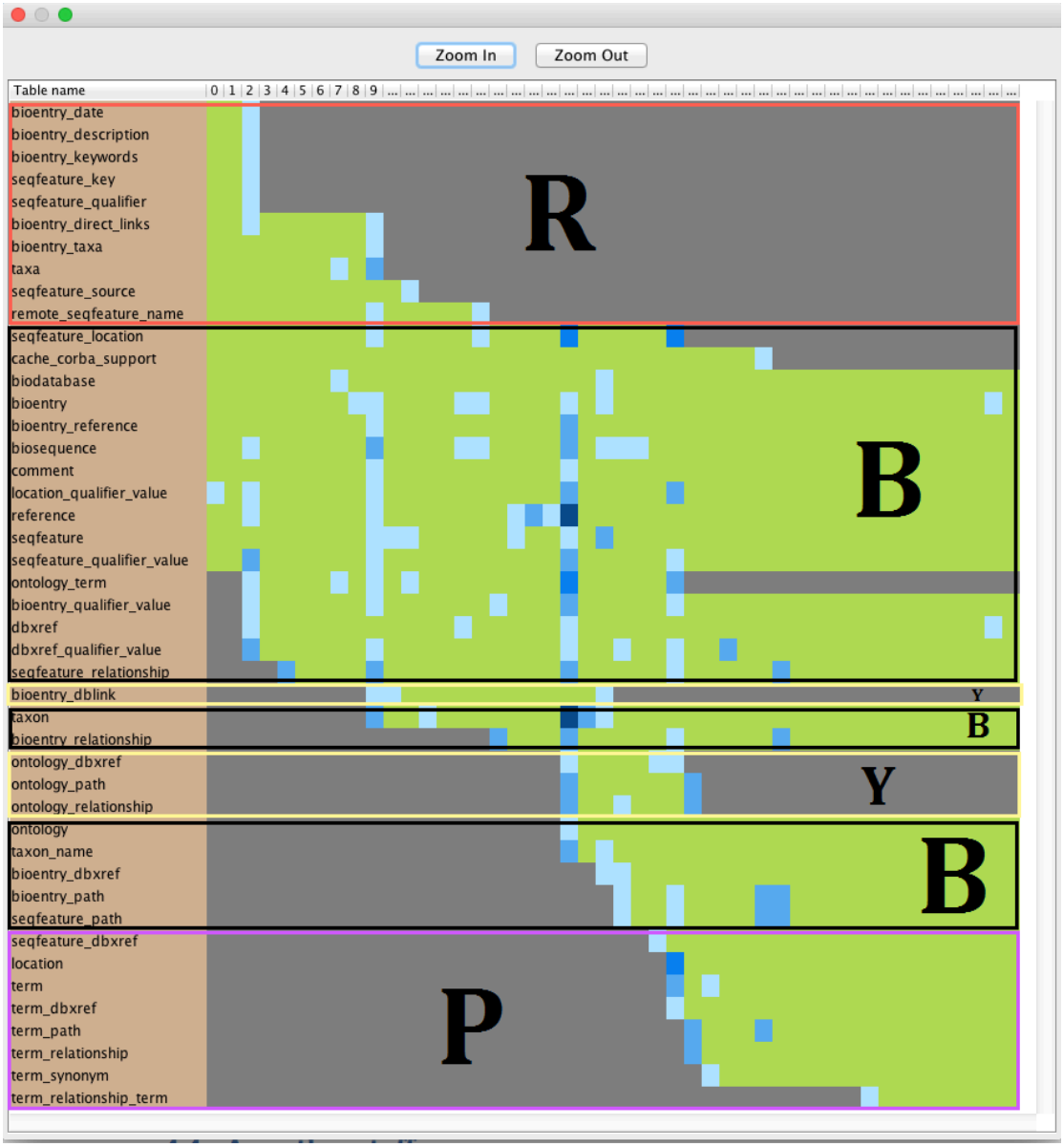


Figure 4.21: bioSQL dataset with classes

After the explanation of the classification process we can have a look at the results about the others measures that were analyzed before for the bioSQL dataset and for different set of parameters. We have to note that Class 1 of the tables with the results refers to “red class”, Class 2 to “black class”, Class 3 to “purple class” and Class 4 to “yellow class”.

Table 4-4: Results for wb: 0.333 , wd: 0.333 , wc: 0.333

	Class 1 (R)	Class 2 (B)	Class 3 (P)	Class 4 (Y)
Precision				
Cluster 0	0.50	0.50	0.00	0.00
Cluster 1	0.31	0.55	0.00	0.14
Cluster 2	0.00	0.55	0.45	0.00
Cluster 3	0.00	0.00	1.00	0.00
Recall				
Cluster 0	0.10	0.04	0.00	0.00
Cluster 1	0.90	0.70	0.00	1.00
Cluster 2	0.00	0.26	0.63	0.00
Cluster 3	0.00	0.00	0.38	0.00
F-Measure				
Cluster 0	0.17	0.08	0.00	0.00
Cluster 1	0.46	0.62	0.00	0.24
Cluster 2	0.00	0.35	0.53	0.00
Cluster 3	0.00	0.00	0.55	0.00

Table 4-5: Results for wb:0.0 wd:1.0 wc:0.0

	Class 1 (R)	Class 2 (B)	Class 3 (P)	Class 4 (Y)
Precision				
Cluster 0	1.00	0.00	0.00	0.00
Cluster 1	0.83	0.00	0.00	0.17
Cluster 2	0.00	0.50	0.00	0.50
Cluster 3	0.00	0.71	0.29	0.00
Recall				
Cluster 0	0.50	0.00	0.00	0.00
Cluster 1	0.50	0.00	0.00	0.25
Cluster 2	0.00	0.13	0.00	0.75
Cluster 3	0.00	0.87	1.00	0.00
F-Measure				
Cluster 0	0.67	0.00	0.00	0.00
Cluster 1	0.63	0.00	0.00	0.20
Cluster 2	0.00	0.21	0.00	0.60
Cluster 3	0.00	0.78	0.44	0.00

Table 4-6: Results for wb:0.0 wd:0.5 wc:0.5

	Class 1 (R)	Class 2 (B)	Class 3 (P)	Class 4 (Y)
Precision				
Cluster 0	1.00	0.00	0.00	0.00
Cluster 1	0.64	0.14	0.00	0.21
Cluster 2	0.00	0.76	0.24	0.00
Cluster 3	0.00	0.40	0.40	0.20
Recall				
Cluster 0	0.10	0.00	0.00	0.00
Cluster 1	0.90	0.09	0.00	0.75
Cluster 2	0.00	0.83	0.75	0.00
Cluster 3	0.00	0.09	0.25	0.25
F-Measure				
Cluster 0	0.18	0.00	0.00	0.00
Cluster 1	0.75	0.11	0.00	0.33
Cluster 2	0.00	0.79	0.36	0.00
Cluster 3	0.00	0.14	0.31	0.22

Table 4-7: Results for wb:0.0 wd:0.0 wc:1.0

	Class 1 (R)	Class 2 (B)	Class 3 (P)	Class 4 (Y)
Precision				
Cluster 0	0.20	0.20	0.60	0.00
Cluster 1	0.27	0.52	0.09	0.12
Cluster 2	0.00	0.83	0.17	0.00
Cluster 3	0.00	0.00	1.00	0.00
Recall				
Cluster 0	0.10	0.04	0.38	0.00
Cluster 1	0.90	0.74	0.38	1.00
Cluster 2	0.00	0.22	0.13	0.00
Cluster 3	0.00	0.00	0.13	0.00
F-Measure				
Cluster 0	0.13	0.07	0.46	0.00
Cluster 1	0.42	0.61	0.15	0.22
Cluster 2	0.00	0.34	0.14	0.00
Cluster 3	0.00	0.00	0.22	0.00

Table 4-8: Results for wb:0.5 wd:0.5 wc:0.0

	Class 1 (R)	Class 2 (B)	Class 3 (P)	Class 4 (Y)
Precision				
Cluster 0	1.00	0.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00
Cluster 2	0.00	0.00	0.00	1.00
Cluster 3	0.00	0.00	1.00	0.00
Recall				
Cluster 0	1.00	0.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00
Cluster 2	0.00	0.00	0.00	1.00
Cluster 3	0.00	0.00	1.00	0.00
F-Measure				
Cluster 0	1.00	0.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00
Cluster 2	0.00	0.00	0.00	1.00
Cluster 3	0.00	0.00	1.00	0.00

Table 4-9: Results for wb:0.5 wd:0.0 wc:0.5

	Class 1 (R)	Class 2 (B)	Class 3 (P)	Class 4 (Y)
Precision				
Cluster 0	0.26	0.53	0.13	0.08
Cluster 1	0.00	0.33	0.33	0.33
Cluster 2	0.00	1.00	0.00	0.00
Cluster 3	0.00	0.00	1.00	0.00
Recall				
Cluster 0	1.00	0.87	0.63	0.75
Cluster 1	0.00	0.04	0.13	0.25
Cluster 2	0.00	0.09	0.00	0.00
Cluster 3	0.00	0.00	0.25	0.00
F-Measure				
Cluster 0	0.42	0.66	0.22	0.14
Cluster 1	0.00	0.08	0.18	0.29
Cluster 2	0.00	0.16	0.00	0.00
Cluster 3	0.00	0.00	0.40	0.00

Table 4-10: Results for wb:1.0 wd:0.0 wc:0.0

	Class 1 (R)	Class 2 (B)	Class 3 (P)	Class 4 (Y)
Precision				
Cluster 0	0.38	0.62	0.00	0.00
Cluster 1	0.00	0.64	0.00	0.36
Cluster 2	0.00	0.00	1.00	0.00
Cluster 3	0.00	0.00	1.00	0.00
Recall				
Cluster 0	1.00	0.70	0.00	0.00
Cluster 1	0.00	0.30	0.00	1.00
Cluster 2	0.00	0.00	0.88	0.00
Cluster 3	0.00	0.00	0.13	0.00
F-Measure				
Cluster 0	0.56	0.65	0.00	0.00
Cluster 1	0.00	0.41	0.00	0.53
Cluster 2	0.00	0.00	0.93	0.00
Cluster 3	0.00	0.00	0.22	0.00

Having all these results at hand, it is not obvious to determine which set of parameters is better. Therefore, we resort to a summarization of these tables. We have extracted the average values of the different sets of parameters for precision, recall and F-Measure. The metric that gave us a clear result was the average F-measure which can be found in Table 4-10 in the right column. In the left column of table 4.10 there are the values for the different sets of parameters. Wb refers to the weights that we assign to the birth date. Wd refers to the weights that we assign to the death date. Wc is related with the weights that we assign to the heartbeat of changes. The bigger the value of average F-measure is the better, because that means that clusters contain objects of one class for their most part.

Table 4-11: Average F-Measure

Parameters Set wb-wd-wc	Average F-Measure
0.33 - 0.33 - 0.33	0.19
0.00 - 1.00 - 0.00	0.22
0.00 - 0.50 - 0.50	0.20
0.00 - 0.00 - 1.00	0.17
0.50 - 0.50 - 0.00	**0.25**
0.50 - 0.00 - 0.50	0.16
1.00 - 0.00 - 0.00	0.21

As we can observe, the best results are given by the set of parameters that do not take the changes into account. This was also the case for the classifying process too. For weights equal to 0.5, 0.5, 0.0 for birth, death and changes respectively, the measures give us the best results. One cluster contains objects that belong only to one class something that was depicted from precision measure. Also, each cluster not only contains objects of one class but contains the whole set of these objects, a knowledge that was extracted from recall measure.

The results that are related with the entropies that we obtained with different parameters for all the datasets are presented in Tables 4.11 to 4.14. The first three columns are related with the weights that were assigned to the distance function. The first column, w_b refers to the weights that were assigned to the birth date, the second column w_d refers to the weights that were assigned to the death date and the third column w_c is connected with the weights that were assigned to the heartbeat of changes. Finally the bold values denote the best score for each dataset.

Table 4-12: Atlas results

w_b	w_d	w_c	Entropy (e)
0.333	0.333	0.333	0.40
0	1	0	0.45
0	0.5	0.5	0.51
0	0	1	1.14
0.5	0.5	0	0.32
0.5	0	0.5	0.50
1	0	0	**0.30**

Table 4-13: bioSQL results

w_b	w_d	w_c	Entropy (e)
0.333	0.333	0.333	1.13
0	1	0	0.79
0	0.5	0.5	1.06
0	0	1	1.14
0.5	0.5	0	**0.00**
0.5	0	0.5	0.57
1	0	0	0.52

Table 4-14: Coppermine results

w_b	w_d	w_c	Entropy (e)
0.333	0.333	0.333	0.38
0	1	0	0.19
0	0.5	0.5	0.38
0	0	1	0.38
0.5	0.5	0	0.19
0.5	0	0.5	0.60
1	0	0	**0.00**

Table 4-15: phpBB results

w_b	w_d	w_c	Entropy (e)
0.333	0.333	0.333	0.13
0	1	0	0.94
0	0.5	0.5	0.28
0	0	1	0.28
0.5	0.5	0	**0.00**
0.5	0	0.5	0.20
1	0	0	0.26

Now, we discuss the circumstances under which our algorithm gives the best results. In the Atlas dataset the best entropy is given for the set of weights (1, 0, 0) whereas in second place we find the set (0.5, 0.5, 0). However, the interesting here is that when the set of parameters (0.5, 0.5, 0) "looses", it has close result with the winner, whereas when it "wins" it has a much bigger distance from the second. That is the reason that we choose this set of parameterσ as the best alternative. The worst results are come from the set (0, 0, 1) that is the set that does not take note of the birth and death date of each data point which are the

basic criteria that were taken into account for the classification. This pattern seems to be followed from the next three datasets too, where the entropy in some cases is even equal to zero. So we can conclude that our algorithm has a good performance on these datasets and it is quite adaptive as concerns the values of weights.

4.2.3.2 Internal Evaluation

Internal evaluation contains these types of methods that do not need any external knowledge as mentioned previously and they can validate the quality of the clusters with information that is kept by each cluster by itself. Often, internal evaluation helps us to decide the right set of parameters for the best quality of the clustering. Two of the most common metrics for this type of evaluation are cohesion and separation. In general, according to [TaSK05], we can express the overall cluster validity for a set of K clusters as a weighted sum of the validity of individual clusters. This is expressed by this formula:

$$overall\ validity = \sum_{i=1}^{K} w_i validity(C_i)$$

The validity function can be expressed by various metrics such as cohesion, separation or even a combination of them. With regard to the cohesion, higher values are better, whereas lower values are better for separation. Weights vary among different metrics.

Cohesion of a cluster can be defined as the sum of the proximities with respect to the prototype (centroid or medoid) of the cluster. With the term proximity we mean a distance function. So here it is the formula that we used to measure cohesion:

$$cohesion(C_i) = \sum_{x \in C_i} proximity(x, c_i) = distance(x, c_i)$$

where c_i is the prototype of cluster C_i .

The prototype of each cluster was computed as follows:

We considered that the prototype of a cluster is a vector [*x, y, z*]. For our purposes, each data point which is a table of the cluster contains a vector [x, y, z] too, where x is the birth date of the table, y is the death date of the table and z is the number of changes of the table. Now we can define how prototype's vector is calculated. Prototype's *x* value is the average of the x values of the data points that are included by the cluster. *y* value is the average of the y values of the data points that are included by the cluster and *z* value is the average of the z values of the data points.

Respectively, *separation* of a cluster is defined as the proximity between the centroid c_i of the cluster and an overall centroid that has been calculated by the whole set of data points. Here is how it is expressed in mathematical formula:

$$separation(C_i) = proximity(c_i, c)$$

where *c* is defined the overall centroid of the dataset.

We can define overall centroid of the dataset similarly with the prototype of the cluster with the only difference that in this case the entire set of data points take part in calculation of *x*, *y* and *z* values of overall centroid.

For our purposes, we used the Euclidean distance as a measure of proximity. The weight for cohesion was set to 1, whereas weight for separation was set to the number of objects of each cluster, because separation combines both information from the cluster itself and information from the whole dataset, so we would like each cluster to reflect to the final result depending on its size. Now, we can see the results about some concrete values for the set of the parameters. In the Wb column there are the values that were given to the birth weight of distance function, whereas in the Wd column there are the values that are related with the death weight and in the Wc column the values that are connected with the changes respectively. In the Cohesion column there are the results for the cohesion metric and in column Separation the results for separation metrics.

Table 4-16: Atlas Dataset Results

Wb	Wd	Wc	Cohesion	Separation
0.00	1.00	0.00	1323.69	2115.12
0.33	0.33	0.33	650.45	2598.62
0.50	0.50	0.00	331.31	2797.45
0.50	0.00	0.50	**1383.74**	**2049.81**
1.00	0.00	0.00	1271.30	2314.82

Atlas is a dataset that does not have a lot of deletions of tables compared to insertions and changes, so the set of parameters that is more connected with the birth and the changes and not so much with the deaths give us the best results with respect to cohesion and separation. In second place comes the set which is related with only the deaths because there is a big cluster of tables that die in very close versions and the larger percentage of tables live until the end of the database, which means that have the same death date.

Table 4-17: Coppermine Dataset

Wb	Wd	Wc	Cohesion	Separation
0.00	1.00	0.00	**211.33**	**362.10**
0.33	0.33	0.33	35.62	421.41
0.50	0.50	0.00	40.16	418.65
0.50	0.00	0.50	35.62	421.41
1.00	0.00	0.00	40.16	418.65

Coppermine's best set of parameters is 0, 0, 1 for birth, death and changes respectively. This happens because Coppermine's dataset contains only one permanent deletion of table and one temporary. This means that the majority of the tables has the same death date and so the best clustering is given by the set of parameters that has to do with only the deaths.

Table 4-18: bioSQL Dataset

Wb	Wd	Wc	Cohesion	Separation
0.00	1.00	0.00	320.45	781.45
0.33	0.33	0.33	159.73	890.94
0.50	0.50	0.00	122.00	886.21
0.50	0.00	0.50	**473.46**	**668.98**
1.00	0.00	0.00	253.59	827.05

The Biosql dataset also could be characterized as an ascending dataset with respect to its size. That means that a big weight to deaths would not give as the best result. As we can see at Table 4-16 this is something that is reflected by the result, because the winner is the set that is connected with the other two parameters births and deaths.

Table 4-19: Ensembl Dataset

Wb	Wd	Wc	Cohesion	Separation
0.00	1.00	0.00	11167.50	30780.37
0.33	0.33	0.33	**21301.49**	**21566.66**
0.50	0.50	0.00	5289.72	33661.45
0.50	0.00	0.50	19684.44	24562.84
1.00	0.00	0.00	14182.14	27347.17

The Ensembl dataset is one of the largest datasets that we evaluated. That means that it has a big number of commits that contain all types of changes such as deletions, insertions and updates among the whole life of the database. This could give the intuition that a more balanced set of weights would give us the best results. Finally, this intuition came true because we have the best cohesion and separation for the set of 0.33, 0.33, and 0.33.

Table 4-20: mwiki Dataset

Wb	Wd	Wc	Cohesion	Separation
0.00	1.00	0.00	4653.55	**6882.07**
0.33	0.33	0.33	1752.76	9397.74
0.50	0.50	0.00	1033.57	9561.50
0.50	0.00	0.50	**5349.92**	7390.00
1.00	0.00	0.00	4775.46	7740.74

Table 4-18 shows the results concerning the mediaWiki dataset. The deletions that are committed in mediaWiki can be grouped with the naked eye, because there are three different groups of tables that they died in independent time points in relation to the lifetime of the database something that explains why this parameter on its own give us the best separation. Although, if we take into account the other two parameters, we can see that they give as better result with regard to cohesion. That happens because this set contains the parameter of the

changes that are not negligible during the whole life of the database combining with the births.

Table 4-21: Opencart Dataset

Wb	Wd	Wc	Cohesion	Separation
0.00	1.00	0.00	3924.06	15890.54
0.33	0.33	0.33	3359.07	16089.72
0.50	0.50	0.00	2366.92	16189.32
0.50	0.00	0.50	**7604.85**	**13317.76**
1.00	0.00	0.00	3202.38	16068.17

The Opencart dataset is another “quiet” dataset with regard to deaths. The most births of the tables occur in some specific points of the database’s life. The largest part of changes happened in the early life of the database and only a few changes committed later. So, these two features we could say that they give us the largest balance between the tables something that is depicted by Table 4-19 too.

Table 4-22: phpBB Dataset

Wb	Wd	Wc	Cohesion	Separation
0.00	1.00	0.00	766.54	2053.20
0.33	0.33	0.33	**2243.29**	**512.37**
0.50	0.50	0.00	506.25	2196.72
0.50	0.00	0.50	2104.33	565.81
1.00	0.00	0.00	506.25	2196.72

The phpBB dataset could be described by calmness until the middle age of its whole life. However, a big amount of deletions, insertions and updates occur at the middle of the life of the database. In our case this instability can be overcome if we take into account all types of changes that have been committed and give a balanced set of the three parameters. That is projected to Table 4-20 where we

can observe that the set of 0.33, 0.33, and 0.33 give us the clustering with the best cohesion and separation.

Table 4-23: typo3 Dataset

Wb	Wd	Wc	Cohesion	Separation
0.00	1.00	0.00	**414.14**	**1096.70**
0.33	0.33	0.33	192.07	1240.47
0.50	0.50	0.00	239.91	1213.34
0.50	0.00	0.50	208.57	1212.90
1.00	0.00	0.00	277.97	1200.29

The Typo3 dataset does not have an important increase of its size from the beginning of the life of the database until the end of it. This means that birth is not the best feature for clustering the tables. The changes are also only a few during the whole life of the database, so it could not a give a specific result for the clustering. On the other hand, there is an important piece of the typo3's life where tables die continuously. Intuitively, this criterion could give us a better clustering result, because it includes some more information against the other two criteria that are almost flat for the whole life of the database. That is why the set of 0, 1, and 0 in Table 4-21 give us the best results for the metrics of the cohesion and separation.

To conclude, table 4-24 shows cumulatively how many times each set of parameters wins. Observe that the winner set is the set that does not take into account the deaths of the tables. This happens because the datasets generally are "quiet" something that means that they have not a lot of changes and if they have we can say that span across many tables. Moreover, usually we have bursts of births in which a group of tables is born simultaneously and more rarely a unique table is observed to be born. This happens more often with the deaths of the tables. So, the changes feature combining with the births give us the best quality of table clusters.

Table 4-24: Number of wins for different sets of parameters

Wb	Wd	Wc	Cohesion	Separation
0.00	1.00	0.00	2	3
0.33	0.33	0.33	2	2
0.50	0.50	0.00	-	-
0.50	0.00	0.50	**4**	**3**
1.00	0.00	0.00	-	-

4.3 Zoom into a specific point of overview

If the PLD has as the x-axis the phases that have been extracted by our algorithm and as the y-axis the tables of the relational schema of a database, then the zooming into a specific point of the PLD gives as a result the changes that have been committed to this table at this specific phase for the transitions that this phase consists of.

In Figure 4.22 which was exported from PPL tool for the loaded dataset of Atlas, we can see an example of such a case. More concretely, we can see the result of zooming into a specific point of the whole overview. The selected cell (orange cell) from the Parallel Lives Diagram refers to the table *hlt_prescale_set* and to the Phase 4. This phase includes three transitions, the transitions 4, 5 and six as we can see. So, as the result of the drilling into this point we have the changes that have been committed to *hlt_prescale_set* at these transitions something that is shown up in Zoom Area.

Figure 4.22: Zoom into a specific point of overview (1)

If the PLD has as the x-axis the phases that have been extracted by our algorithm again, but as y-axis the clusters that have been created by the clustering algorithm, then zooming into a specific point of this type of PLD would give us the lives of the tables of this cluster for the transitions of which the respective phase consists of.

The difference between Figure 4.22 and Figure 4.23 is that at y-axis we have the clusters that have been created by clustering algorithm. So, the selected cell in Parallel Lives Diagram refers to the Phase 47 of the loaded dataset of phpBB database and to the Cluster 13. As a result of zooming into to this point of the total overview of phpBB database we get the part that is shown up to the Zoom Area of Figure 4.23. In this area, we can see the lives of the tables of the Cluster 13 for the transitions that are included in Phase 47.

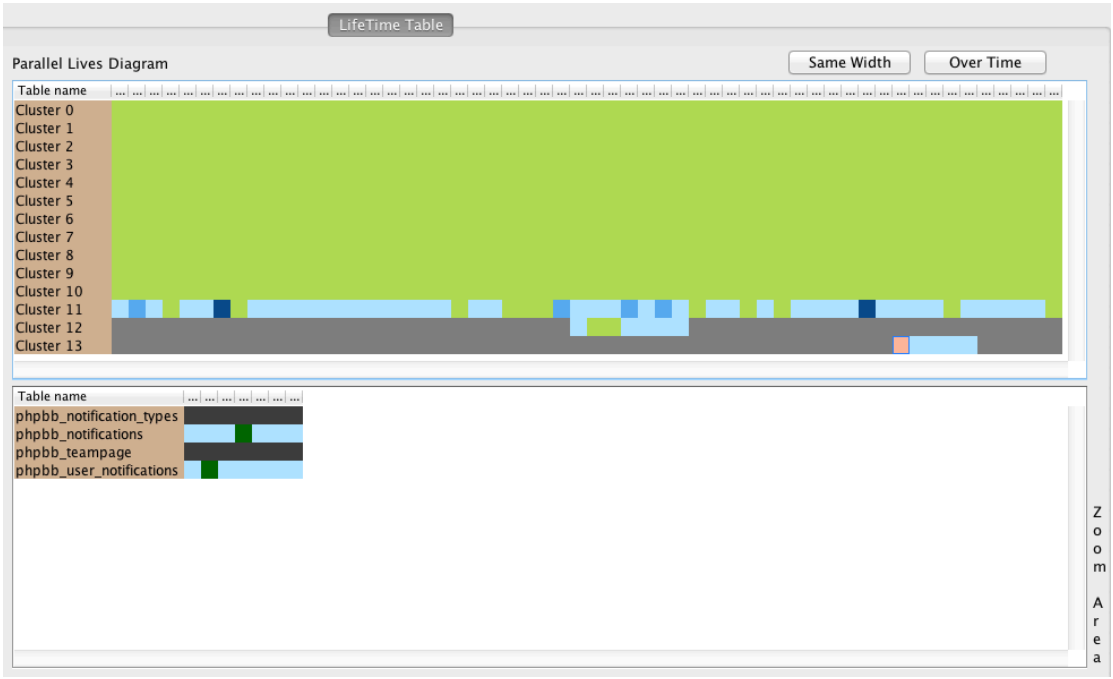


Figure 4.23: Zoom into a specific point of overview (2)

4.4 Filter the overview of the history of a relational database schema

The result of the filtering of a PLD is related with the format that the PLD has. More specifically, if the PLD has as its x-axis the phases that have been extracted, then we can filter our overview by a unique phase and get the behavior of the elements of the y-axis, either this axis consists of the tables of the database or it consists of the clusters that have been created for the transitions of this phase only.

In Figure 4.24, the selected column refers to Phase 47 of the phpBB dataset. The filtering of the total overview according to this phase, give as a result the lives and the changes for every table of the database only for the transitions that constitute this phase.

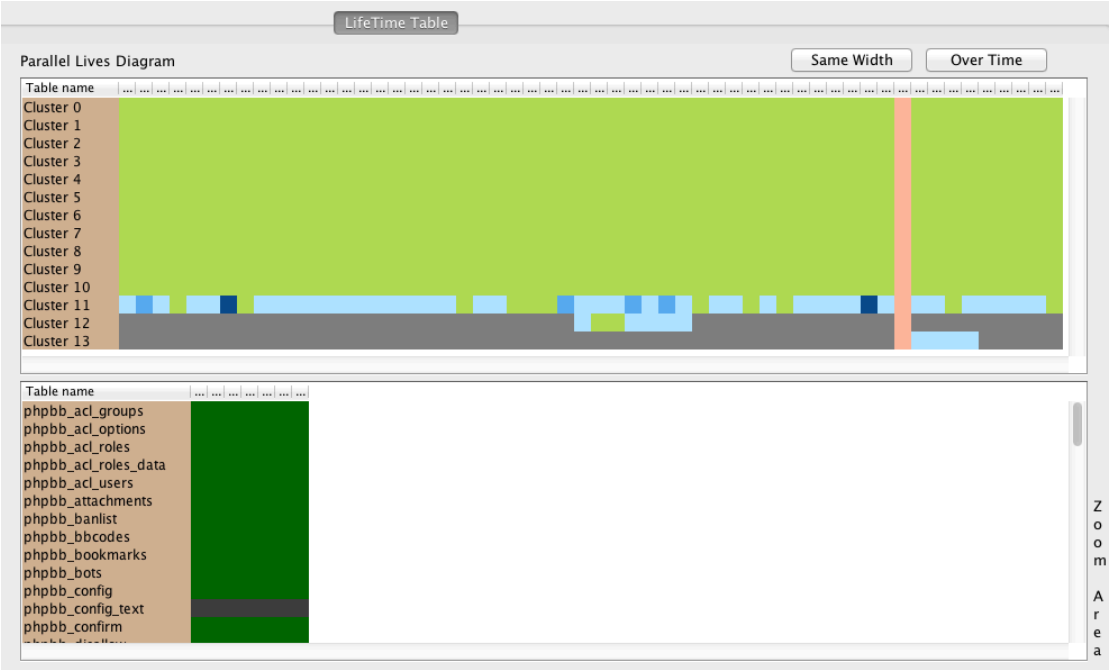


Figure 4.24: Filter by a specific phase

Moreover if the PLD has as its y-axis the tables of a database we can select one or more of them and get the whole life only for the selected.

In Figure 4.25 we can see such an example, where we select the table *phpbb_users* from the PLD and we isolate its entire life. Its life appears to the Zoom Area with the changes that have been committed to this table during its existence.

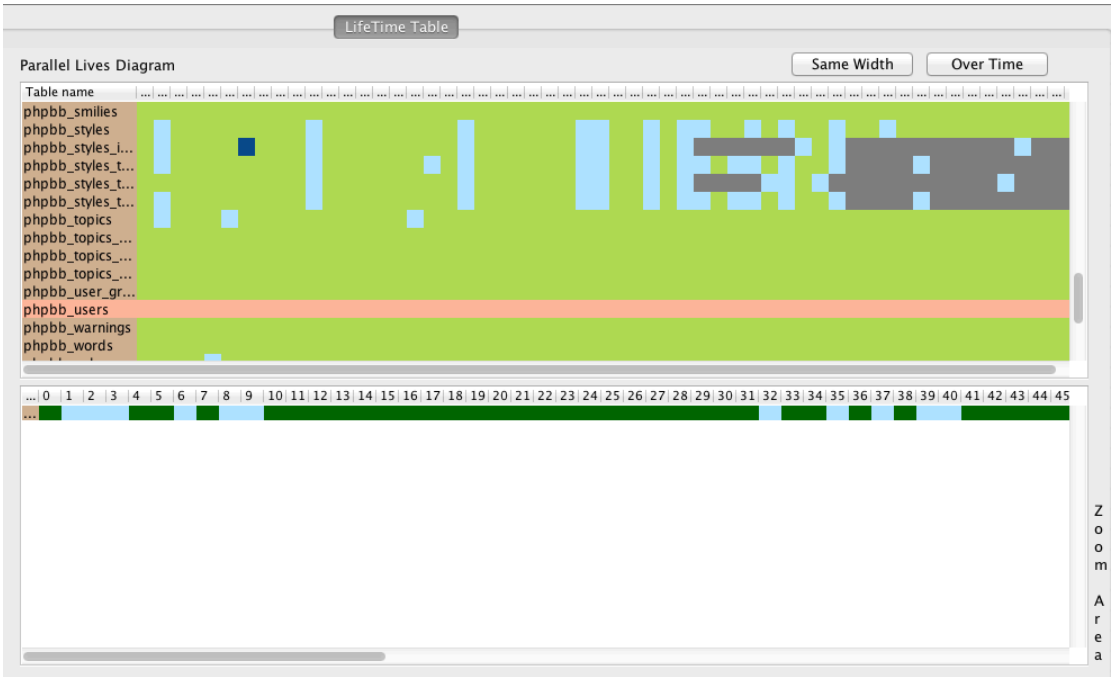


Figure 4.25: Filter by a specific table

If the PLD has as its y-axis the clusters that have been created we can filter our overview only according to the desired clusters and the tables that they consist of and get the behavior only for these.

Similarly, in Figure 4.26 we have two selected rows from the PLD. These rows correspond to Cluster 12 and Cluster 13. So, we can filter the whole overview of the phpBB database according to these rows and get the result that is projected to the Zoom Area of Figure 4.26. This area contains the tables of the selected clusters with their behavior during the whole life of the database, segmented into the same phases such as in PLD.

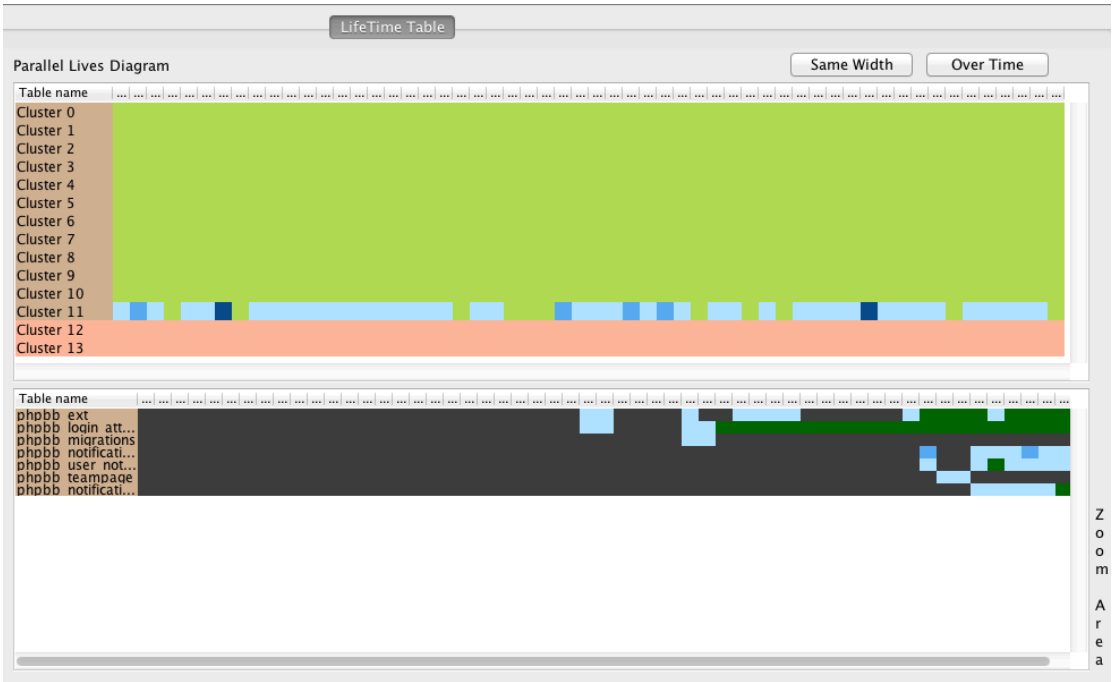


Figure 4.26: Filter by specific clusters

4.5 Details on demand

We can get different kinds of information according to the format that the PLD has. More precisely, if the PLD has as its x-axis the phases, we can select one specific column (phase) and get a more comprehensive description about the selected phase, such as the ID of the transition that starts or ends with, the total number of changes that have been committed to this phase and more specifically the number of updates, additions or deletions for the whole phase.

In Figure 4.27 we can see the details that we get if we select an entire phase (column) from the PLD. In yellow rectangle where there are the details for the selected phase, we can get information such as the name of the selected phase which is Phase 2, the transition ID in which this phase starts which is transition 3 and the transition ID in which this phase ends which is transition 6. Moreover, we are informed that in Phase 2 two changes took place, from which, one was an addition and one was a deletion.

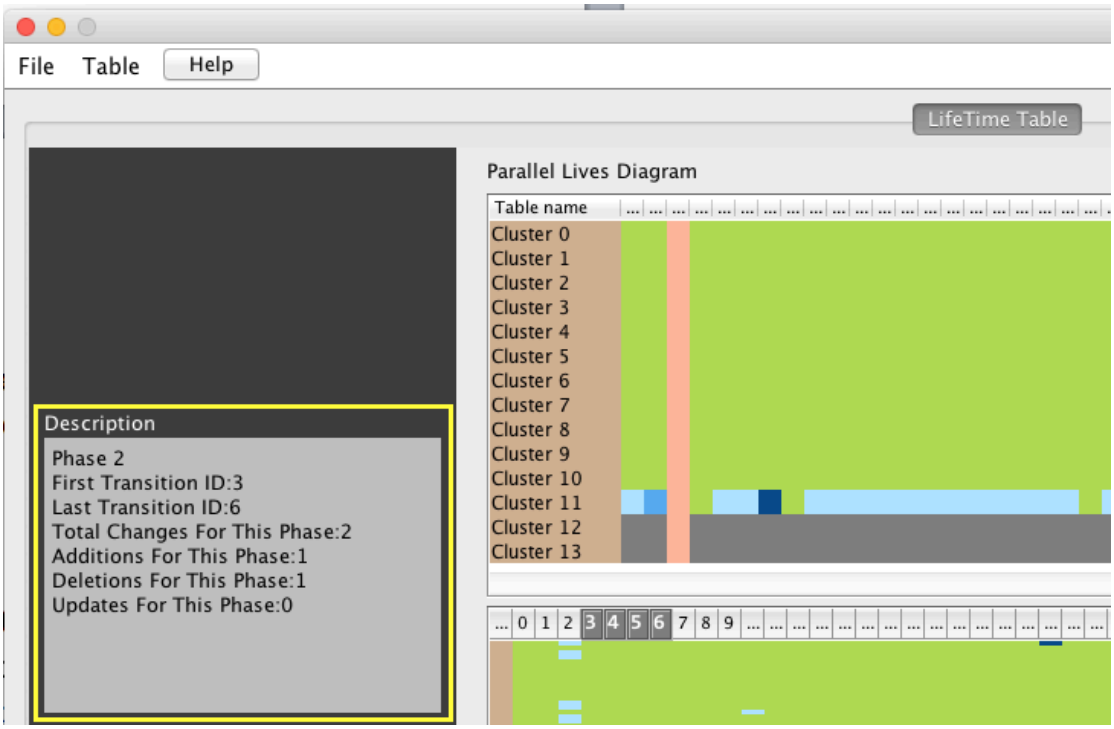


Figure 4.27: Details on demand (1)

Furthermore, if the PLD has in its y-axis the clusters that have been created by the clustering algorithm, we can select a specific cluster and get a detailed description for this, such as the number of tables that it contains, the birth date of the cluster (the smallest birth date of the tables that it includes), the death date of the cluster (the biggest death date of the tables that it includes), the number of the additions, deletions, updates that have been committed to the tables of the cluster, etc.

In the yellow rectangle of the Figure 4.28, we can get details about the selected cluster of the PLD. More concretely, we are informed about the name of this cluster, the name and the transition ID of the birth date of the cluster, the name and the transition ID of the death date of the cluster, the number of tables that are included by this cluster and the total number of changes that have been committed to this cluster during its whole life.

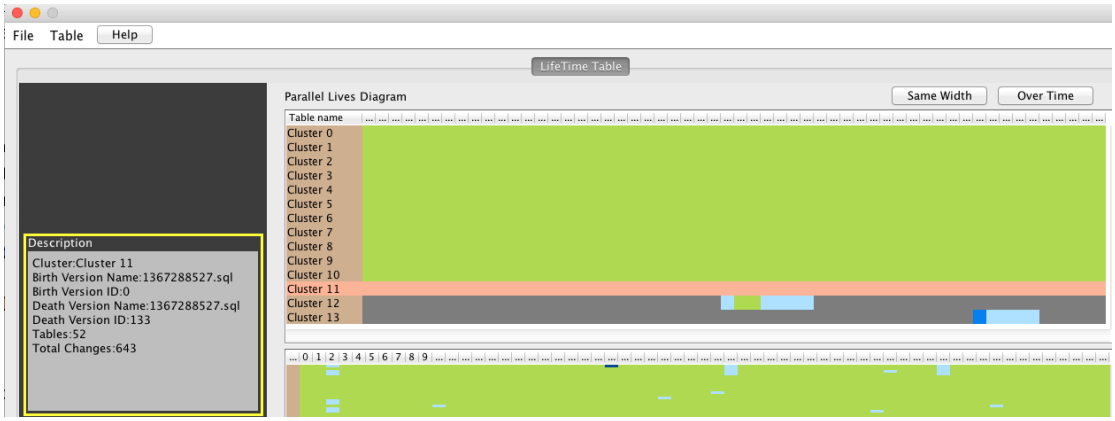


Figure 4.28: Details on demand (2)

If the PLD has as its x-axis the extracted phases and as its y-axis the tables of a database and we select a unique cell, we can get a combination of details for this specific table and for this specific phase, such as when this phase starts/ends, when this table was born, how many changes were committed to this table at this specific phase or what kind of changes were committed to this table in this phase. Similarly, if the y-axis of the PLD consists of the clusters that have been created previously by the clustering algorithm, we can get the respective details for this cluster in relation to this phase.

In Figure 4.29, we have selected a unique cell from a PLD that has the format that have been described above. The yellow rectangle gives us information both for the y-axis value (table) and for the x-axis value (phase). Concerning the phase, we can get details such as the name of the phase, the transition ID of the first transition that it contains and the transition ID of the last transition that it contains. Concerning the table, we can get its name, the sql file and ID of its birth date, the sql file and ID of its death date and the changes that were committed to this table in this phase.

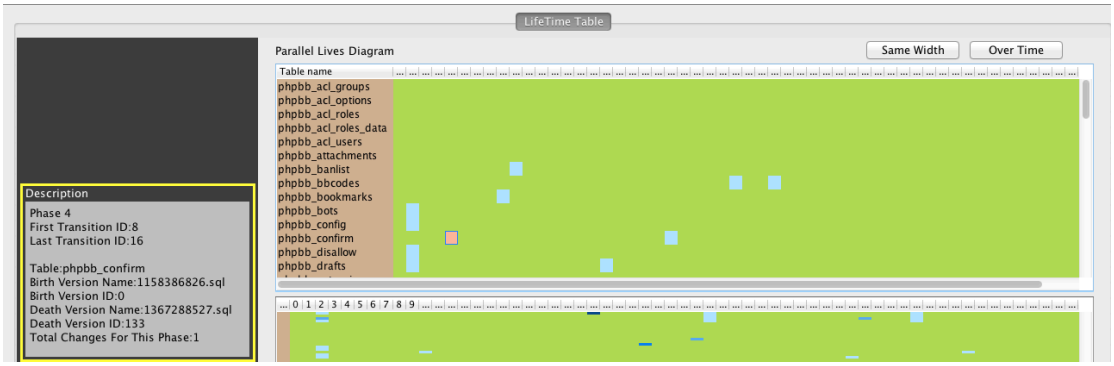


Figure 4.29: Details on demand (3)

Finally, we can get on demand an independent table with a full detailed map of changes that were happened from version to version for every table of the input, with the count and the kind of changes and with the birth and the death of each table during the whole life of the database.

In Figure 4.30 we can see this kind of table. Each row of this table contains the live of one unique table of the dataset that was imported. In this case, we explore the phpBB dataset. There is one column for each version of the database, and between them there are three columns that they refer to the insertions (I), updates (U), and deletions (D). The red color is related with the deletions, the blue color is related with the updates and the green color is connected with the insertions. The deeper a color is, the bigger number of changes for this kind is. For example the red rectangles tell us that from version v1158386826 to version v1158530548 *phpbb_posts* table had one attribute deletion.

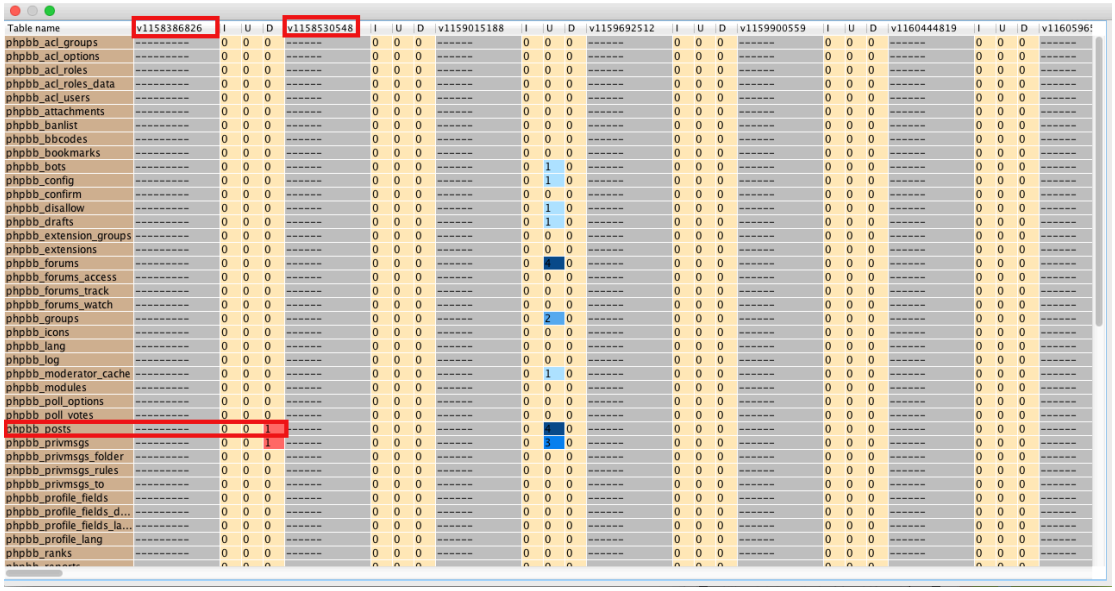


Figure 4.30: Details on demand (4)

CHAPTER 5. SOFTWARE ASPECTS OF OUR SOLUTION – THE PPL TOOL

5.1 Design and Analysis
5.2 Implementation

In this Chapter we will analyze the software aspects of Plutarch's Parallel Lives. In subsection 5.1, we can find the design and the analysis about PPL. Its packages and its classes will be represented with UML package or class diagrams and they will be analyzed. In subsection 5.2, we show information about the implementation of the tool. Moreover, we analyze the technologies and the IDEs that were used and we comment on two selected classes, which are the classes that implement the phase extraction algorithm and the clustering extraction algorithm respectively. Finally, we give some screenshots from Plutarch's Parallel Lives software.

5.1 Design and Analysis

In this subsection we will present the design of our software, the Plutarch's Parallel Lives. PPL consists of a set of packages each of which handles different functions such as input, data processing, data management, graphic user interface, phase analysis or table clustering. In the sequel, we analyze each of this packages including their sub-packages and their classes.

5.1.1 The data Package

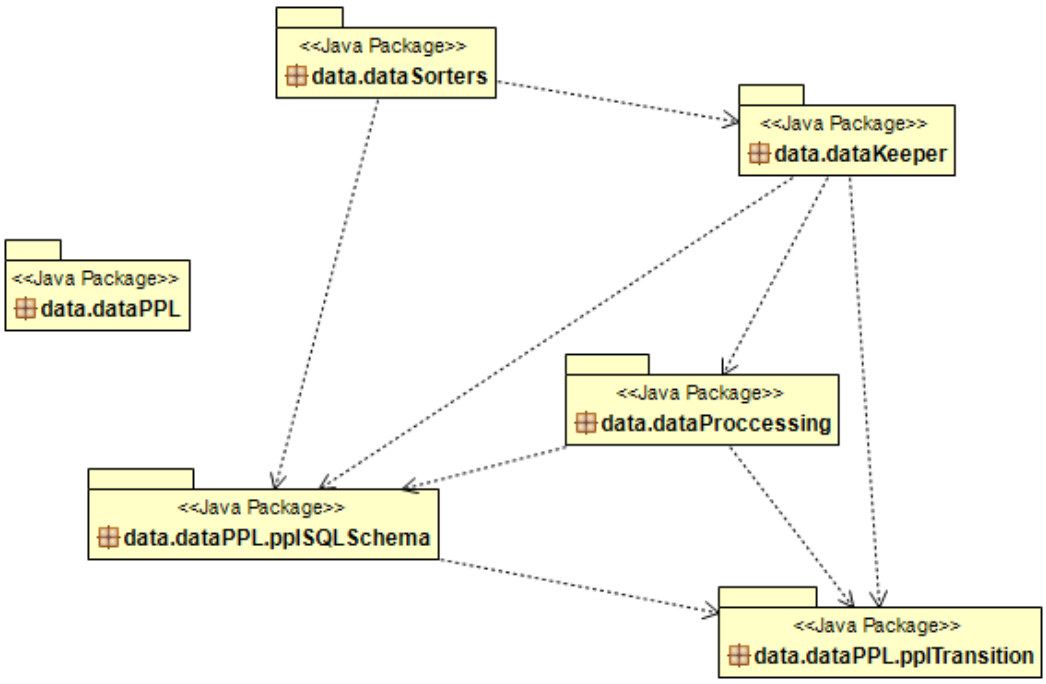


Figure 5.1: data UML Diagram

The `data` package has the main responsibility about whatever is related with the data that are used by our tool. Moreover, it contains the sub-packages `dataKeeper`, `dataPPL`, `dataProcessing` and `dataSorters`.

5.1.1.1 The dataKeeper sub-package

The `DataKeeper` sub-package contains a unique class that is called `GlobalDataKeeper`. This class, as implied by its name, contains all the data that are needed by PPL for its functions. Moreover it keeps the entire input such as tables, schemas, transitions etc. and additionally data that have come from processing input from our algorithms such as phases and clusters. Every other class that needs some information about any of these constructions has to use a `DataKeeper` object of this type of class.

5.1.1.2 The dataProcessing sub-package

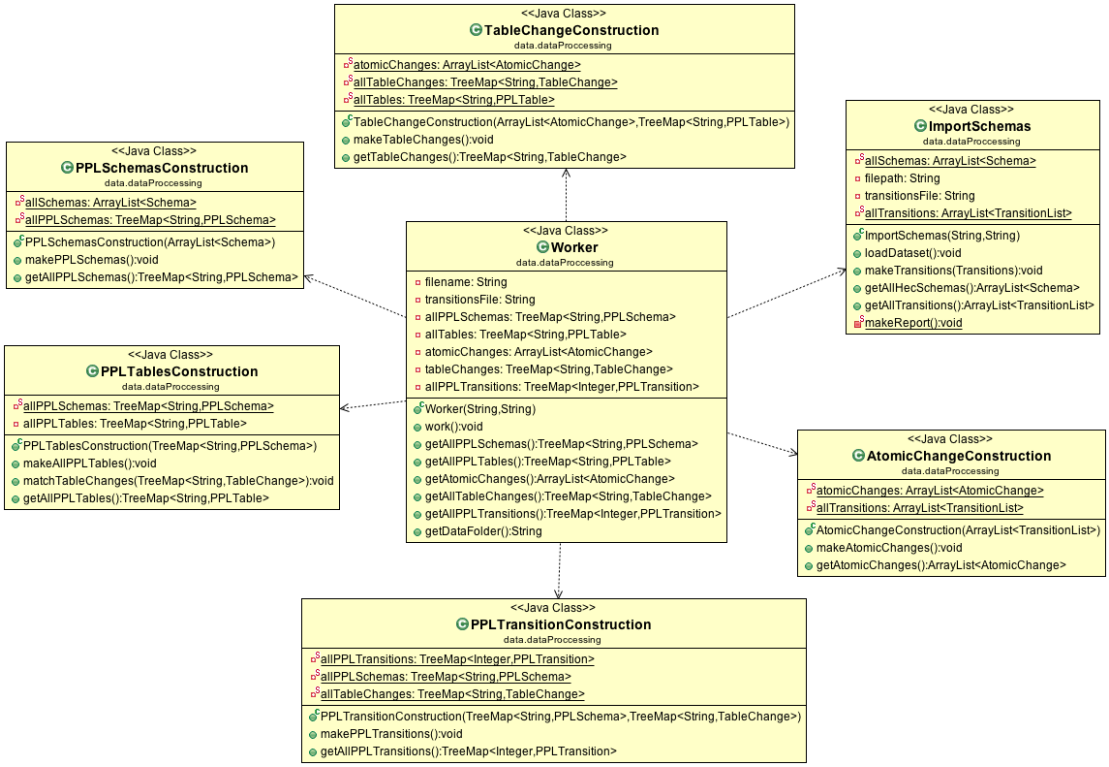


Figure 5.2: dataProccessing UML diagram

This sub-package is responsible for importing the input to the PPL tool and moreover to share it to the relative classes to construct the respective objects.

- Class `Worker` class is the main engine of this package and which handles the rest classes of this sub-package. Initially, it calls `ImportSchemas` class which parses the input with the help of Hecate[3] tool [Heca15] and returns it to `Worker`. Next, the input is shared among the other classes of the package to construct objects which depend on the information that they keep.

[3] Hecate is a tool that gets as input DDL files, parses them and creates sorted lists of relations and the attributes they contain as well as foreign key constrains for the relations. In PPL for each parsed DLL file, it is created an object of type Hecate Schema and includes a list of Hecate Table objects that with their turn they contain a list of Hecate Attribute objects.

- Class `PPLSchemasConstructions` has to construct objects of `PPLSchema` type for each SQL schema was parsed.
- Class `PPLTableConstruction` constructs one object of `PPLTable` type for each table of input.
- Class `PPLTransitionConstruction` makes an object of `PPLTransition` type for each transition that was parsed from the input.
- Class `TableChangeConstruction` construct an object of `TableChange` type for the changes that have been committed to each `PPLTable`.
- Class `AtomicChangesConstruction` class is responsible for the construction of objects of `AtomicChange` type which are individual changes for each Class `PPLTable` and they are fed to the `TableChange's` objects.

5.1.1.3 The dataPPL sub-package

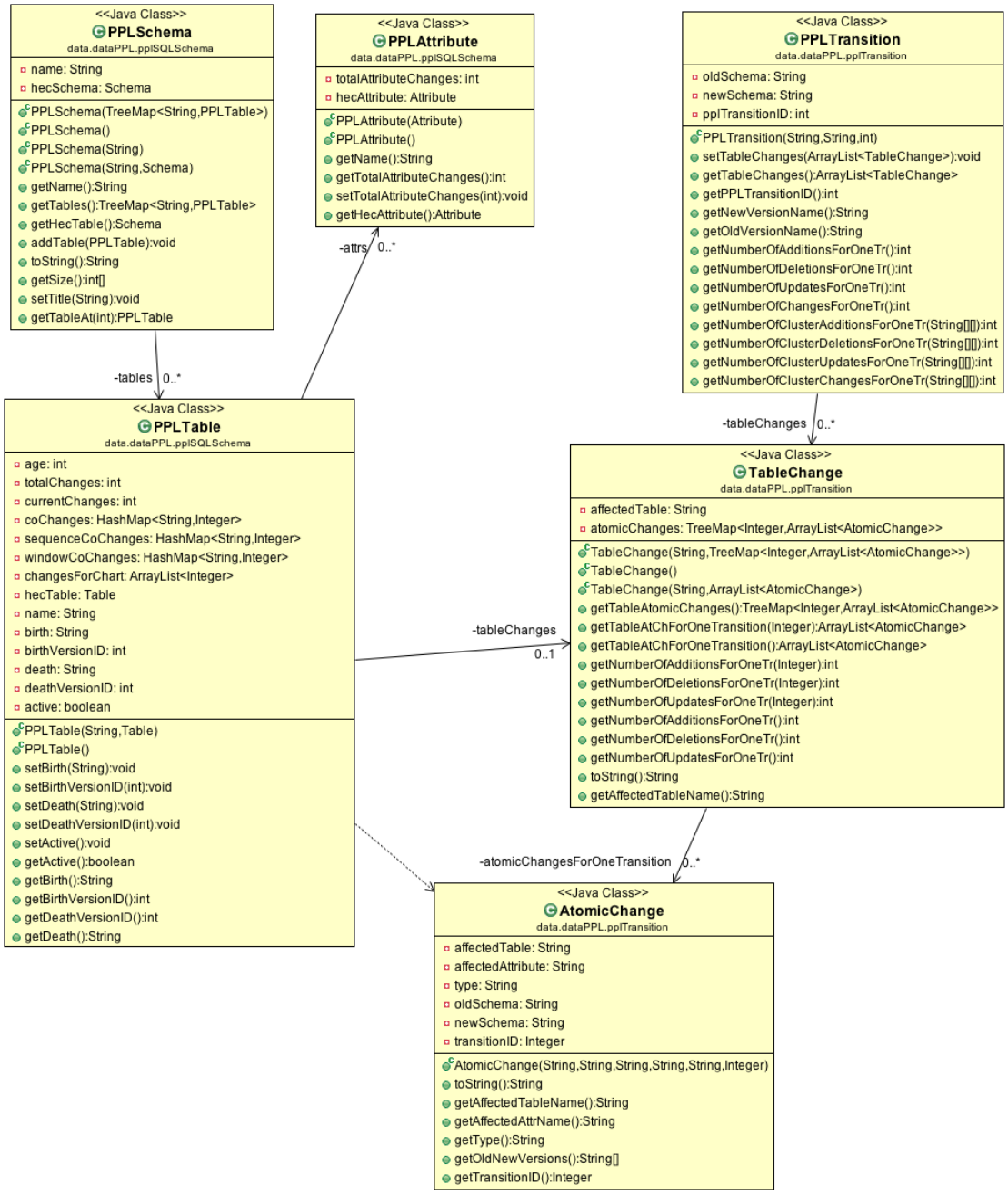


Figure 5.3: dataPPL UML diagram

The `DataPPL` package contains classes that could be characterized as the translation of the input to the objects that PPL understands. It includes two sub-packages according to the function that they handle. The first sub-package is

called `pplSQLSchema` and contains classes that represent the basic features of a database such as schema, table and attribute.

- Class `PPLSchema` keeps the name of an SQL schema, a list of (PPL) tables that are contained to this schema and a reference to a Hecate schema which is the initial format of a schema that was parsed by Hecate.
- Class `PPLTable` keeps all these information that is connected with a table. Moreover, it includes information such as the name of the table, a list of Class `PPLAtributes` which are the attributes of the table, total changes of the table (count and objects), birth dates, death dates, etc.
- Class `PPLAttribute` is the translation of a database's attribute. It contains information about the name of the attribute, the table that belongs to and a reference to a Hecate attribute, which is the initial format of an attribute when the input was parsed by Hecate.

We have to note that some classes keep a reference to Hecate objects because we would not like to keep double information about each object. So each class keeps the information that it needs and together they keep the whole information about a specific object of the database that was given as input.

The second sub-package is called `pplTransition` and it includes classes that keep information that is related with the transitions over versions.

- Class `PPLTransition` is the class that contains the whole information about a transition between two schemas. Moreover, it keeps the names of the old and the new version respectively, an ID that makes a transition unique and a list with the whole set of changes that were happened to this transition.
- Class `TableChange` is responsible for the information that has to do with the changes that were committed to the database. Furthermore it contains the name of the table whose changes are kept, and a list of these changes.
- Class `AtomicChange` class has the responsibility to handle individual changes that were committed to the database. It refers to the attribute

that changed, in which table this attribute was belonged to, in which transition this change was committed and the type of the change, such as insertion, deletion or update.

5.1.1.4 The dataSorters sub-package

This package has to do with the sorting of lists, tables, etc., that have to be sorted by PPL tool. Specifically:

- Class `PPLTableSorting` class handles the sorting of whole set of the tables according to their birth date firstly, and their death date secondly.
- Class `PldRowSorter` is responsible for the sorting of PLD rows, to be sorted by birth date firstly and by death date secondly.

5.1.2 The phaseAnalyzer Package

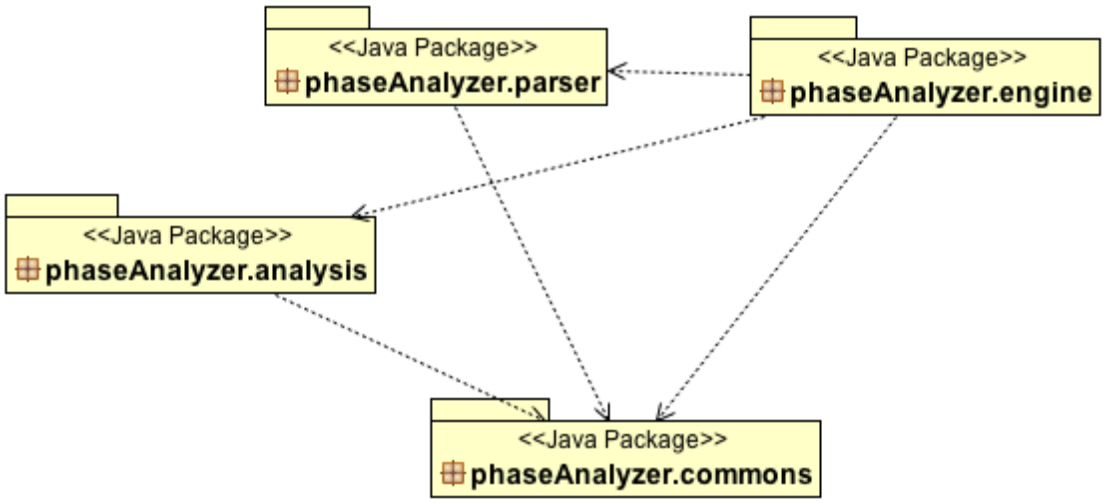


Figure 5.4: phaseAnalyzer UML diagram

The `PhaseAnalyzer` package is connected with those functions that are necessary for the extraction of phases from the life of a database. Let's have a more comprehensive look to them.

5.1.2.1 The engine sub-package

This package as its name denotes is the main engine for the extraction of the phases. It includes one unique class that is called `PhazeAnalyzerMainEngine` which it manages the rest of the classes that are needed to break the life of a database into segments.

5.1.2.2 The parser sub-package

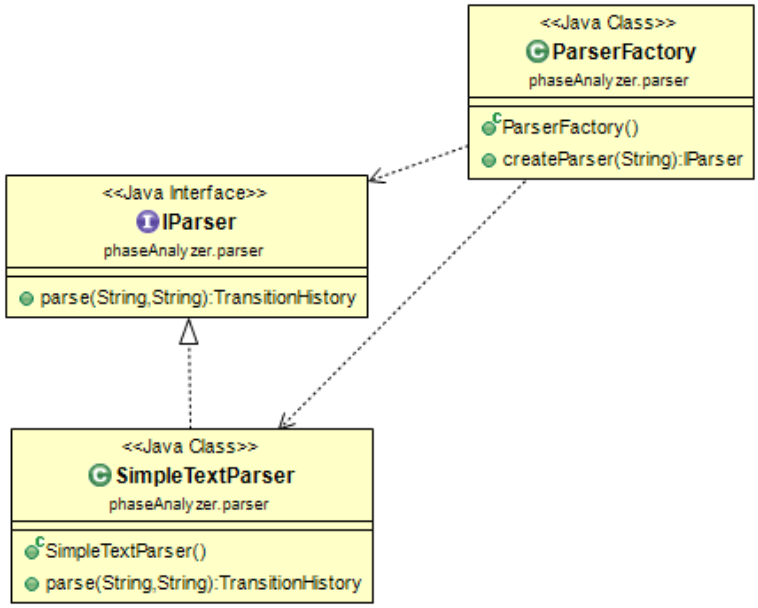


Figure 5.5: parser UML diagram

This package contains the parser of the input that is needed for the extraction of the phases. Moreover it includes three classes.

- Class `IParser` is an interface that has been designed for different kinds of parsers.
- Class `ParserFactory` is a factory of `IParser` objects that returns different kinds of objects of `IParser` type according to the value that it takes as argument.
- Class `SimpleTextParser` is the class that substantially parses the input and transforms it into objects of the relative classes. When it parses a file

it returns an object of `TransitionHistory` type that will be analyzed below. It implements `IParser` interface.

5.1.2.3 The commons sub-package

Figure 5.6: commons UML diagram

The commons sub-package includes all these classes that are used most for the procedure of the extraction of the phases.

- Class `TransitionHistory` is the class that keeps the information of the input for the phase extraction when it is parsed. It contains the whole set of transitions during the life of the database including the stats of each of them.
- Class `TransitionStats` contains the information about the stats of a unique transition such as old or new version of a transition, number of updates, deletions or insertions, etc. For each transition during the life of the database, `TransitionHistory` keeps an object of this type.
- Class Phase is the class that keeps the elements about a unique phase such as when it starts or when it ends, which transitions it includes, number of updates, insertions, deletions, etc.
- Class `PhaseCollector` keeps a list for objects of type Phase. In short, it keeps all the phases that have been extracted by the phase extraction algorithm.

5.1.2.4 The analysis sub-package

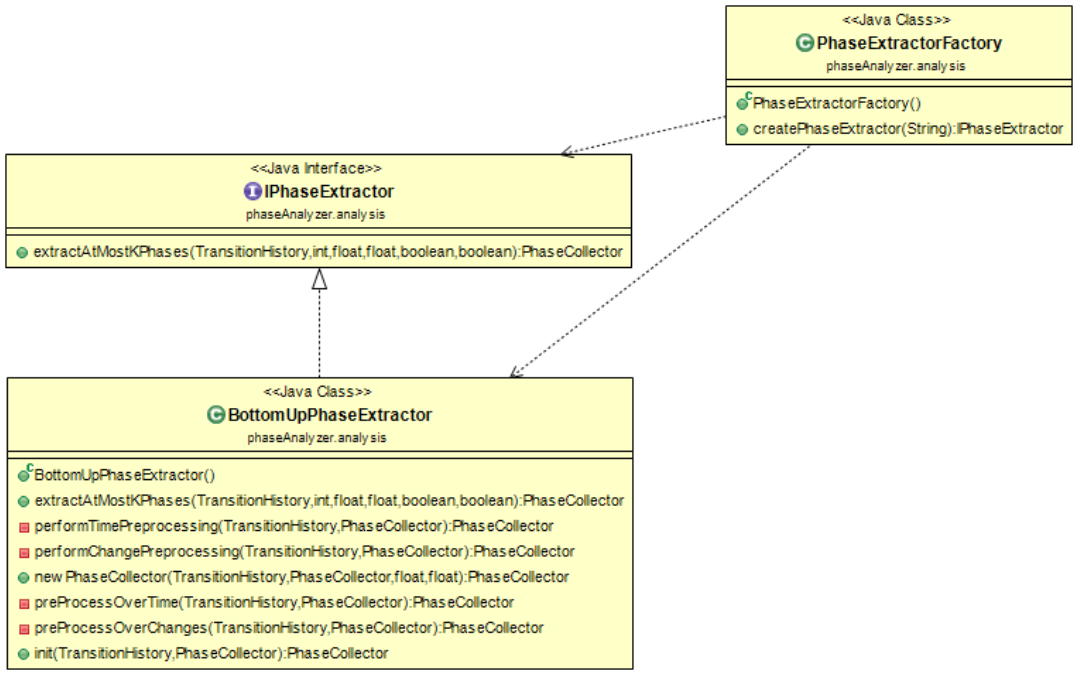


Figure 5.7: analysis UML diagram

The `analysis` sub-package contains classes that have to do with the extraction of the phases.

- Class `PhaseExtractor` is an interface that was designed with this way with the purpose to be implemented by different types of phase extractors.
- Class `PhaseExtractorFactory` is a factory of `PhaseExtractor` objects that it returns different kinds of this type of objects according to the value that is passed as an argument.
- Class `BottomUpPhaseExtractor` is our basic class that contains the algorithm for the extraction of the phases and the preprocessing of the data for the phase extraction. It implements the `PhaseExtractor` interface. We analyze this class much more in section 5.2.2.

5.1.3 The tableClustering package

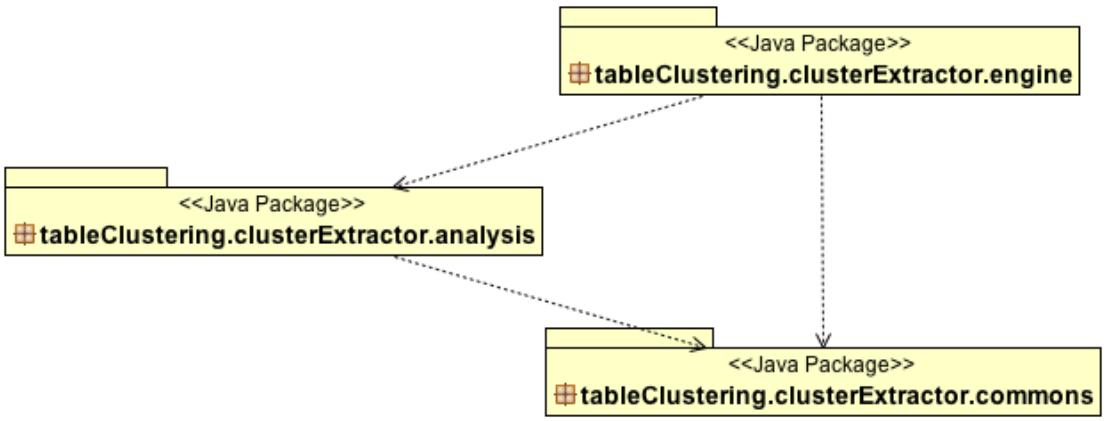


Figure 5.8: tableClustering UML diagram

This package offers the functionality for the construction of the clusters of the tables and the functionality for assessing the validity of these.

5.1.3.1 The engine sub-package

This package contains the main engine which is called `TableClusteringMainEngine` for the clustering of the tables. It manages all these classes that has to do with this procedure.

5.1.3.2 The commons sub-package

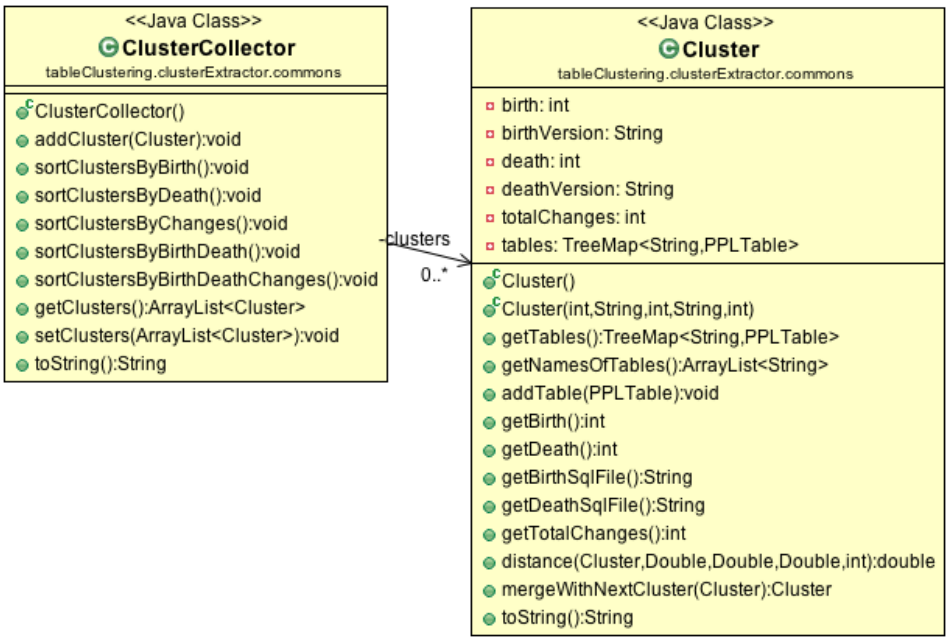


Figure 5.9: commons UML diagram

The `commons` sub-package contains the most used classes for the procedure of the clustering of the tables.

- Class `Cluster` class keeps information for one cluster such as when it was born or when it died, how many changes were committed to this cluster and which tables are included to this.
- Class `ClusterCollector` keeps a list for the whole set of clusters that have been created.

5.1.3.3 The analysis sub-package

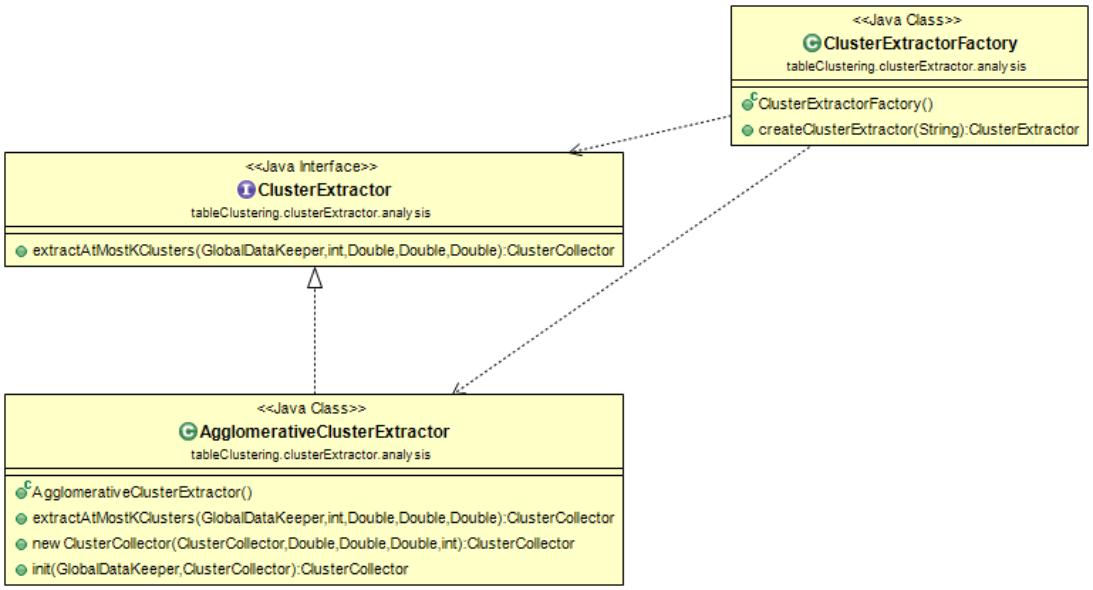


Figure 5.10: analysis UML diagram

This package contains the classes that have to do with the construction of the clusters.

- Class `ClusterExtractor` is an interface, which is implemented by different king of cluster extractors.
- Class `ClusterExtractorFactory` is a factory for constructors of different kind of `ClusterExtractor` objects according to the parameter that is getting as argument.
- Class `AgglomerativeClusterExtractor` is the basic class implementing the procedure of the clustering of the tables. In this class there is the main algorithm that constructs the clusters according to the similarity of the tables. Class `AgglomerativeClusterExtractor` implements the `ClusterExtractor` interface. A more detailed analysis for this class exists in 5.2.2 section.

5.1.4 The gui package

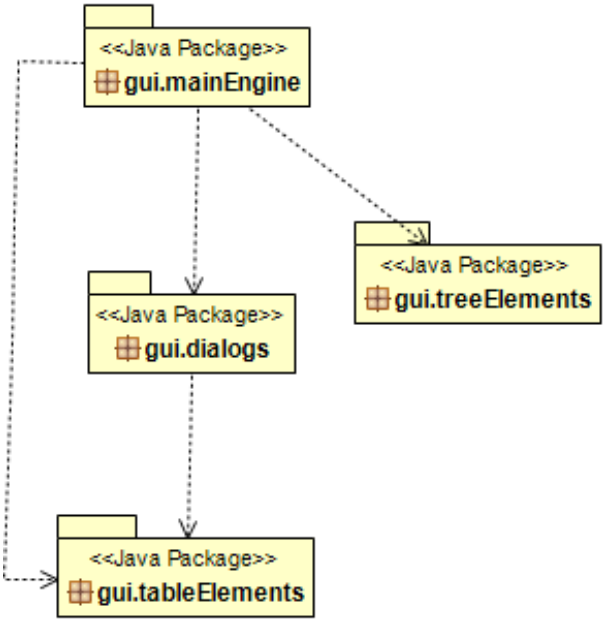


Figure 5.11: gui UML diagram

As implied by its name, this package includes the elements of the graphics user interface.

5.1.4.1 The mainEngine sub-package

The `gui` sub-package contains the main controller for all the software and the graphics use interface which is the class GUI.

5.1.4.2 The dialogs sub-package

The `dialogs` sub-package includes the classes that is related with the dialogs that are shown up in PPL tool.

- Class `CreateProjectJDialog` is the class that is connected with the dialog that appears when a new project either is going to be created, or edited or even loaded.

- Class `EnalargeTable` is the class that has to do with the enlargement of the zoom area into a new `JDialog` window.
- Class `ParametersJDialog` is the class that is related with the `JDialog` that appears to the user to give value to the various parameters either for the phase extraction or for the table clustering.

5.1.4.3 The treeElements sub-package

The `treeElements` package contains those classes that are relative with the construction of the `JTrees` that appear to the graphics user interface.

- Class `TreeConstruction` is an interface that is implemented by different kinds of `JTrees`.
- Class `TreeConstructionGeneral` is a class that implements the `TreeConstruction` interface and returns a `JTree` that is connected with the versions of the database and the tables that they include.
- Class `TreeConstructionPhases` is a class that implements the `TreeConstruction` interface too, but it is related with the construction of a `JTree` relative to the phases and the transitions that they include.
- Class `TreeConstructionPhasesWithClusters` is a class that also implements the `TreeConstruction` interface and it is connected with the clusters and the tables that they include.

5.1.4.4 The tableElements sub-package

This package bothers with the construction of the `JTables` that are come along into the main graphics user interface into the PLD area or Zoom area.

- Class `JvTable` is a class that extends `JTable` class and handles the specification that are needed by our tool such as the height of the row, or the width of the columns. This is the type of all our `JTables` of the PPL tool.

- Class `PLDTableModel` extend the `AbstractTableModel` and it is necessary for the specifications of the data that we have to pass to our kind of `JTables`.
- Class `Pld` is an interface that is implemented by the whole set of those classes that construct data for PPL's `JTables`.
- Class `TableConstructionIDU` is the class into which are constructed the rows and the columns of the PLD `JTable`. It implements `Pld` interface.
- Class `TableConstructionPhases` is the class, which constructs the rows and the columns of the Phases Pld JTable. It implements `Pld` interface.
- Class `TableConstructionWithClusters` is the class, which constructs the rows and the columns of the PhasesWith Clusters Pld JTable. It implements Pld interface.
- Class `TableConstructionPhasesClusterTable` constructs the rows and the columns of the JTable that appears into the zoom area when details for a specific phase and cluster are demanded. It implements Pld interface.
- Class `TableConstructionZoomArea` constructs the rows and the colunms of the `JTable` that appears into the zoom area when a whole column or row is selected on the PLD and was asked to zoom in. It implements Pld interface.

5.2 Implementation

5.2.1 Programming Tools and IDEs

PPL has been designed and implemented with Object Oriented Programming principles, because of the advantages and the flexibility of this kind of programming.

PPL was implemented in programming language Java so the IDE that was selected was one of the most common IDEs in Java programming, Eclipse IDE [Ecli15]. Eclipse is widely known for its frequent updates, in order to be up to date with new technologies and programming techniques. Moreover, there are plenty of plugins that are useful for object oriented programming such as Object Aid plugin that can be used for the production of class or sequence diagrams, or WindowsBuilder that is a helpful tool for the design of graphics user interface.

5.2.2 Selected Classes

In this subsection we will present the most important classes of our tool which are the classes that handle the phase extraction and the clustering of the tables into groups. These two classes are called BottomUpPhaseExtractor and AgglomerativeClusterExtractor.

BottomUpPhaseExtractor is the implementation of Phase Extractor algorithm that was analyzed above in programming language Java. BottomUpPhaseExtractor class is an implementation of the interface IPhaseExtractor which has one unique method to be overridden, the method *extractAtMostKPhases*. This is the first method that we can see below.

This method gets as arguments one object of TransitionHistory class in order to segment the whole history into phases, the desired number of phases, the weights for time and changes that we want to assign to the distance function and two boolean variables if we want the data to be preprocessed according to time or changes. At the beginning of the method, we have to initialize our phases according to Phase Extractor algorithm, so private method *init* is called.

Init method constructs one object of type Phase for each transition. This object has fields such as transition history, start position, end position and total updates

for this phase. All of these objects that have been constructed, were being added to one object of type PhaseCollector that keeps the whole set of phases.

After the initialization of the phases, back to *extractAtMostKPhases* method, next step is to preprocess or not our data over time according to the value of the variable preProcessingTime. If it is true, *performTimePreprocessing* method is being called.

PerformTimePreprocessing calls the *preProcessOverTime* method until no more phases can be merged into one phase and sends the solution back to *extractAtMostKPhases* method. In *preProcessOverTime* method phases that have time distance[4] smaller than three days are merged into one common phase with start position the start position of the phase *i* and end position the end position of the phase *i+1.*

After this part of code, in *extractAtMostKPhases* the equivalent procedure for the preProcessing of the data or not over changes is taken place into *performChangePreprocessing* method where the *preProcessOverChanges* method is being called repetitively until no more phases can be merged into one. The merging procedure merges continuous phases that both of them have zero changes.

After the preprocessing methods, in *extractAtMostKPhases* is being called the most important method that is responsible for the merging of the most similar phases every time until the desired number of phases have been extracted. This private method is called *newPhaseCollector* and gets as arguments the transition history of the dataset, the previous phase collector that keeps the whole set of phases and the weights that we would like to assign to time and to changes for the distance function. It return a new phase collector with the new phases.

---

[4] Actually we talk about the distance between the last transition of the phase *i* and the first transition of the phase *i+1*

NewPhaseCollector computes the distance between every continuous pair of phases. When the computation has been finished, it finds the most similar pair of phases according to distance function that has to be merged. Then, it constructs a new phase collector, which has the same objects as the previous phase collector until the point that it is the turn of the two phases that will be merged to be added. In this point, a new phase is created which has as start position the start position of the first of the two phases that are to be merged and as end position the end position of the second phase. The sum of the changes for both of these phases is set as the number of changes of the new phase. After the creation of this new phase, it is added to the phase collector. Finally, the remaining objects of the previous phase collector, after the second phase they are added too. *NewPhaseCollector* is being called by *extractAtMostKPhases* method until the desired number of methods have been created.

At this point we will analyze the other significant class of PPL tool which is the class that implements the Clustering Extractor algorithm and is called AgglomerativeClusterExtractor.

AgglomerativeClusterExtractor implements the interface ClusterExtractor that has a unique method which is called *extractAtMostKClusters*. This is the method that is overridden by AgglomerativeClusterExtractor and handles the procedures for the cluster extraction.

ExtractAtMostKClusters method gets as arguments one object of GlobalDataKeeper type that contains the whole set of tables that are going to be clustered. The second argument refers to the number of clusters that we desire to be extracted by the Cluster Extractor algorithm. The next three arguments are connected with the distance function and more precisely with the weights that we want to assign to the birth date, the death date and to the number of changes. At the beginning of the method, *init* function is called.

Init function as its name denotes, is responsible for the initialization of the clusters and the construction of the first cluster collector. As it has been

described to the Clustering Extractor algorithm, *init* function constructs one cluster for each table of the dataset. This cluster has as its birth date the birth date of the table that is being transformed into cluster, as its death date the date of the corresponding table and as its changes the changes of the table. All of these clusters after their construction, they are added to the cluster collector which is an ArrayList that keeps the whole set of the clusters.

After the initialization of the clusters, the next step of *extractAtMostKClusters* method is to call repetitively the most significant method of the class, the method *newClusterCollector* until the desired number of clusters will have been extracted. This is the method that is responsible for the merging of the most similar clusters according to the clustering distance function. It takes as arguments the previous cluster collector, the weights that are connected with the distance function and the duration[5] of the dataset and returns as a result a new cluster collector with the new clusters.

The *newClusterCollector* method initially computes the distance functions for every possible pair of clusters that exist to the previous cluster collector. After the computation of the distance functions, it searches for the smallest value of them something that means that it is searching for the most similar pair of clusters. When it finds this, it constructs a new cluster from this pair, which has as its birth date the smaller birth date of these two clusters, as its death date the bigger death date of these two clusters, as its changes the sum of the changes of these clusters and finally as its tables the union of the two sets of tables of the two clusters. After the creation of the new cluster, it is created a new cluster collector that contains the whole set of objects of the previous cluster collector except the two clusters that have been merged into one. Finally the new cluster is added too to the new phase collector and it is returned to the *extractAtMostKClusters* method.

---

[5] Actually duration is the number of transitions

5.2.3 PPL Tool Screenshots

We had the opportunity to present some of the PPL's features in section 4, which had to do with the zooming into a specific point of the PLD, with the filtering of the PLD and with the details on demand about a phase, a cluster, a table or a unique point of the Parallel Lives Diagram. The most significant feature of PPL however, is its ability to provide an overview of the entire PLD. This overview is generated automatically after the creation, the editing or the loading of a project. So, we will cite below two images for each dataset. The upper image is the overview of the imported dataset and the other image is the full detailed PLD of the same dataset or a part of this, according to its size.

Observe that the overviews of each dataset are fairly good approximations of the detailed PLD. There are many points of the detailed PLD that can be recognized to the overview. One of them are the phases which have a lot of changes and more concretely the tables of the imported dataset have a lot of co-changes. These are the columns that have a lot of blue into them. Another observation is that tables that have exactly the same birth and death date can be recognized at the overview PLD because all of them have been clustered to the same cluster which has the same birth and death date with them.

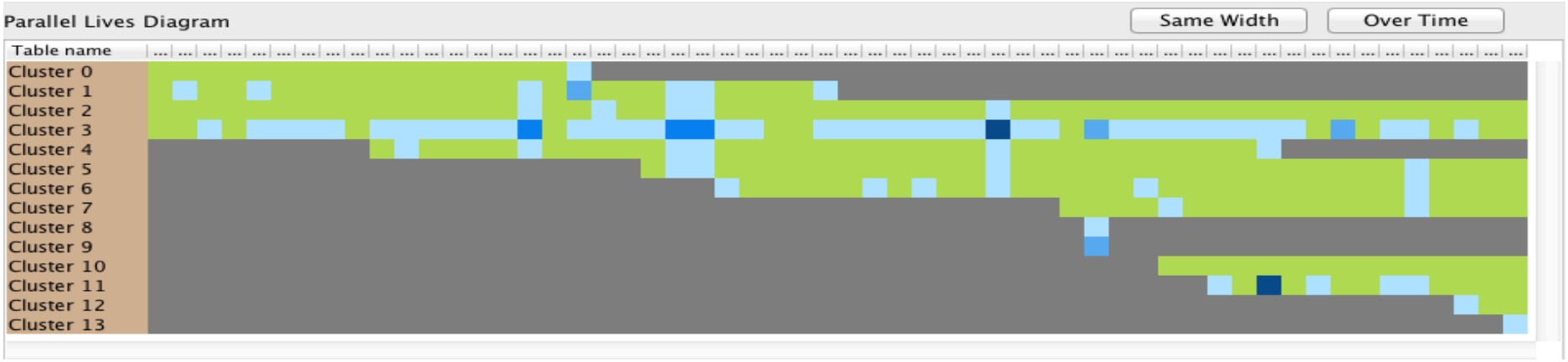


Figure 5.12: Synopsis of Atlas

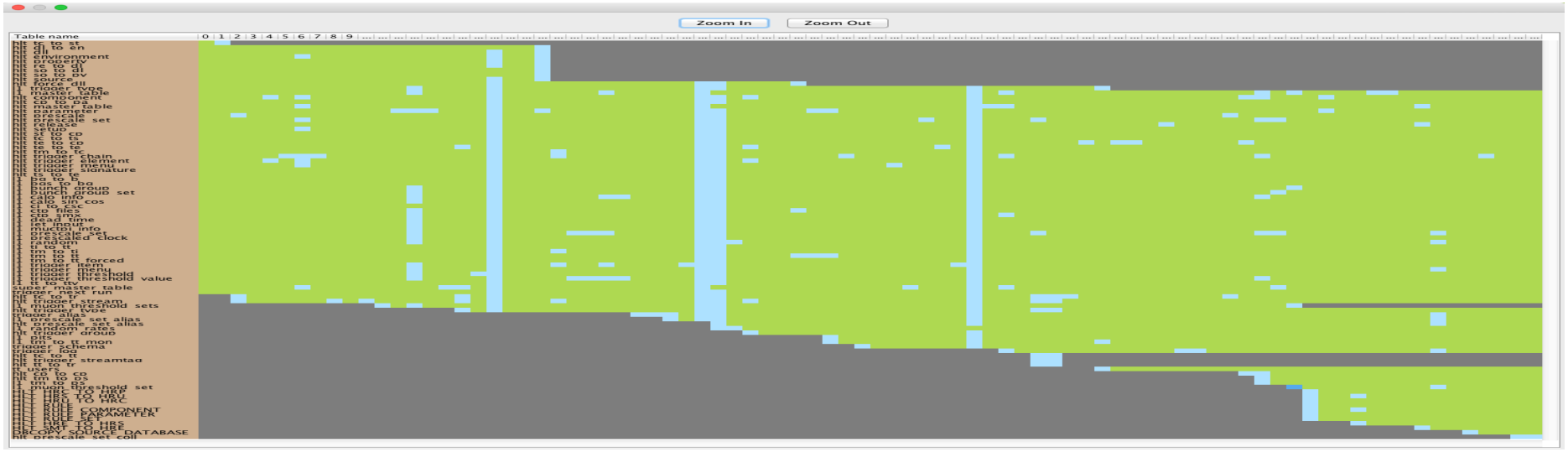


Figure 5.13: PLD of Atlas

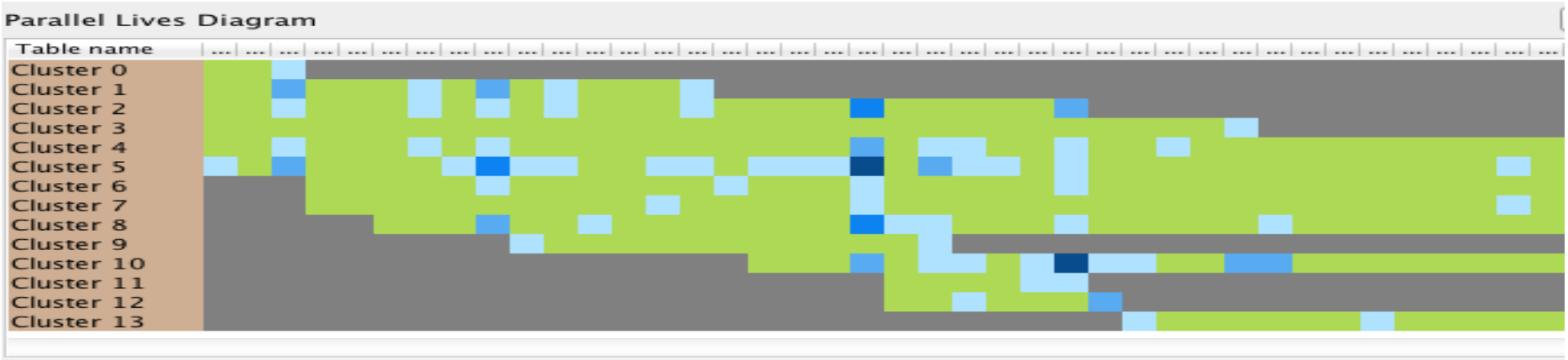


Figure 5.14: Synopsis of bioSQL

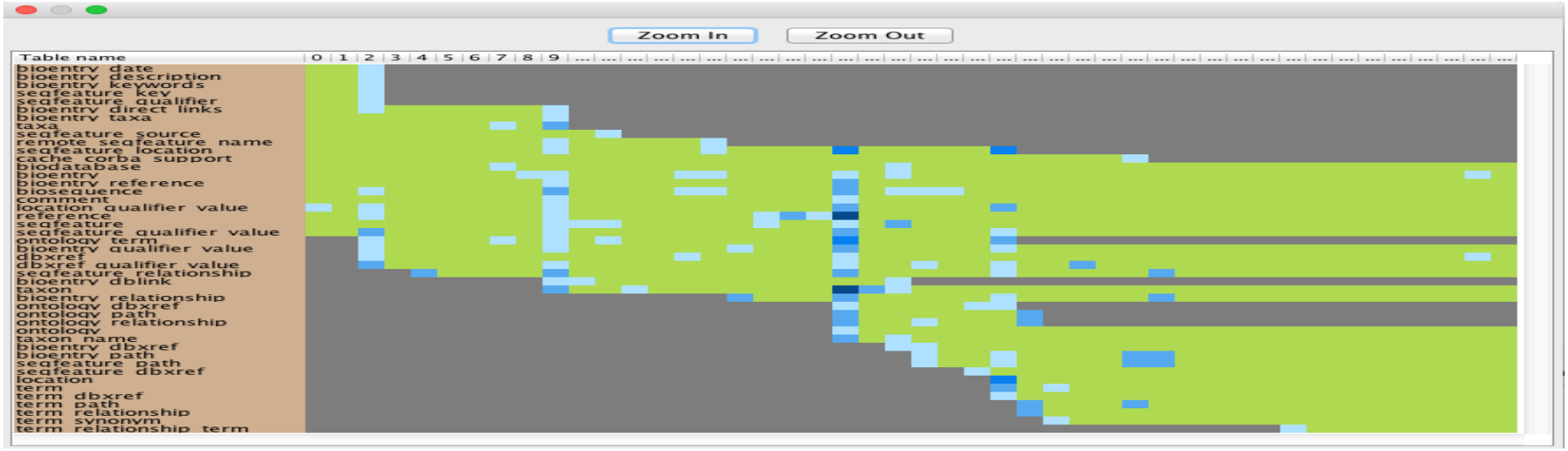


Figure 5.15: PLD of bioSQL

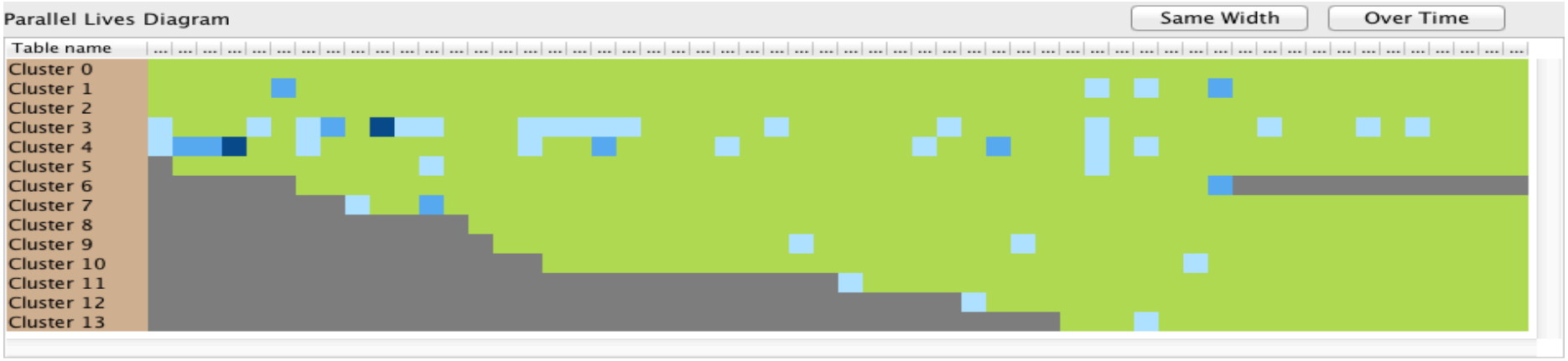


Figure 5.16: Synopsis of Coppermine

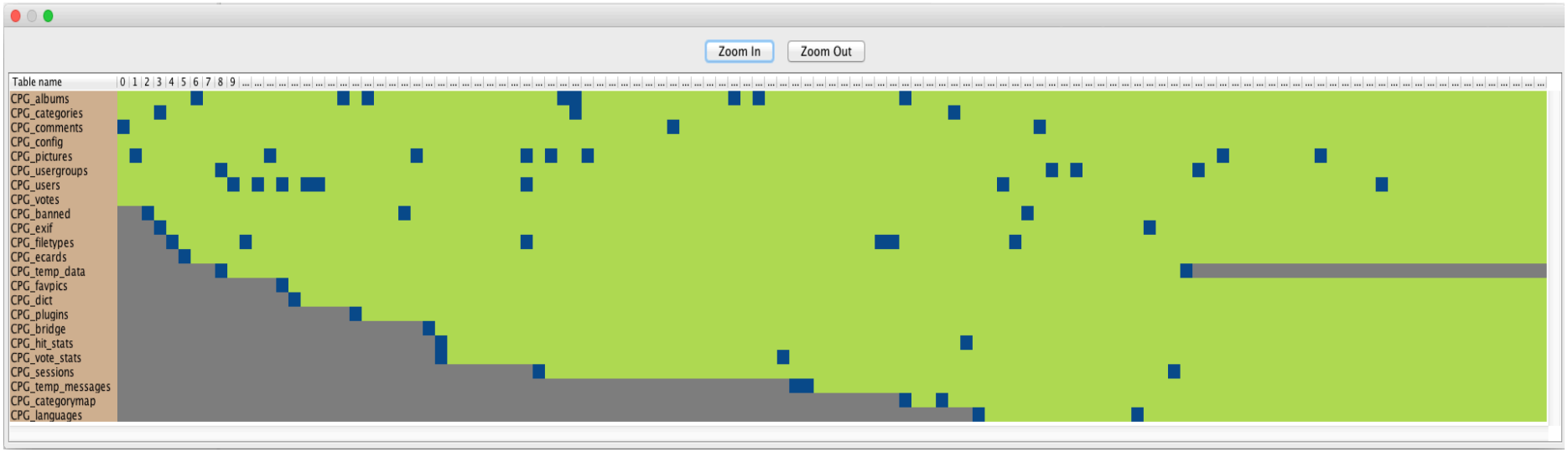


Figure 5.17: PLD of Coppermine

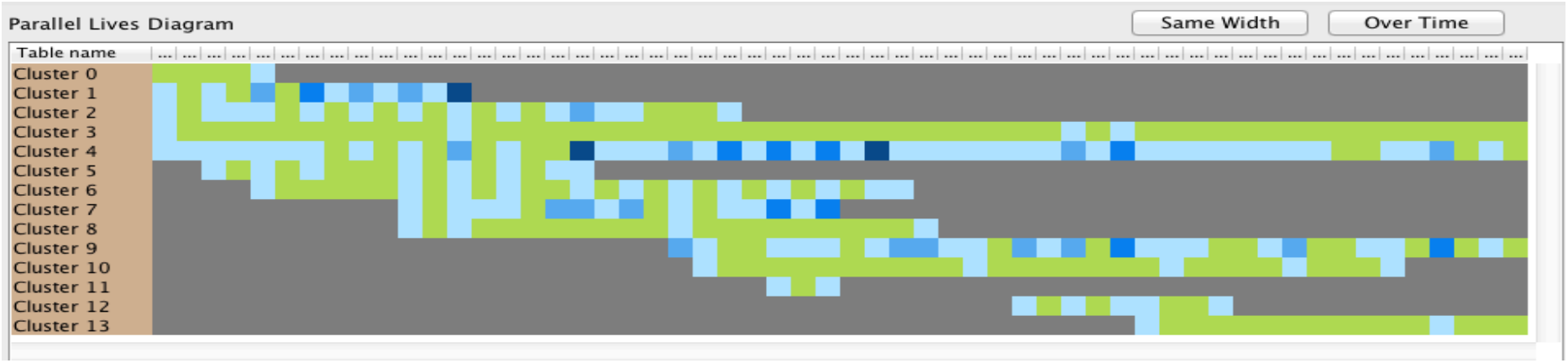


Figure 5.18: Synopsis of Ensembl

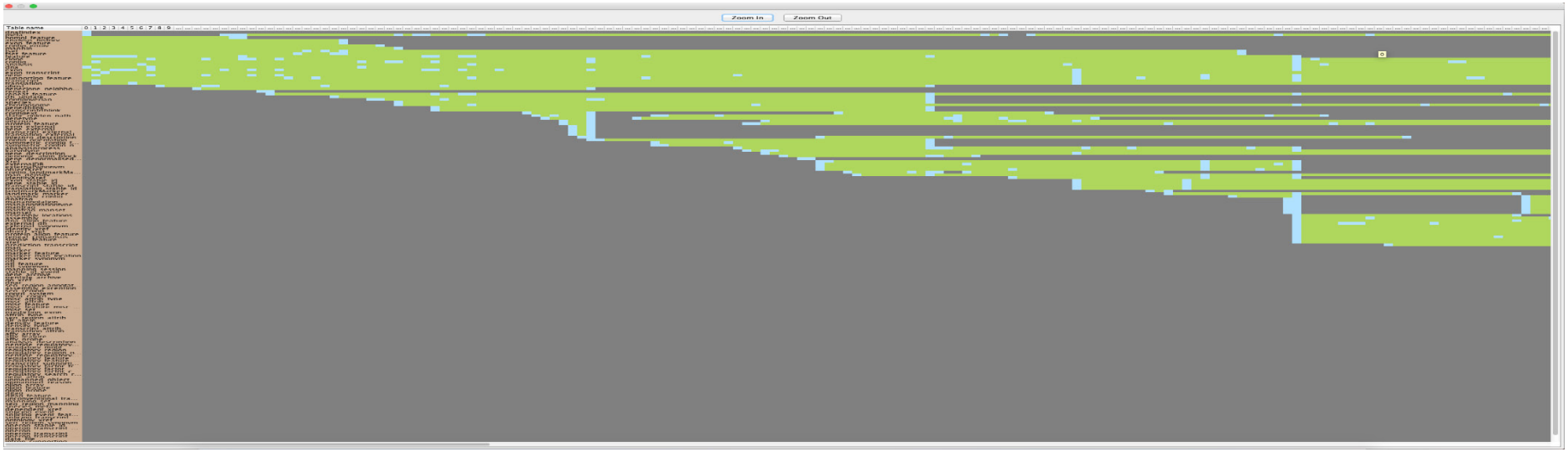


Figure 5.19: A part of PLD of Ensembl

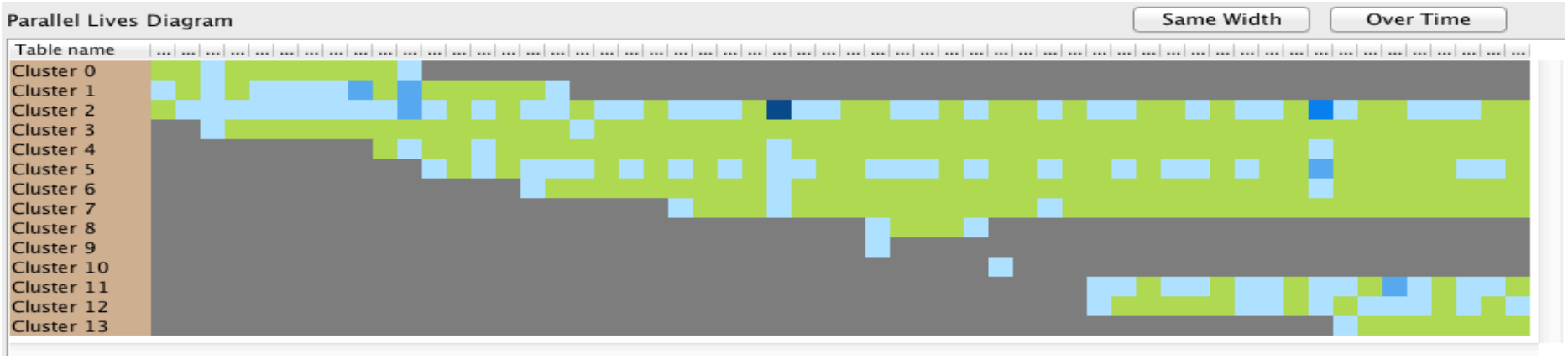


Figure 5.20: Synopsis of mediaWiki

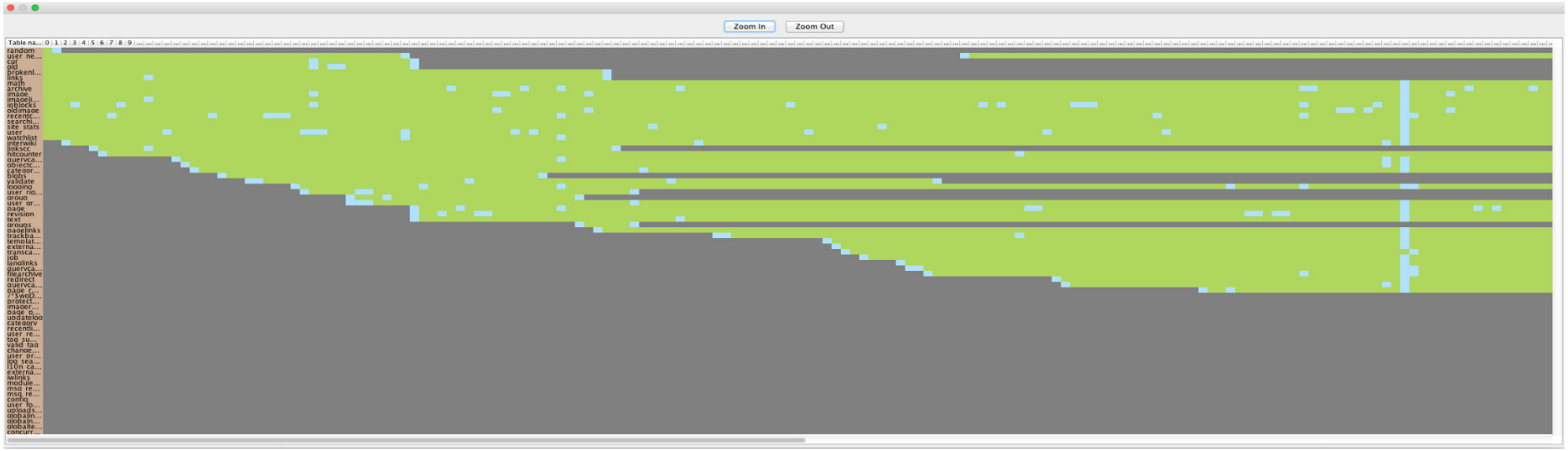


Figure 5.21: A part of PLD of mediaWiki

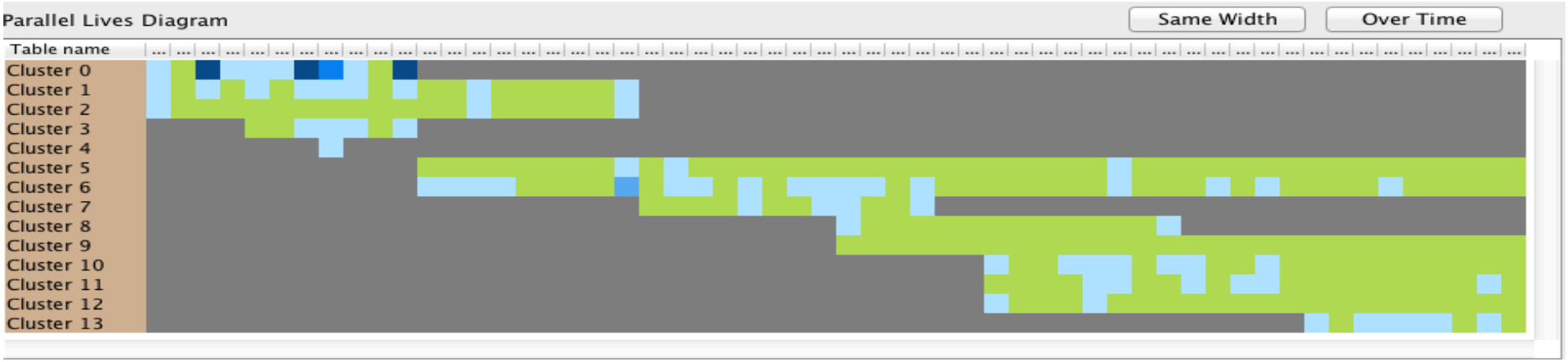


Figure 5.22: Synopsis of Opencart

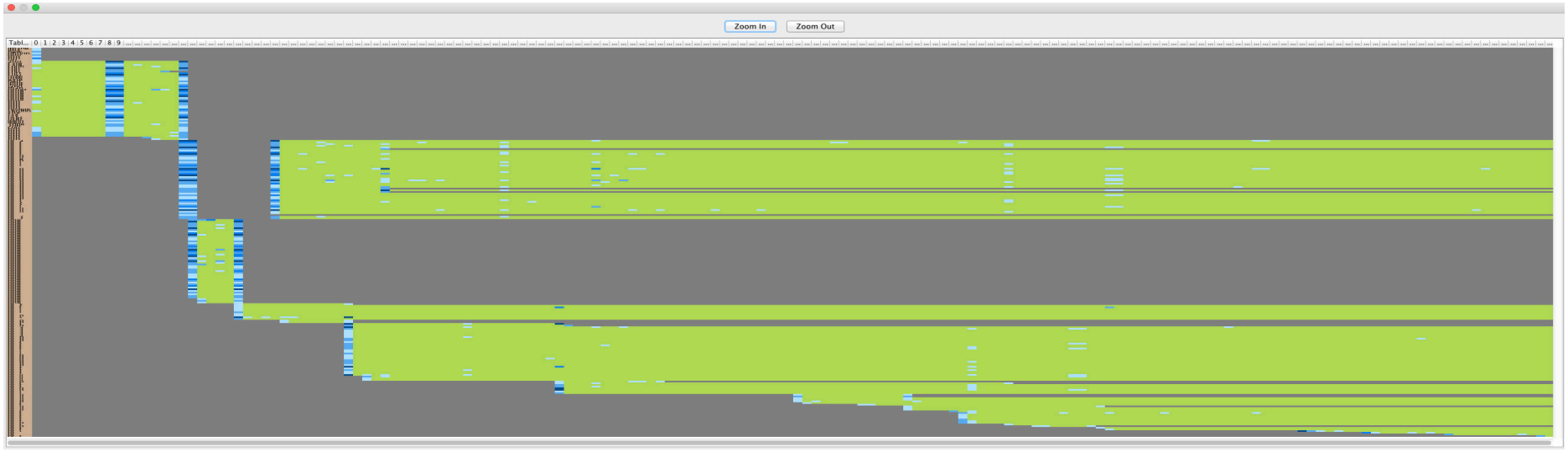


Figure 5.23: PLD of Opencart

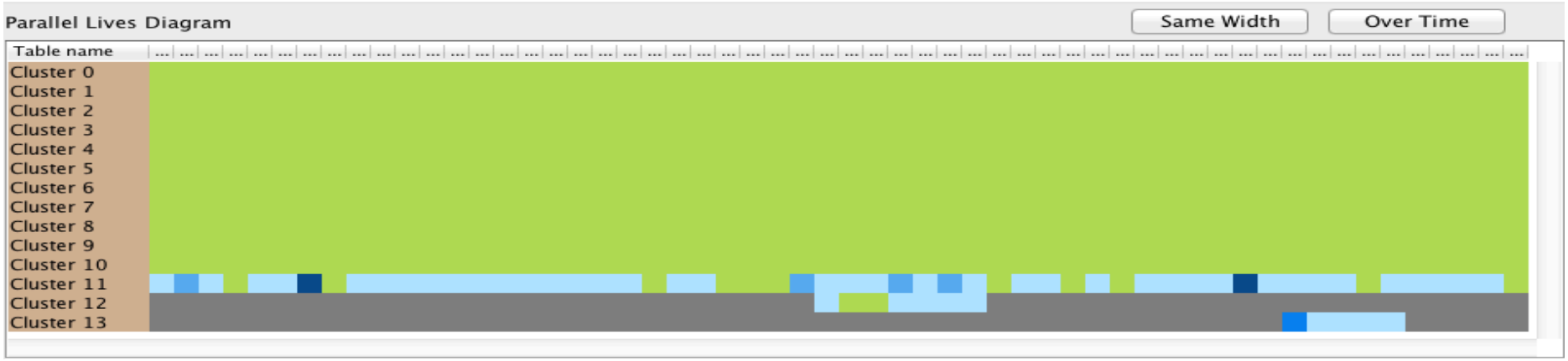


Figure 5.24: Synopsis of phpBB

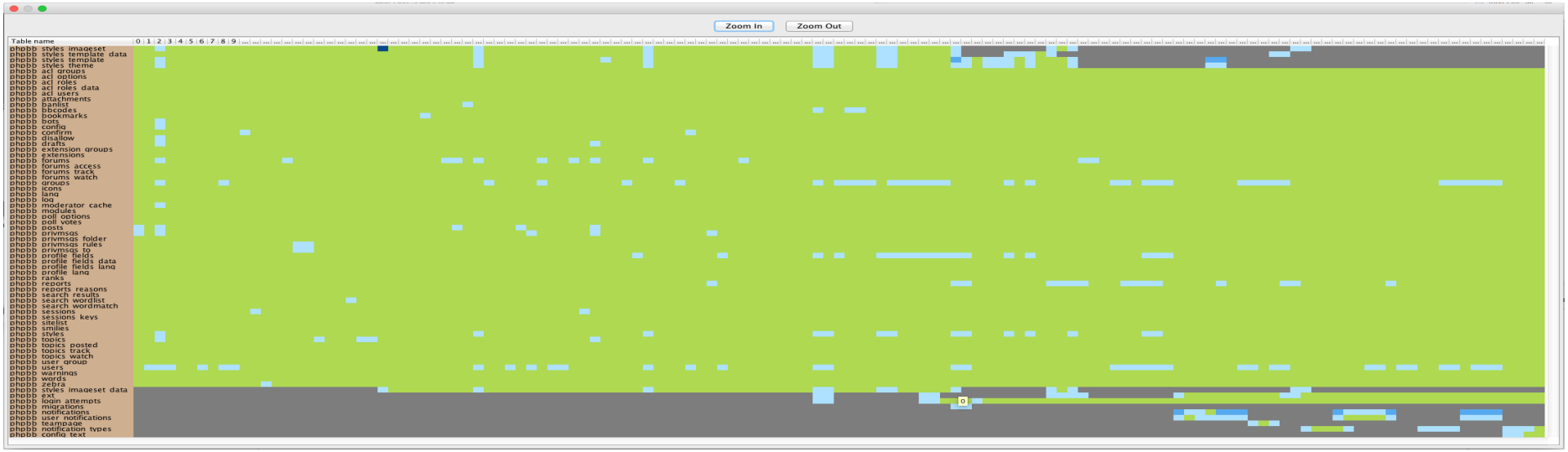


Figure 5.25: PLD of phpBB

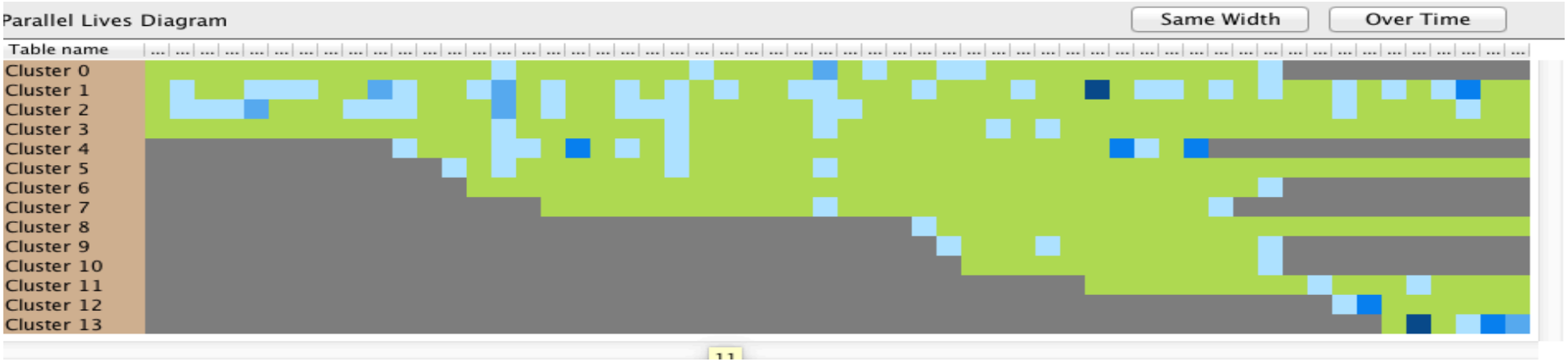


Figure 5.26: Synopsis of Typo3

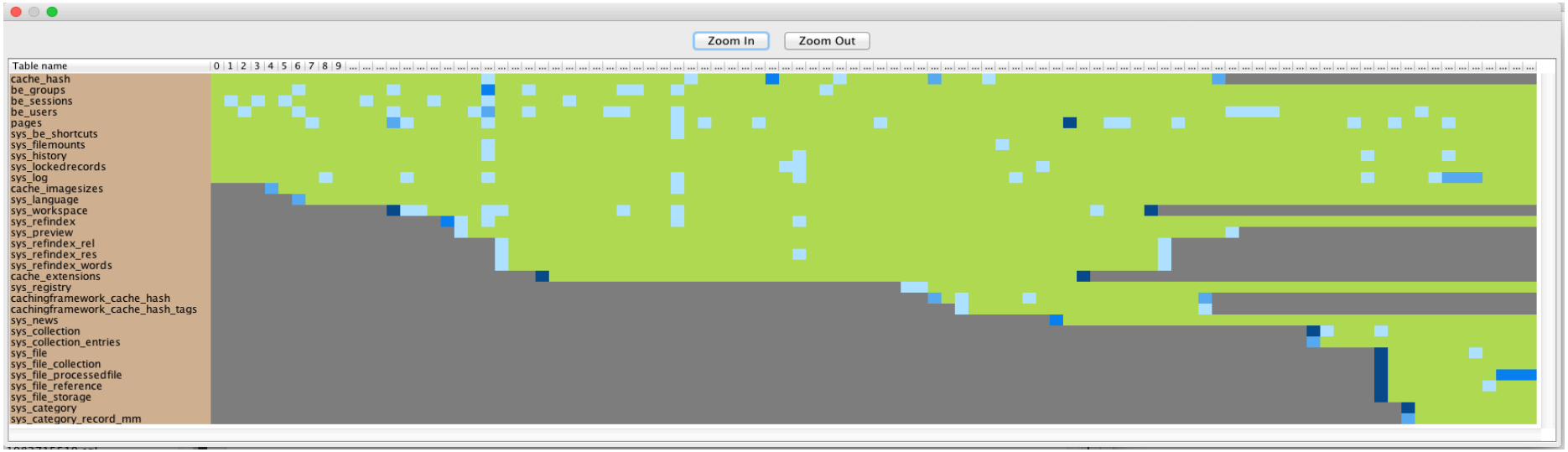


Figure 5.27: PLD of Typo3

CHAPTER 6. RELATED WORK

6.1 Empirical Studies of software and schema evolution
6.2 Timeseries Segmentation
6.3 Data Visualization

In Chapter 6 we will analyze the related work that we studied to get insight for our project. Furthermore in subsection 6.1 we discuss the articles concerning the evolution of schemas and software. In subsection 6.2 there is the analysis of the articles that are connected with timeseries segmentation. At the end of this chapter we analyze the related work that has to do with data visualization.

6.1 Empirical Studies of software and schema evolution

In [SkVZ14], the authors would like to examine if open-source databases comply with Lehman's Laws with regard to Software Evolution. First of all, the authors remind us some basic terminology about the comprehension of the laws such as what does "positive-feedback" mean (the emphasis at adoption to a changing environment and growing for more functionality) or the meaning of "negative-feedback" (changes that are committed should prevent the deterioration of the maintainability and manageability of the software), or each of the laws.

Consequently they provide us with information about the data that they collected and with which they experimented on their goal. There were collected eight different datasets (DDL files of the databases), whose objectives were had to do

with different purposes. Data were processed with Hecate tool [Heca15], a tool that was developed by them too. Hecate can give as a result the differences between two schemas of a database at the attribute level and more specifically changes that have been committed on the attributes of the database's tables such as deletions, additions, data type changes and participation in a changed primary key, or the changes that were committed to the relations level between tables of the database. Additionally it can measure the size of the schemas at the table/attribute level, the total number of changes for each transition between database's versions, etc.

Subsequent to these is the commenting of the experiments that was executed and how much the open-source databases are attached to the Lehman's laws. It seems that the most of the laws like these that refer to the continuous changes, the existence of regulations, the stability, the familiarity and the continuous growing were followed by open-source databases either completely or partially. For these laws that were not referenced, authors commented that there was not a clear opinion because the meanings of the terms of the laws are not obvious enough and this is why it could be a future work.

The subject of [ZhSt05] is the evolution of object-oriented software systems from the point of view of their design. More specifically, the authors propose an algorithm that is called "UMLDiff" and it has as input different versions of UML class diagrams of software systems and produces as output a sequence of differences from version to version. These changes are connected with additions, deletions, movements, renames or even with relations between software entities such as packages, classes, interfaces, methods and fields.

The authors give various definitions for the classification of periods-phases of changes. "Steady state" is the phase that describes the period that the number of additions and deletions of software entities is small enough and the number of movements or modifications is small too. "Restructuring" period has to do with the phase that neither additions nor deletions are many, although the number of

movements and modifications is big. “Functionality extensions (rapidly or slowly developing)” phase refers to the fact that the number of additions and deletions is big or medium for these two statements respectively but the number of movements and modifications is much smaller for both of them. Finally, the “intense developing” phase is defined as the statement that each of the quantities are very big. Interpretively, the references to the quantity of changes such as small, medium or big, are referred to the range of [a<b], where a is the lower bound of additions and removals and b is the higher.

The definitions are followed by the analysis of the evolution of phases with three different techniques that are the “phasic analysis”, the “gamma analysis” and the “optimal matching analysis”, due to various characteristics of the evolution of the software systems. The authors noted that most of the classes when were introduced into the system followed the “slowly-developing” or “steady-state” phases and only a much smaller percentage of them followed the rapidly developing phase. After their introduction a sixty-percent of the classes remained to the “steady-state” phase, a ten-percent went through the “rapid” or “slow” development and a thirty-percent was adapted either from “intense evolution” or “restructuring” phases. Also, the authors underline that most of the system classes went gradually into a steady state but the classes that ended with active rapidly developing, restructuring and slowly developing phases were removed from the system. The final part of the paper is about the evaluation of the “UMLDiff” algorithm that was 95.2% accurate and the commenting on the results.

6.2 Timeseries Segmentation

The authors of [TeTs06] suggest two algorithms for sequence segmentation of a timeseries. Both of them are optimal solutions of a dynamic-programming algorithm that can solve the problem in $O(n^2k)$ time. The first algorithm is called DnS and the second algorithm is called RDnS and both of them have to do with the concise representation of the data of a timeseries assisted by the "piecewise-constant" approximation. This approximation represents a d-dimensional sequence of length n with the help of k non-overlapping continuous segments from which the entire sequence consists of.

Usually, approximation methods are characterized by an error function, whose selection depends on the kind of the problem that would be applied to. So, the authors define the segmentation problem as a problem that gets as input a sequence, an error function and the desired number of segments and gives as output a sequence segmentation and the representatives of each segment, minimizing the error as regards the optimal algorithm.

The DnS (divide and segment) algorithm is the first algorithm that was suggested by the authors and its basic idea is to divide the initial problem to smaller sub-problems. It gets as input a sequence T of length n, a value χ that denotes the number of the sub-problems, and a number k that denotes the number of the representatives for each sub-problem. The intuition behind the algorithm is that we divide the sequence into χ disconnected segments and for each one we find a segmentation Si and a set Mi that is consisting of k representative points with adjacent weights that are dependent on the length of the segment that they represent. Then, the χk representatives are merged and they form a new sequence T'. Finally the sequence T' is given as input to the dynamic programming algorithm that it afterwards gives the output, which is the best possible segmentation of the sequence into k segments. RDnS algorithm is

similar enough with DnS with the only difference that RDnS is run recursively until a depth of recursion has been reached.

Both algorithms were evaluated with the help of both artificial and real data and they were compared with heuristic algorithms such as Top-Down Greedy Algorithm, Bottom-Up Greedy Algorithm, LiR, and GiR. The results were good enough with both kinds of input data and the suggested algorithms performed well in relation to the others. Sometimes they were almost closed to the optimal algorithm.

The basic idea of [TaTT06] is to suggest a segmentation that combines the results of other segmentation algorithms. The authors claim that the suggested method that is called "segmentation aggregation" can be applied to various kinds of data, such as DNA sequences, multi-dimensional categorical data, clustering etc.
The problem definition consists of the input that is a set of segmentations that a timeseries were partitioned and a distance function D between each pair of segmentations. The goal is the finding of a total partition that achieves the minimum sum of the distances from the segmentations that were given as input. The author defines the term "aggregation's cost of the segmentations" that is computed as the sum of each pair of distances of segmentations. They also suggest as the distance function a metric that is called "disagreement distance". Disagreement distance gets two segmentations (P, Q) as input and constructs their union segmentation $U = P \cup Q$ with segments $\{\bar{u}1, \dots, \bar{u}n\}$. Then $P(\bar{u}i) = k$ and $Q(\bar{u}i) = t$ are defined to be the labeling of interval $\bar{u}$i with respect to segmentations P and Q respectively. There is a disagreement when two segments $\bar{u}$i and $\bar{u}$j receive the same label in one segmentation but different in the other. If so, the function D returns a value that is equal with the multiplication of the number of the elements that exist into these specific parts. Otherwise, it returns zero value. The sum of the distances between each pair that is being checked for differences is defined as the total distance for these two segmentations.

After these definitions, authors suggest both optimal and heuristic algorithms for the segmentation aggregation problem. The first algorithm that is named as "Candidate segment boundaries" decreases the range of searching for possible candidate segmentation boundaries from 2^N to 2^n where n is the size of the union segmentation. It also moves the problem to the discrete space because of the reduction of the searching range, and so such kind of algorithms can be applied to resolve this. Next, is another algorithm that belongs to dynamic programming family and uses "breakpoints" to decide the boundaries. The algorithm's functionality is based on the search of the best breakpoint according to an "impact" function. It is called like this because it denotes the impact of each breakpoint to the total cost. Finally, the segmentation that will be given as a result has the minimum cost in comparison with the optimal segmentation. The last algorithm is a greedy Bottom Up algorithm that tries to remove as many as possible boundaries and merge their segments with main goal the achievement of the minimum cost.

The authors evaluated the suggested algorithms with different sets of data that belong to various categories, but the dataset that attracted us more was the dataset that is referred to reality mining data, in which they tried to extract phases from the lives of users using the disagreement distance function for different days that is much similar with the goal of our project for phase extraction of a database history.

6.3 Data Visualization

Schneiderman in [Shne96] has as his main objective to offer to the readers some different ways of data visualization and to motivate them to think about others. The data types that the author deals with include one-dimensional data, two-dimensional data, three-dimensional data, and temporal data such as time lines, multi-dimensional data, trees and networks. The author mentions examples

about the appliance of the paper's motto, which is "Overview first, zoom and filter, then details on-demand" for each of the above data types.

The "Overview" term refers to a general concise representation of the entire data set and also the existence of the ability of the representation for more specific information to an adjacent point. "Zoom and filter" have to do with the need of zooming capability into items that user is interested and which have been primarily filtered and cleaned from the "noise". The last part of the motto "details on-demand", refers to the ability of more comprehensive information about a group of elements or a specific item.

Apart from these capabilities, the author introduces more features that would be desirable for a visualization tool. Furthermore, the relation is suggested as a good feature, because the user could see the relationship between different items. The retainment of the history of user's actions could be another wonderful item because user would have the capability to undo something or even better find some search terms that he have searched. All of these make a tool much pleasant to use. Moreover, the extraction and the sharing of individual data or even the printing or mailing them could make a tool more powerful. Finally this type of tools could have a more complex searching that different parameters would be combined to give the desired result to the user.

CHAPTER 7. CONCLUSIONS AND OPEN ISSUES

In conclusion, the main problem of this thesis was to find a way to fit the entire life of a database that consists of hundreds of transitions and hundreds of tables into a window of our screen.

For this purpose we contracted the x-axis of the initial view which includes the entire set of transitions with the assistance of a phase extraction algorithm that we designed and which creates a desired number of phases that contain the most similar transitions according to a distance function. Our phase extraction algorithm was assessed with two different methods with fairly good results.

In parallel, we designed a cluster extraction algorithm to contract the y-axis of the initial view that contains the whole set of tables of a database into a set of clusters each of which contains the most similar tables according to another distance function. We used both internal validity and external validity techniques to evaluate our cluster extraction algorithm and both of them gave us satisfying results.

By combining these two algorithms we have achieved to get a good overview of the initial map of the life of a database which could not be handled otherwise. Moreover, this overview itself could further enriched so we implemented features such as zooming into specific points of the overview, filtering by different criteria (like one cluster, or one phase, or a unique table) and get details on demand about various elements such phases, clusters, tables, transitions, etc. Finally, all of these features were implemented by our tool Plutarch's Parallel

Lives [PPL15]. PPL is publicly available via Github as free and open-source software.

With regard to the open issues of this thesis and more precisely to the theoretical part of it, the implementation of different distance metrics for both the phase extraction algorithm and the cluster extraction algorithm could be a different approach on this problem.

From the view point of the software, Plutarch's Parallel Lives can be enriched with more features such as the exporting of some kind of reports for the phase and cluster extraction. The user can be provided with the ability to pre-select some transitions or tables to be at the same phase or cluster respectively. Also, the selection of more than one phases to drill into is another feature that can be added. Finally, the graphics of the user interface can be responsive according to different screen dimensions.

REFERENCES

[Ecli15] Eclipse IDE. Available at https://eclipse.org/downloads/. Last accessed 2015-09-30.

[Heca15] Hecate. Available at https://github.com/daintiness-group/hecate . Last accessed 2015-09-30.

[PPL15] Plutarch's Parallel Lives at https://github.com/daintiness-group/plutarch_parallel_lives. Last accessed 2015-09-30

[Shne96] Shneiderman, Ben. "The eyes have it: A task by data type taxonomy for information visualizations." *Visual Languages, 1996. Proceedings., IEEE Symposium on*. IEEE, 1996.

[SkVZ14] Skoulis, Ioannis, Panos Vassiliadis, and Apostolos Zarras. "Open-Source Databases: Within, Outside, or Beyond Lehman's Laws of Software Evolution?."*Advanced Information Systems Engineering*. Springer International Publishing, 2014.

[TaSK05] Pang-Ning Tan, Michael Steinbach and Vipin Kumar. *Introduction to Data Mining.* 1st ed. Pearson, 2005.

[TaTT06] Mielikäinen, Taneli, Evimaria Terzi, and Panayiotis Tsaparas. "Aggregating time partitions." *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006.

[TeTs06] Terzi, Evimaria, and Panayiotis Tsaparas. "Efficient Algorithms for Sequence Segmentation." *SDM*. 2006.

[ZhSt05] Xing, Zhenchang, and Eleni Stroulia. "Analyzing the evolutionary history of the logical design of object-oriented software." *Software Engineering, IEEE Transactions on* 31.10 (2005): 850-868.

APPENDIX

Metrics of change for the database level

The history of the database can provide us with zoomed-out metrics for the quantified version of the database's heartbeat. Specifically, we can employ the following measures for the change that a database schema undergoes in the context of a specific transition *t*.

Relation change: |relations inserted| + |relations updated| + |relations updated|

Relation change measures each newly inserted/deleted/updated relation just once within each transition, independently of the number attributes created/deleted/updated within its schema.

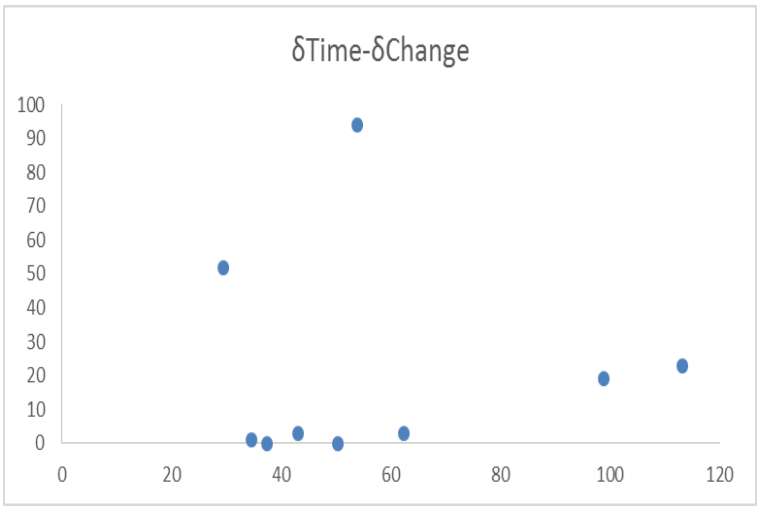
Attribute change:
|attributes born with new relations| + |attributes removed with removed relations| + |attributes injected in existing relations| + |attributed removed from existing relations| + |attributes with type alterations| + |attributes involved in key alterations|
i.e., the sum of the corresponding individual-relation metrics

Assessment of phase extraction

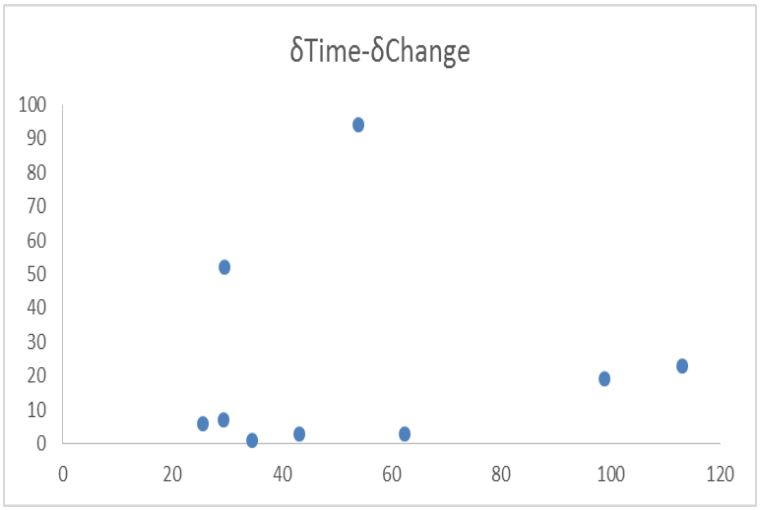
Atlas

Table A- 1: Assessment of phase extraction for Atlas

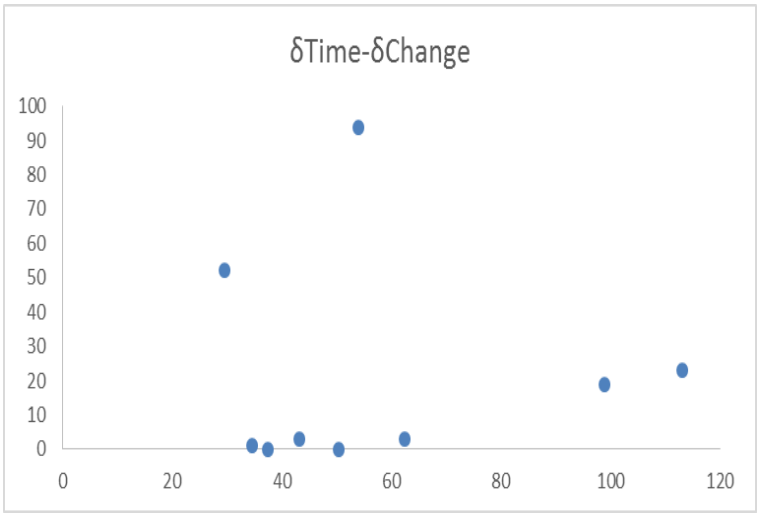
WC: 0.0	WT: 1.0	
PPC:OFF	PPT:OFF	
Phases	δtime	δc
0@1	98.90	19
1@2	29.54	52
2@3	37.47	0
3@4	50.46	0
4@5	53.97	94
5@6	43.12	3
6@7	113.20	23
7@8	34.57	1
8@9	62.45	3



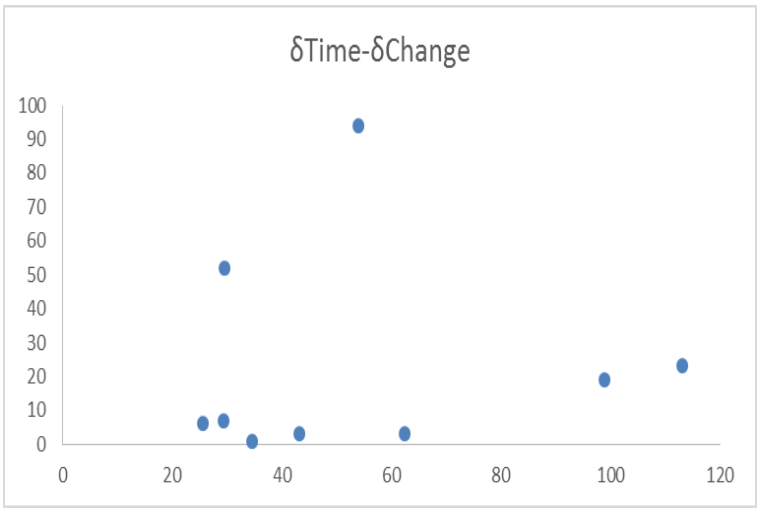
WC: 0.0	WT: 1.0	
PPC:ON	PPT:OFF	
Phases	δtime	δc
0@1	25.59	6
1@2	29.44	7
2@3	98.90	19
3@4	29.54	52
4@5	53.97	94
5@6	43.12	3
6@7	113.20	23
7@8	34.57	1
8@9	62.45	3



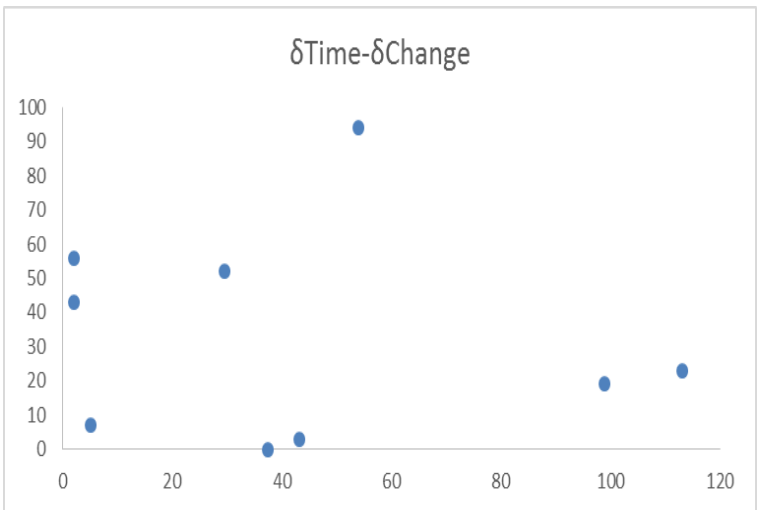
WC: 0.0	WT: 1.0	
PPC:OFF	PPT:ON	
Phases	δtime	δc
0@1	98.90	19
1@2	29.54	52
2@3	37.47	0
3@4	50.46	0
4@5	53.97	94
5@6	43.12	3
6@7	113.20	23
7@8	34.57	1
8@9	62.45	3



WC: 0.0	WT: 1.0	
PPC:ON	PPT:ON	
Phases	δtime	δc
0@1	25.59	6
1@2	29.44	7
2@3	98.90	19
3@4	29.54	52
4@5	53.97	94
5@6	43.12	3
6@7	113.20	23
7@8	34.57	1
8@9	62.45	3

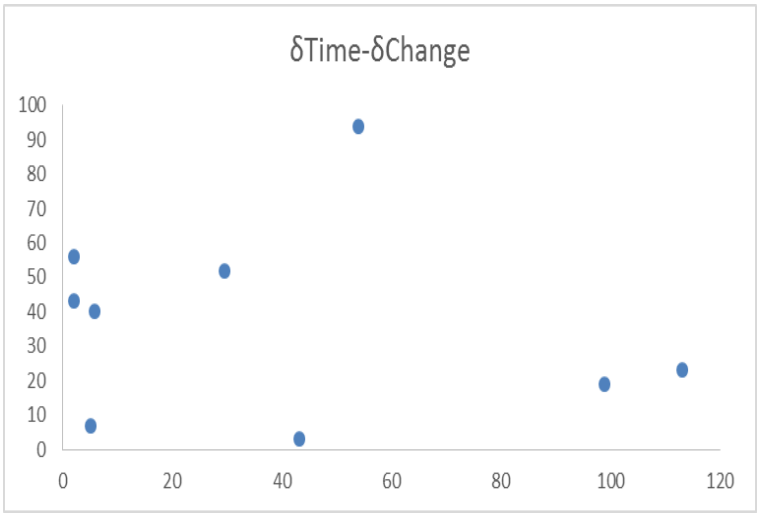


WC: 0.5	WT: 0.5	
PPC:OFF	PPT:OFF	
Phases	δtime	δc
0@1	98.90	19
1@2	2.04	56
2@3	2.04	43
3@4	5.15	7
4@5	29.54	52
5@6	37.47	0
6@7	53.97	94
7@8	43.12	3
8@9	113.20	23



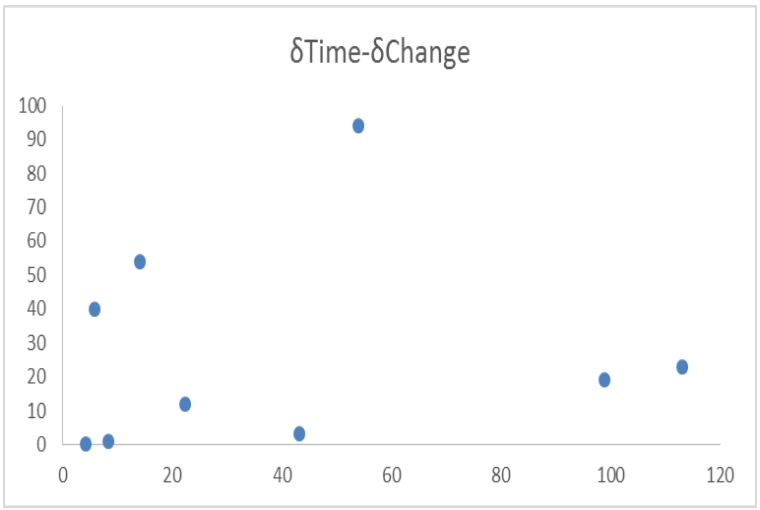
WC: 0.5 WT: 0.5
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	98.90	19
1@2	2.04	56
2@3	2.04	43
3@4	5.15	7
4@5	29.54	52
5@6	53.97	94
6@7	5.79	40
7@8	43.12	3
8@9	113.20	23



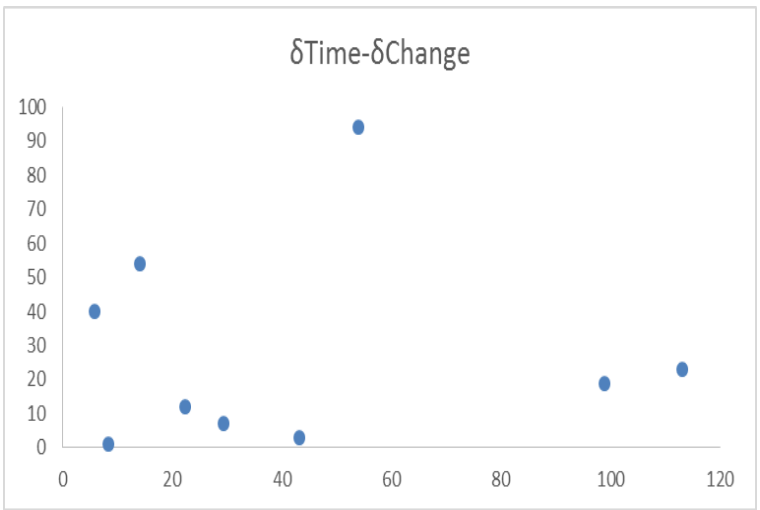
WC: 0.5 WT: 0.5
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	98.90	19
1@2	14.14	54
2@3	8.39	1
3@4	53.97	94
4@5	22.42	12
5@6	5.79	40
6@7	4.24	0
7@8	43.12	3
8@9	113.20	23



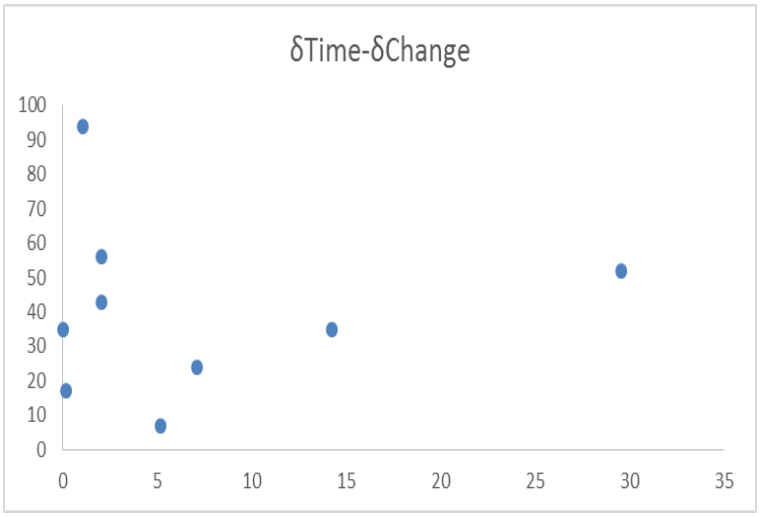
WC: 0.5 WT: 0.5
PPC:ON PPT:ON

Phases	δtime	δc
0@1	29.44	7
1@2	98.90	19
2@3	14.14	54
3@4	8.39	1
4@5	53.97	94
5@6	22.42	12
6@7	5.79	40
7@8	43.12	3
8@9	113.20	23



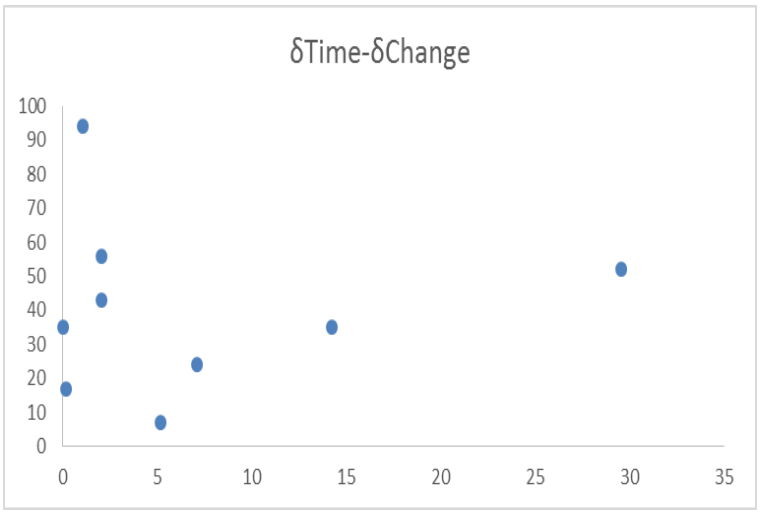
WC: 1.0 WT: 0.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	2.04	56
1@2	2.04	43
2@3	5.15	7
3@4	29.54	52
4@5	1.07	94
5@6	0.01	35
6@7	0.17	17
7@8	7.09	24
8@9	14.21	35



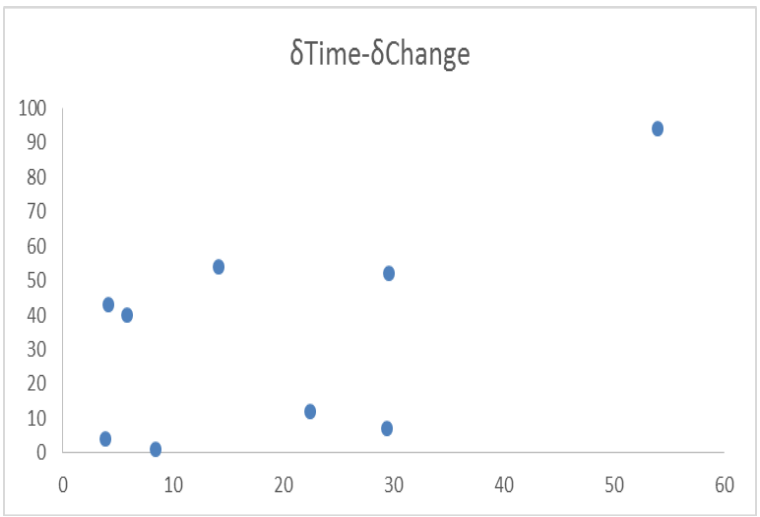
WC: 1.0 WT: 0.0
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	2.04	56
1@2	2.04	43
2@3	5.15	7
3@4	29.54	52
4@5	1.07	94
5@6	0.01	35
6@7	0.17	17
7@8	7.09	24
8@9	14.21	35



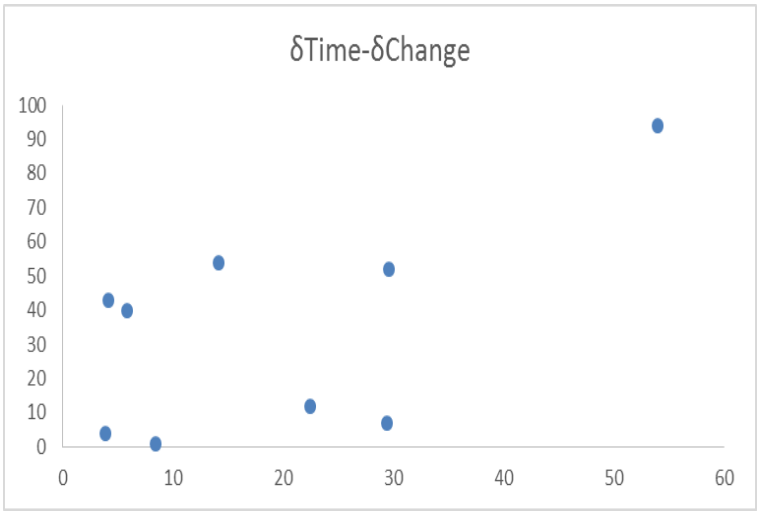
WC: 1.0 WT: 0.0
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	29.44	7
1@2	14.14	54
2@3	3.87	4
3@4	29.54	52
4@5	8.39	1
5@6	53.97	94
6@7	22.42	12
7@8	5.79	40
8@9	4.15	43



WC: 1.0 WT: 0.0
PPC:ON PPT:ON

Phases	δtime	δc
0@1	29.44	7
1@2	14.14	54
2@3	3.87	4
3@4	29.54	52
4@5	8.39	1
5@6	53.97	94
6@7	22.42	12
7@8	5.79	40
8@9	4.15	43

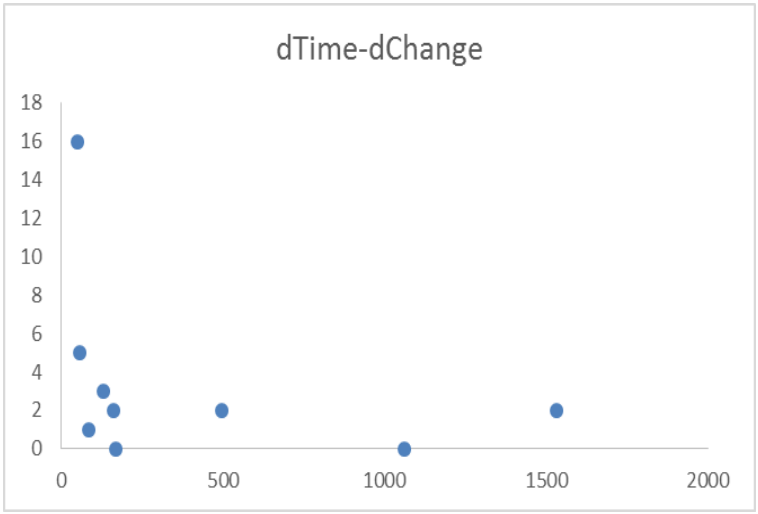


bioSQL Dataset

Table A- 2: Assessment of phase extraction for bioSQL

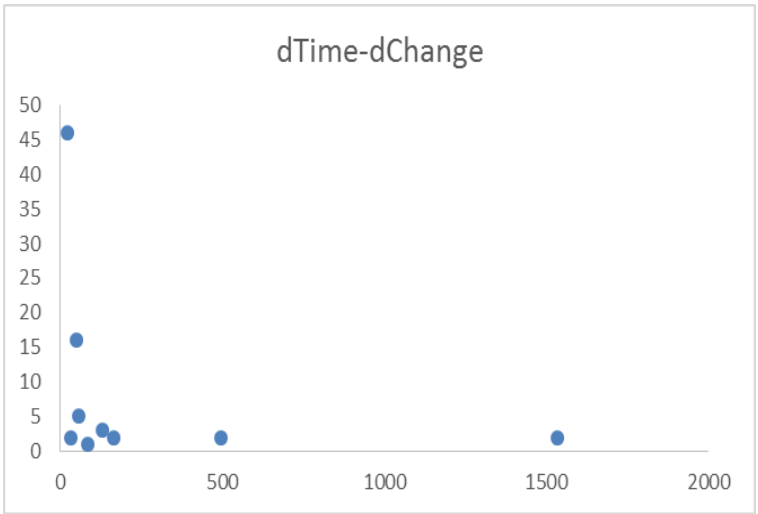
WC: 0.0 WT: 1.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	57.75	5
1@2	130.54	3
2@3	85.76	1
3@4	50.35	16
4@5	497.28	2
5@6	168.66	0
6@7	1061.08	0
7@8	163.49	2
8@9	1533.19	2



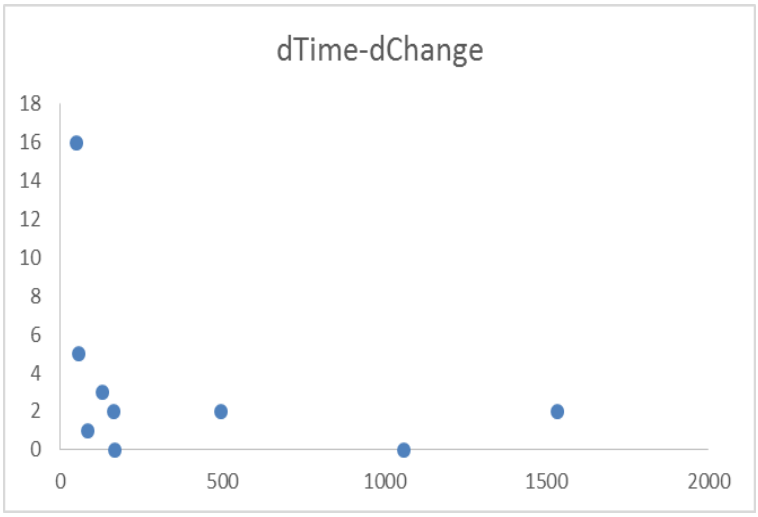
WC: 0.0 WT: 1.0
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	57.75	5
1@2	130.54	3
2@3	85.76	1
3@4	20.88	46
4@5	50.35	16
5@6	497.28	2
6@7	31.68	2
7@8	163.49	2
8@9	1533.19	2



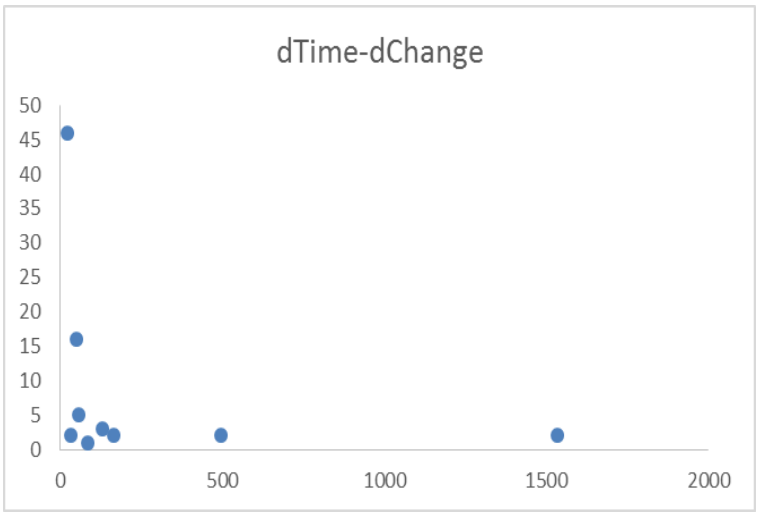
WC: 0.0 WT: 1.0
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	57.75	5
1@2	130.54	3
2@3	85.76	1
3@4	50.35	16
4@5	497.28	2
5@6	168.66	0
6@7	1061.08	0
7@8	163.49	2
8@9	1533.19	2



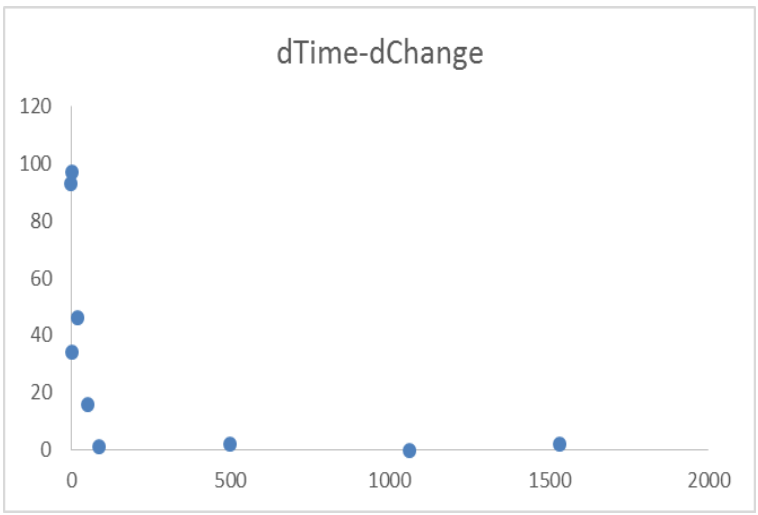
WC: 0.0 WT: 1.0
PPC:ON PPT:ON

Phases	δtime	δc
0@1	57.75	5
1@2	130.54	3
2@3	85.76	1
3@4	20.88	46
4@5	50.35	16
5@6	497.28	2
6@7	31.68	2
7@8	163.49	2
8@9	1533.19	2

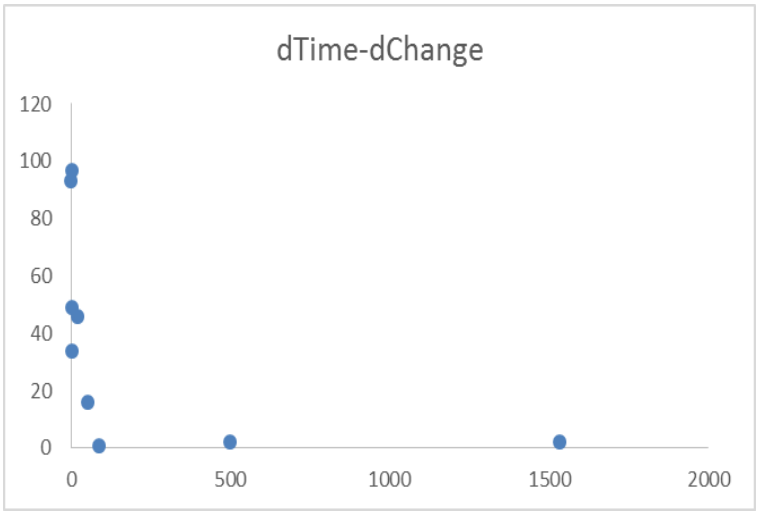


WC: 0.5 WT: 0.5
PPC:OFF PPT:OFF

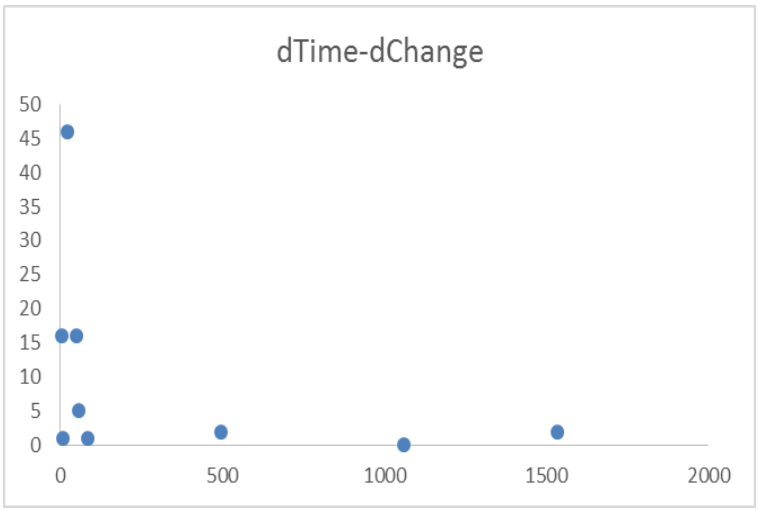
Phases	δtime	δc
0@1	2.14	34
1@2	85.76	1
2@3	1.51	97
3@4	0.03	93
4@5	20.88	46
5@6	50.35	16
6@7	497.28	2
7@8	1061.08	0
8@9	1533.19	2



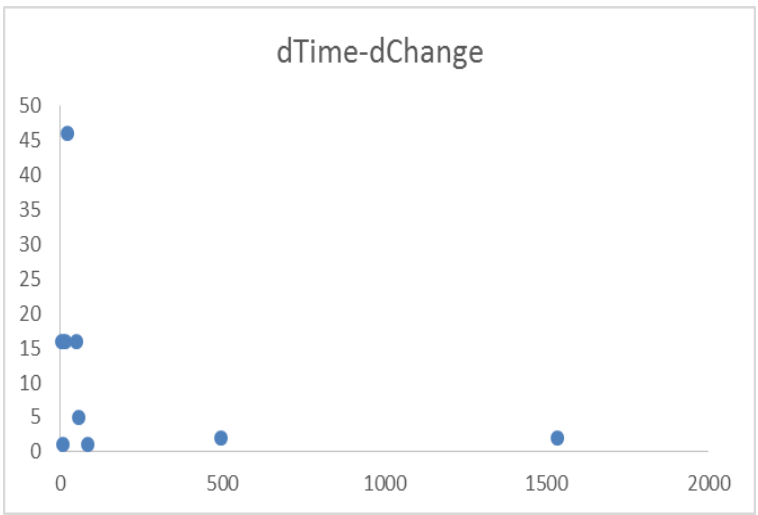
WC: 0.5	WT: 0.5	
PPC:ON	PPT:OFF	
Phases	δtime	δc
0@1	2.14	34
1@2	1.28	49
2@3	85.76	1
3@4	1.51	97
4@5	0.03	93
5@6	20.88	46
6@7	50.35	16
7@8	497.28	2
8@9	1533.19	2



WC: 0.5	WT: 0.5	
PPC:OFF	PPT:ON	
Phases	δtime	δc
0@1	57.75	5
1@2	6.44	1
2@3	85.76	1
3@4	20.88	46
4@5	4.97	16
5@6	50.35	16
6@7	497.28	2
7@8	1061.08	0
8@9	1533.19	2

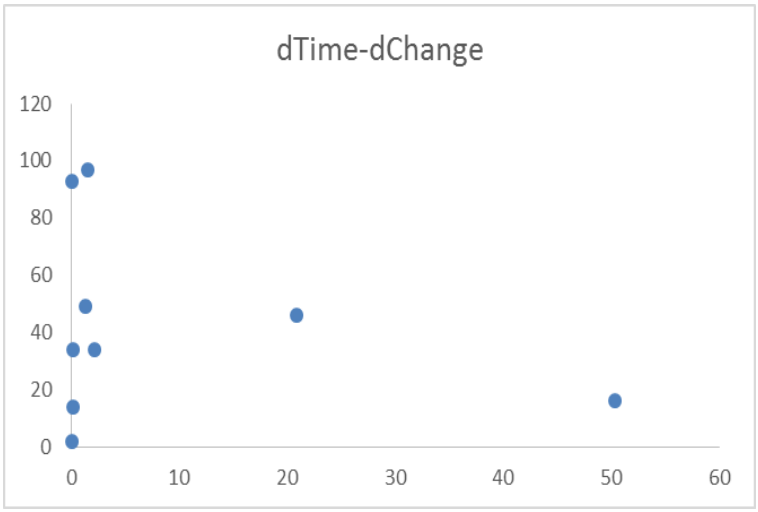


WC: 0.5	WT: 0.5	
PPC:ON	PPT:ON	
Phases	δtime	δc
0@1	57.75	5
1@2	6.44	1
2@3	85.76	1
3@4	20.88	46
4@5	4.97	16
5@6	14.77	16
6@7	50.35	16
7@8	497.28	2
8@9	1533.19	2



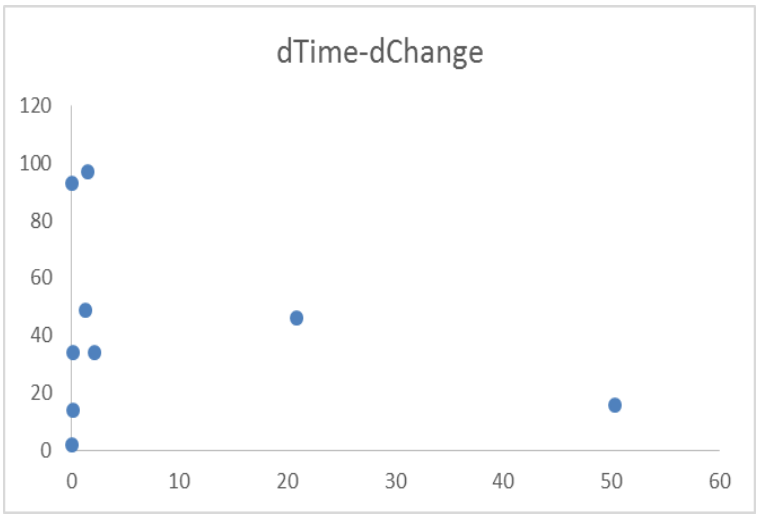
WC: 1.0 WT: 0.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	2.14	34
1@2	0.15	34
2@3	1.28	49
3@4	0.08	2
4@5	1.51	97
5@6	0.03	93
6@7	0.16	14
7@8	20.88	46
8@9	50.35	16



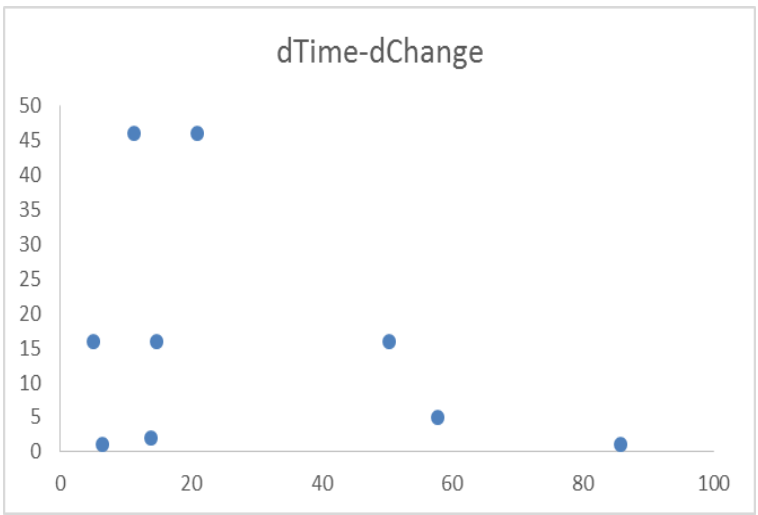
WC: 1.0 WT: 0.0
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	2.14	34
1@2	0.15	34
2@3	1.28	49
3@4	0.08	2
4@5	1.51	97
5@6	0.03	93
6@7	0.16	14
7@8	20.88	46
8@9	50.35	16

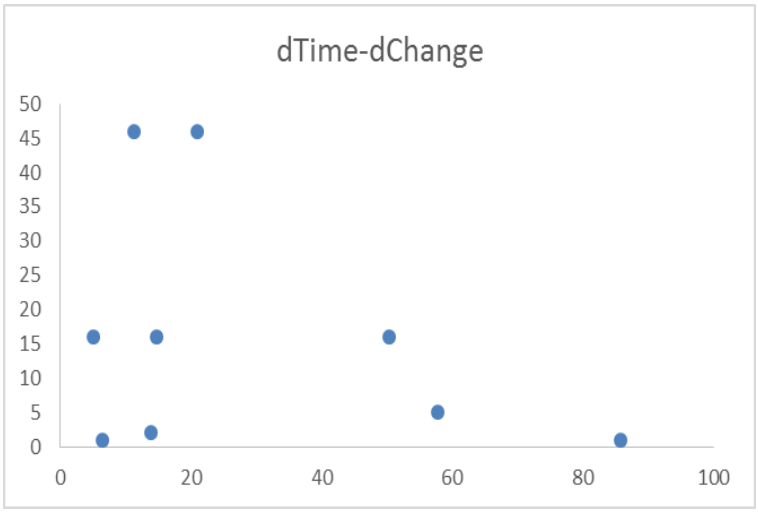


WC: 1.0 WT: 0.0
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	57.75	5
1@2	6.44	1
2@3	11.21	46
3@4	13.84	2
4@5	85.76	1
5@6	20.88	46
6@7	4.97	16
7@8	14.77	16
8@9	50.35	16



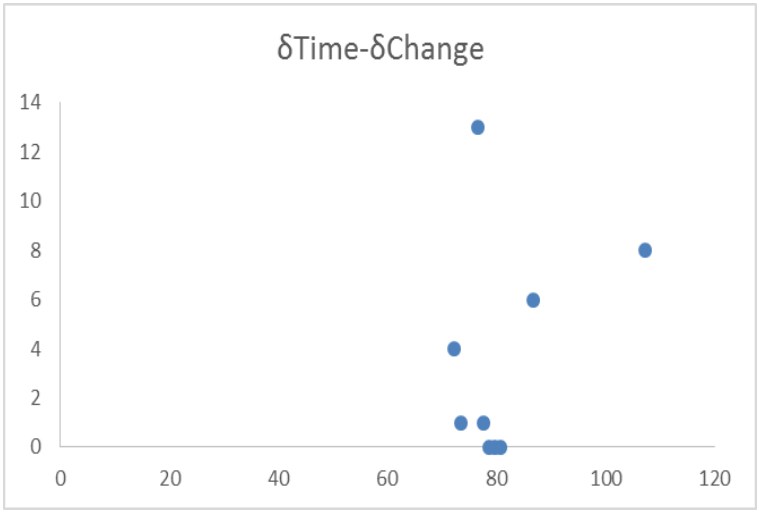
WC: 1.0	WT: 0.0	
PPC:ON	PPT:ON	
Phases	δtime	δc
0@1	57.75	5
1@2	6.44	1
2@3	11.21	46
3@4	13.84	2
4@5	85.76	1
5@6	20.88	46
6@7	4.97	16
7@8	14.77	16
8@9	50.35	16



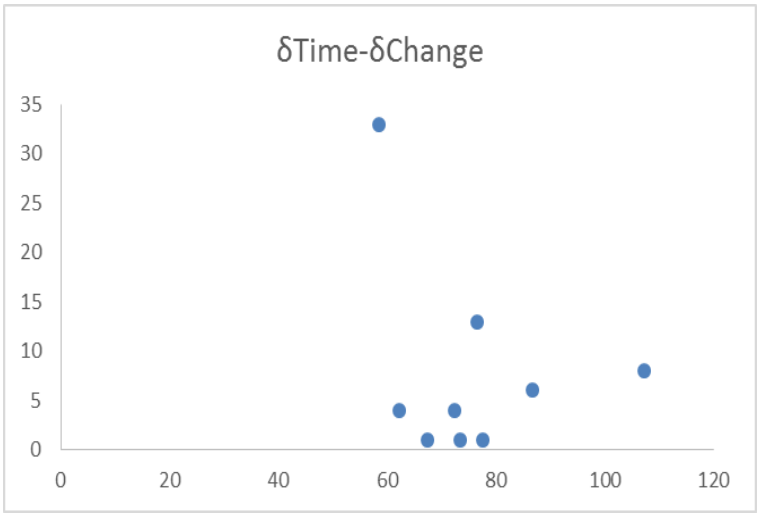
Ensembl Dataset

Table A- 3: Assessment of phase extraction for Ensembl

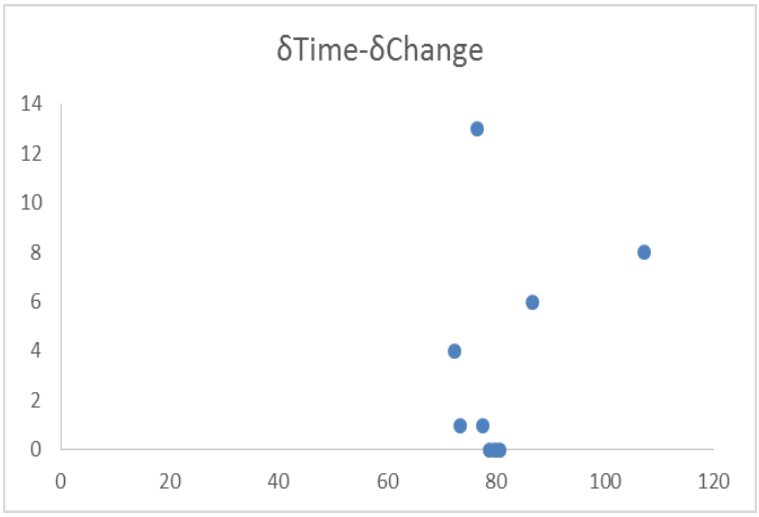
WC: 0.0	WT: 1.0	
PPC:OFF	PPT:OFF	
Phases	δTime	δc
0@1	107.20	8
1@2	86.75	6
2@3	76.51	13
3@4	80.71	0
4@5	79.80	0
5@6	72.28	4
6@7	77.62	1
7@8	78.75	0
8@9	73.39	1



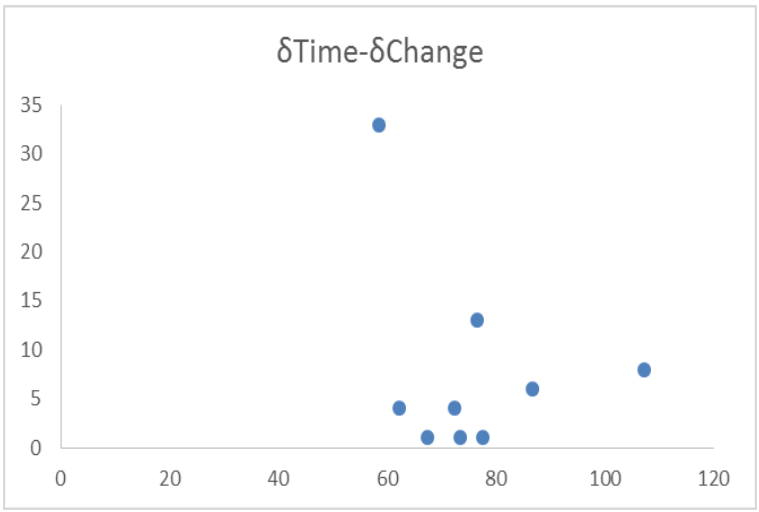
WC: 0.0	WT: 1.0	
PPC:ON	PPT:OFF	
Phases	δTime	δc
0@1	58.37	33
1@2	62.16	4
2@3	107.20	8
3@4	86.75	6
4@5	76.51	13
5@6	67.45	1
6@7	72.28	4
7@8	77.62	1
8@9	73.39	1



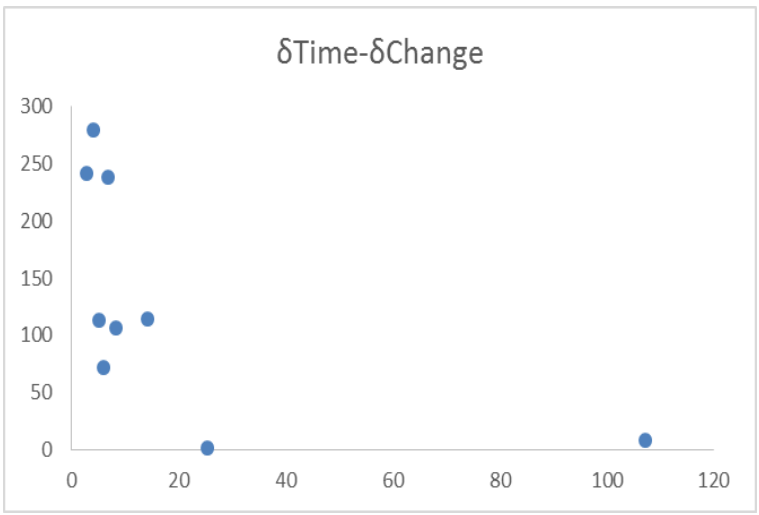
WC: 0.0	WT: 1.0	
PPC:OFF	PPT:ON	
Phases	δTime	δc
0@1	107.20	8
1@2	86.75	6
2@3	76.51	13
3@4	80.71	0
4@5	79.80	0
5@6	72.28	4
6@7	77.62	1
7@8	78.75	0
8@9	73.39	1



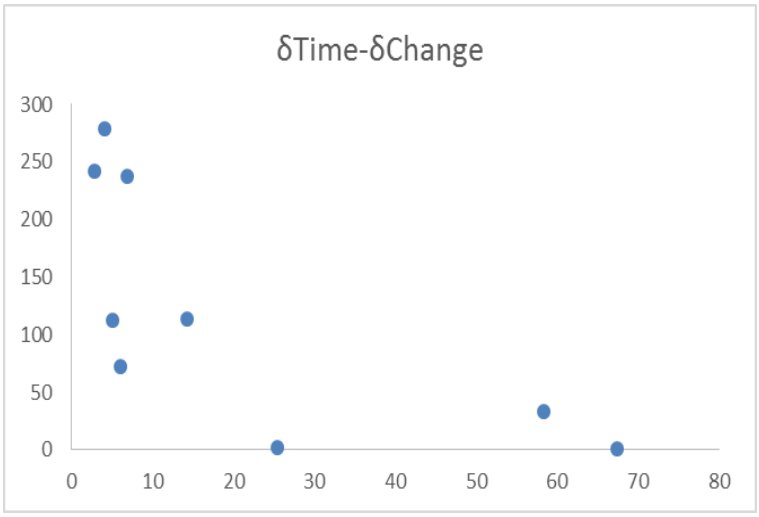
WC: 0.0	WT: 1.0	
PPC:ON	PPT:ON	
Phases	δTime	δc
0@1	58.37	33
1@2	62.16	4
2@3	107.20	8
3@4	86.75	6
4@5	76.51	13
5@6	67.45	1
6@7	72.28	4
7@8	77.62	1
8@9	73.39	1



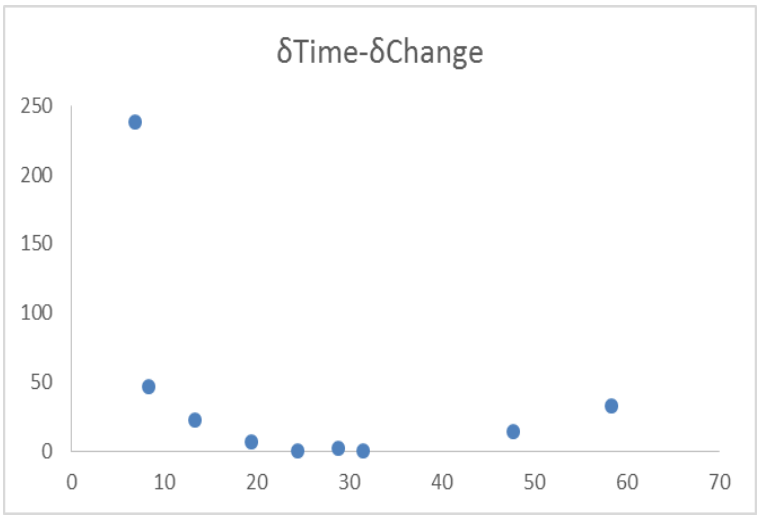
WC: 0.5	WT: 0.5	
PPC:OFF	PPT:OFF	
Phases	δTime	δc
0@1	25.39	2
1@2	6.02	72
2@3	4.11	279
3@4	8.23	107
4@5	6.91	238
5@6	2.81	242
6@7	5.14	113
7@8	14.29	114
8@9	107.20	8



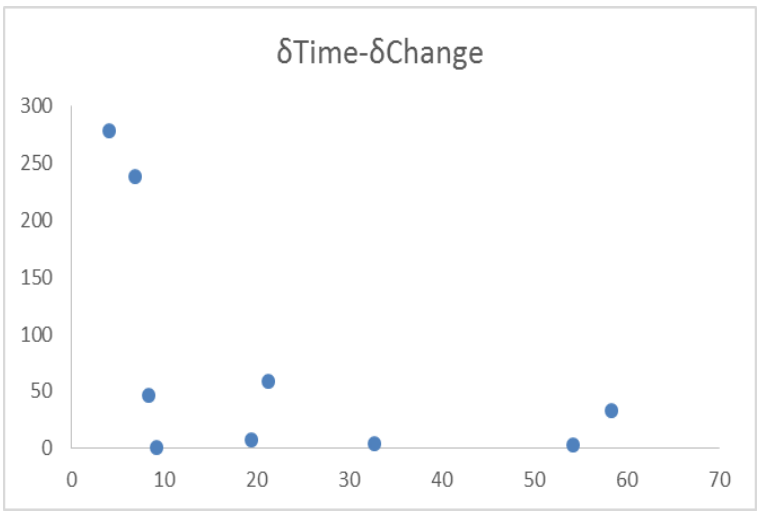
WC: 0.5	WT: 0.5	
PPC:ON	PPT:OFF	
Phases	δTime	δc
0@1	25.39	2
1@2	6.02	72
2@3	4.11	279
3@4	58.37	33
4@5	6.91	238
5@6	2.81	242
6@7	5.14	113
7@8	14.29	114
8@9	67.45	1



WC: 0.5	WT: 0.5	
PPC:OFF	PPT:ON	
Phases	δTime	δc
0@1	8.36	47
1@2	19.48	7
2@3	31.55	0
3@4	13.32	23
4@5	58.37	33
5@6	6.91	238
6@7	24.50	0
7@8	47.80	14
8@9	28.80	2

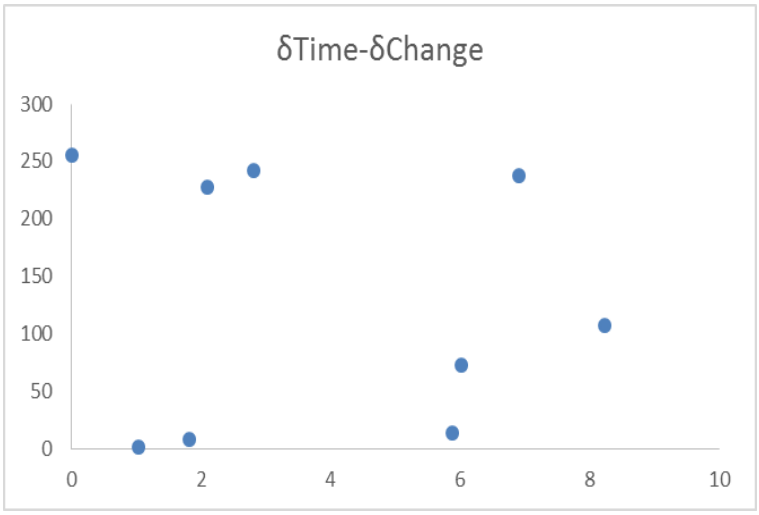


WC: 0.5	WT: 0.5	
PPC:ON	PPT:ON	
Phases	δTime	δc
0@1	8.36	47
1@2	19.48	7
2@3	21.26	59
3@4	4.11	279
4@5	58.37	33
5@6	6.91	238
6@7	32.70	4
7@8	9.15	1
8@9	54.23	3



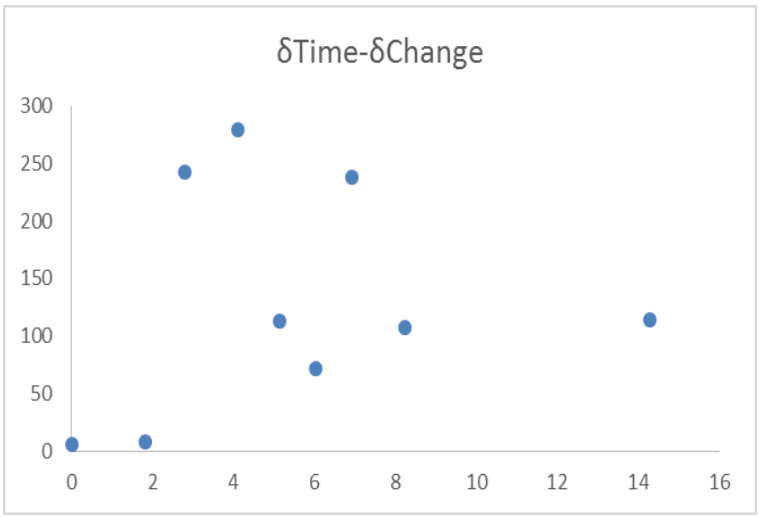
WC: 1.0 WT: 0.0
PPC:OFF PPT:OFF

Phases	δTime	δc
0@1	1.82	8
1@2	6.02	72
2@3	0.00	255
3@4	1.03	1
4@5	8.23	107
5@6	5.88	13
6@7	2.10	228
7@8	6.91	238
8@9	2.81	242



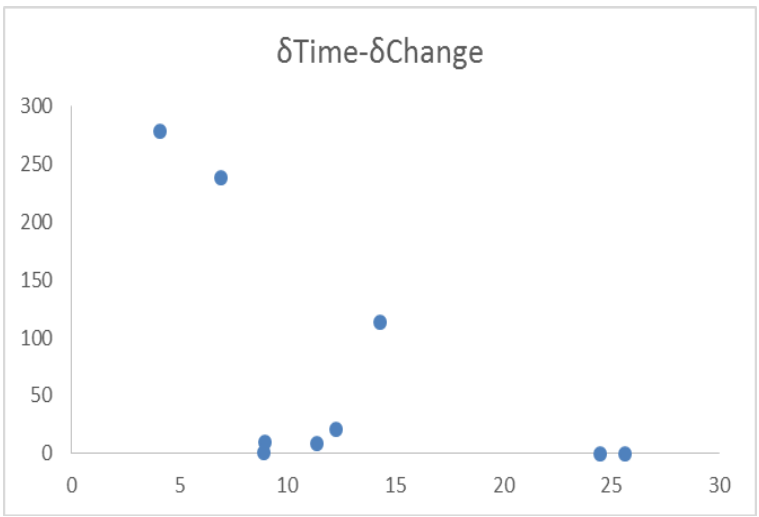
WC: 1.0 WT: 0.0
PPC:ON PPT:OFF

Phases	δTime	δc
0@1	1.82	8
1@2	6.02	72
2@3	4.11	279
3@4	8.23	107
4@5	6.91	238
5@6	2.81	242
6@7	5.14	113
7@8	14.29	114
8@9	0.00	6



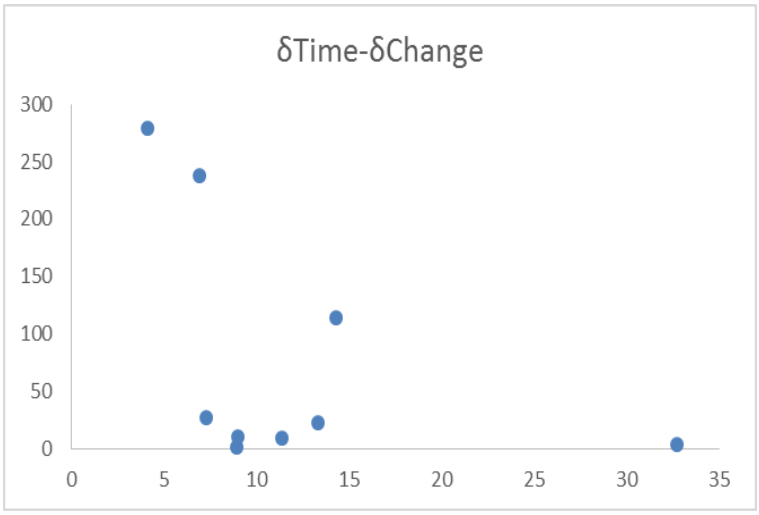
WC: 1.0 WT: 0.0
PPC:OFF PPT:ON

Phases	δTime	δc
0@1	8.98	10
1@2	8.90	1
2@3	4.11	279
3@4	12.28	21
4@5	6.91	238
5@6	24.50	0
6@7	11.37	9
7@8	14.29	114
8@9	25.65	0



WC: 1.0 WT: 0.0
PPC:ON PPT:ON

Phases	δTime	δc
0@1	8.98	10
1@2	8.90	1
2@3	13.32	23
3@4	4.11	279
4@5	6.91	238
5@6	32.70	4
6@7	11.37	9
7@8	14.29	114
8@9	7.25	27

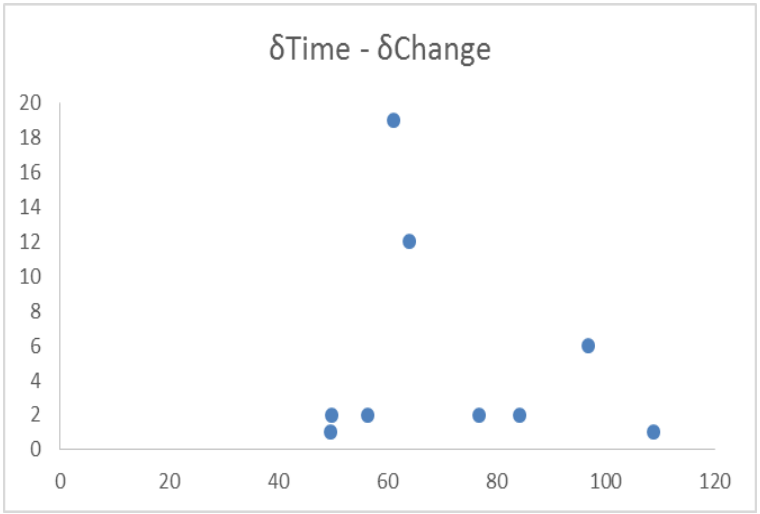


MediaWiki Dataset

Table A- 4: Assessment of phase extraction for mediaWiki

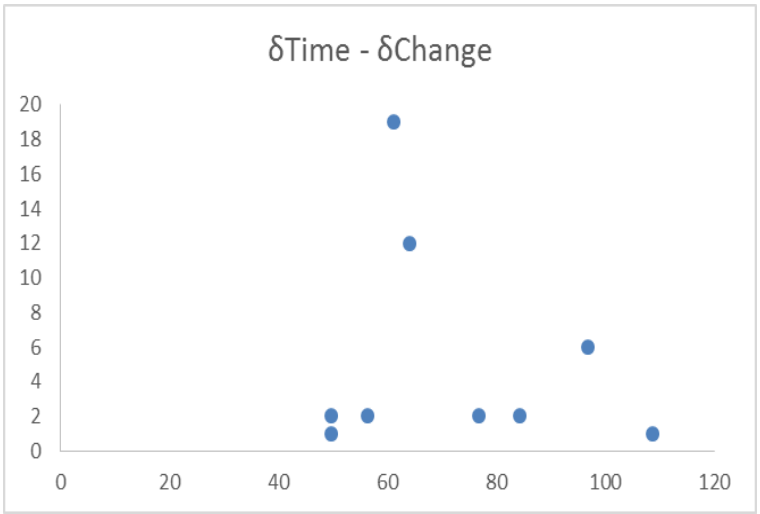
WC: 0.0 WT: 1.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	108.78	1
1@2	61.13	19
2@3	56.32	2
3@4	96.87	6
4@5	49.69	1
5@6	64.05	12
6@7	84.22	2
7@8	49.73	2
8@9	76.84	2



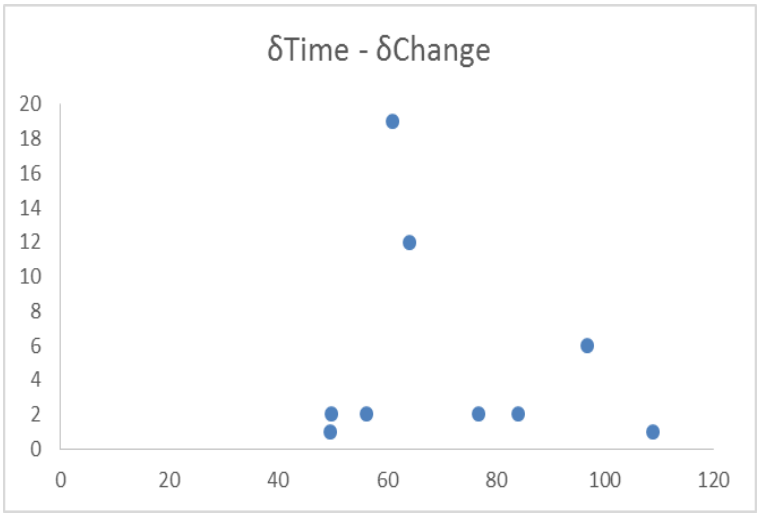
WC: 0.0 WT: 1.0
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	108.78	1
1@2	61.13	19
2@3	56.32	2
3@4	96.87	6
4@5	49.69	1
5@6	64.05	12
6@7	84.22	2
7@8	49.73	2
8@9	76.84	2



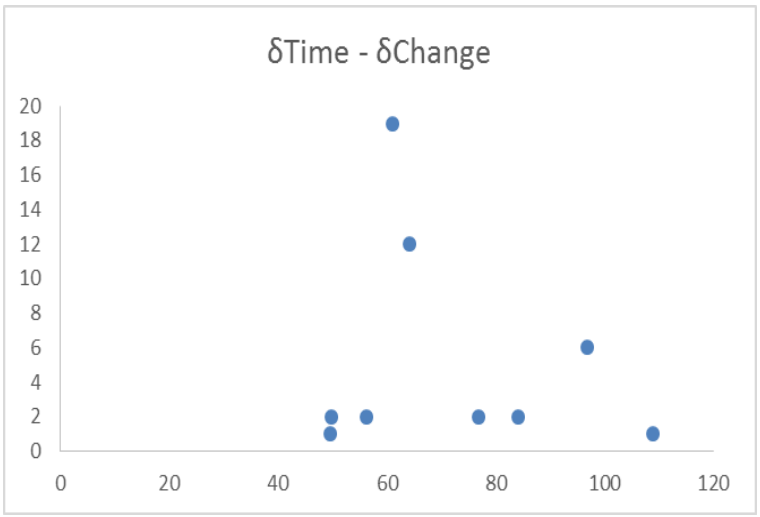
WC: 0.0 WT: 1.0
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	108.78	1
1@2	61.13	19
2@3	56.32	2
3@4	96.87	6
4@5	49.69	1
5@6	64.05	12
6@7	84.22	2
7@8	49.73	2
8@9	76.84	2



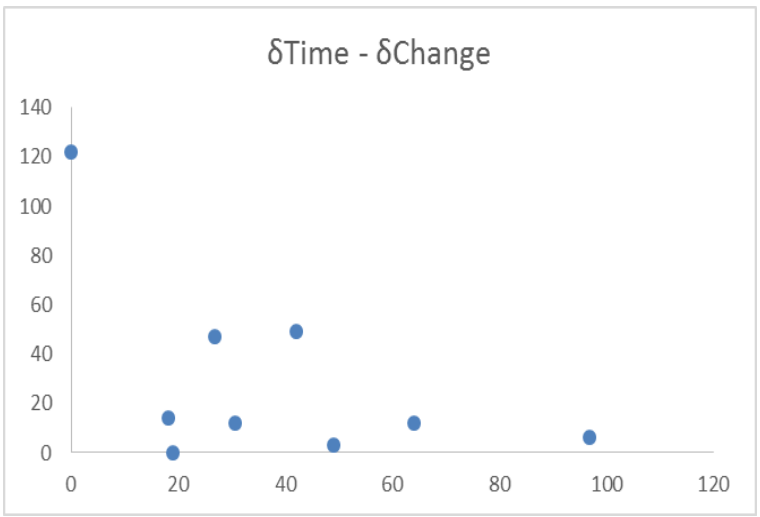
WC: 0.0 WT: 1.0
PPC:ON PPT:ON

Phases	δtime	δc
0@1	108.78	1
1@2	61.13	19
2@3	56.32	2
3@4	96.87	6
4@5	49.69	1
5@6	64.05	12
6@7	84.22	2
7@8	49.73	2
8@9	76.84	2

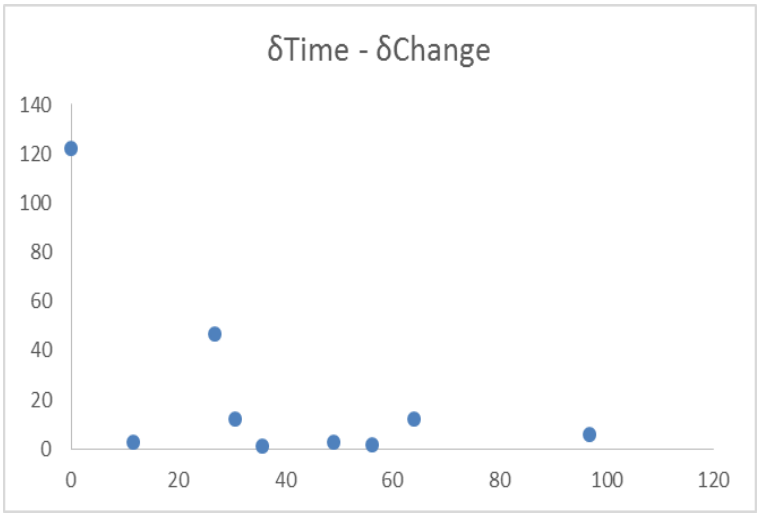


WC: 0.5 WT: 0.5
PPC:OFF PPT:OFF

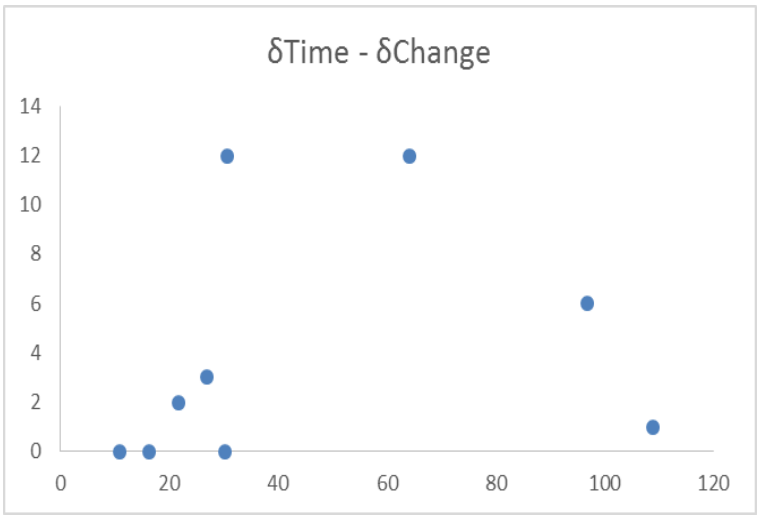
Phases	δtime	δc
0@1	49.12	3
1@2	26.80	47
2@3	19.09	0
3@4	0.07	122
4@5	96.87	6
5@6	64.05	12
6@7	30.65	12
7@8	42.15	49
8@9	18.09	14



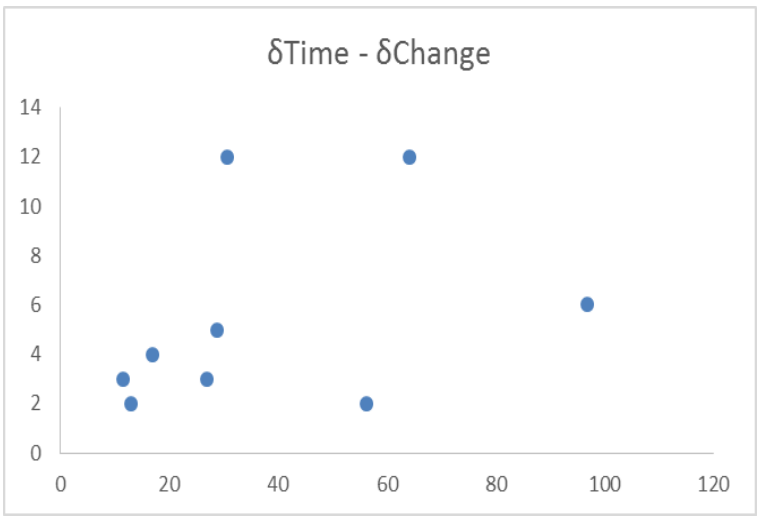
WC: 0.5	WT: 0.5	
PPC:ON	PPT:OFF	
Phases	δtime	δc
0@1	49.12	3
1@2	26.80	47
2@3	35.62	1
3@4	11.59	3
4@5	56.32	2
5@6	0.07	122
6@7	96.87	6
7@8	64.05	12
8@9	30.65	12



WC: 0.5	WT: 0.5	
PPC:OFF	PPT:ON	
Phases	δtime	δc
0@1	108.78	1
1@2	21.75	2
2@3	30.21	0
3@4	16.35	0
4@5	96.87	6
5@6	64.05	12
6@7	30.65	12
7@8	10.89	0
8@9	26.85	3

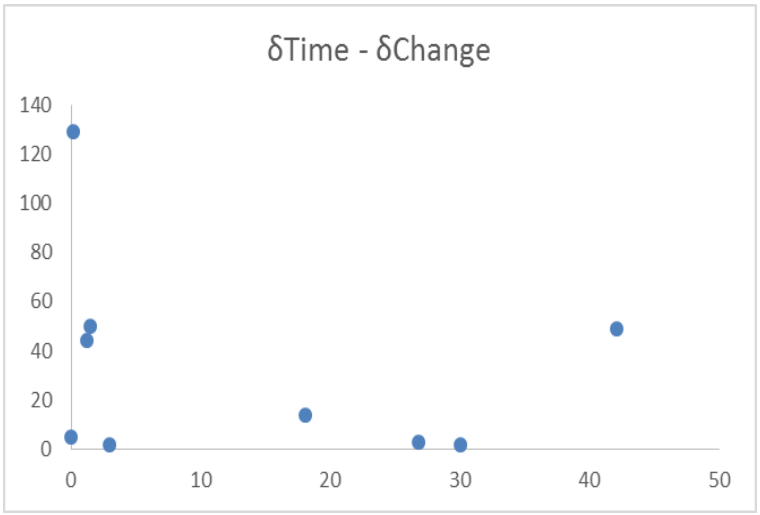


WC: 0.5	WT: 0.5	
PPC:ON	PPT:ON	
Phases	δtime	δc
0@1	16.96	4
1@2	11.59	3
2@3	56.32	2
3@4	13.05	2
4@5	96.87	6
5@6	64.05	12
6@7	30.65	12
7@8	28.81	5
8@9	26.85	3



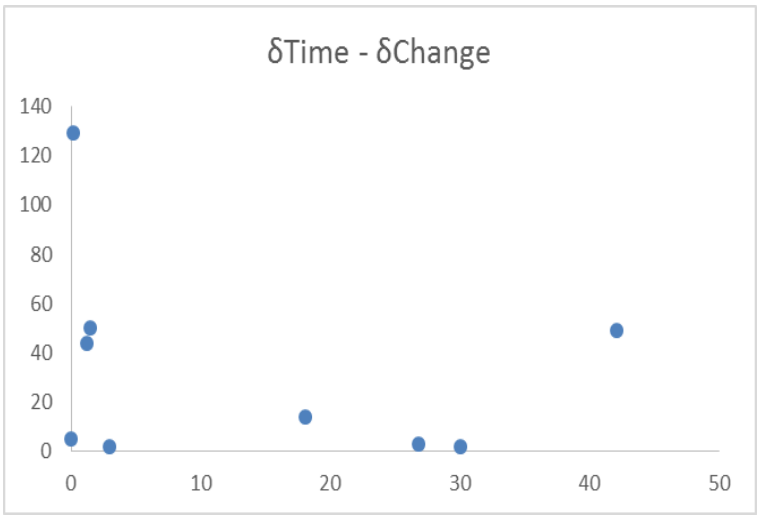
WC: 1.0 WT: 0.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	1.20	44
1@2	0.02	5
2@3	0.14	129
3@4	30.02	2
4@5	2.93	2
5@6	1.50	50
6@7	42.15	49
7@8	18.09	14
8@9	26.85	3



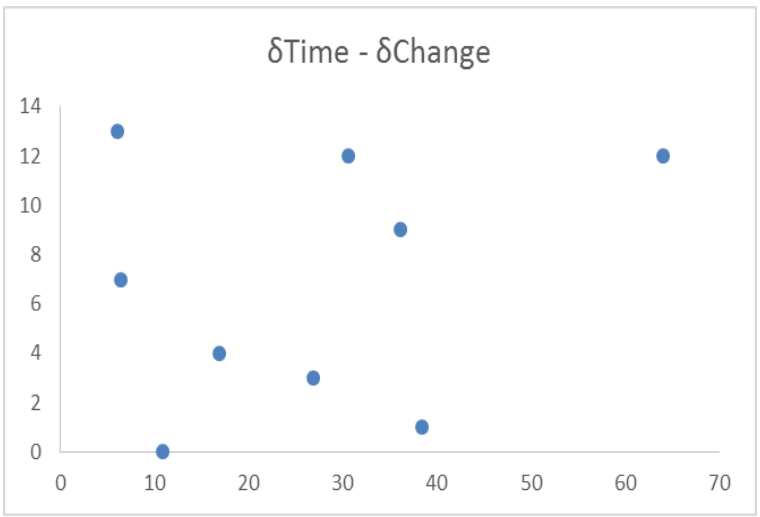
WC: 1.0 WT: 0.0
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	1.20	44
1@2	0.02	5
2@3	0.14	129
3@4	30.02	2
4@5	2.93	2
5@6	1.50	50
6@7	42.15	49
7@8	18.09	14
8@9	26.85	3



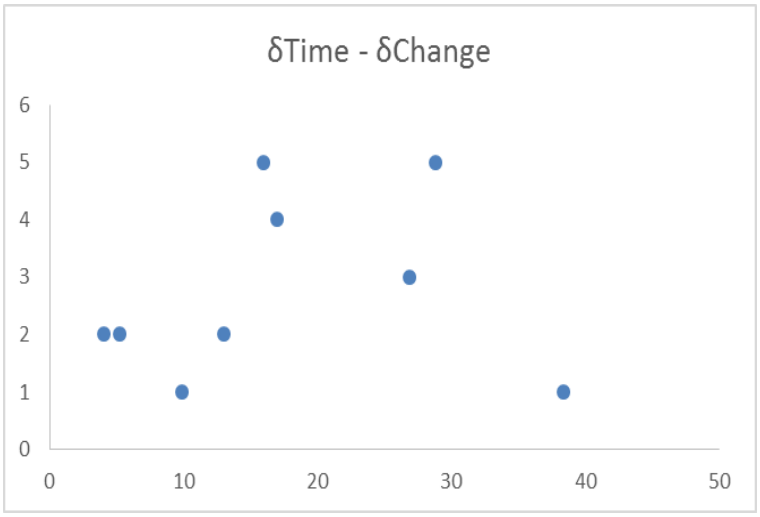
WC: 1.0 WT: 0.0
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	38.42	1
1@2	16.96	4
2@3	6.02	13
3@4	6.43	7
4@5	64.05	12
5@6	30.65	12
6@7	36.15	9
7@8	10.89	0
8@9	26.85	3



WC: 1.0 WT: 0.0
PPC:ON PPT:ON

Phases	δtime	δc
0@1	4.09	2
1@2	38.42	1
2@3	16.96	4
3@4	5.28	2
4@5	15.95	5
5@6	13.05	2
6@7	9.92	1
7@8	28.81	5
8@9	26.85	3

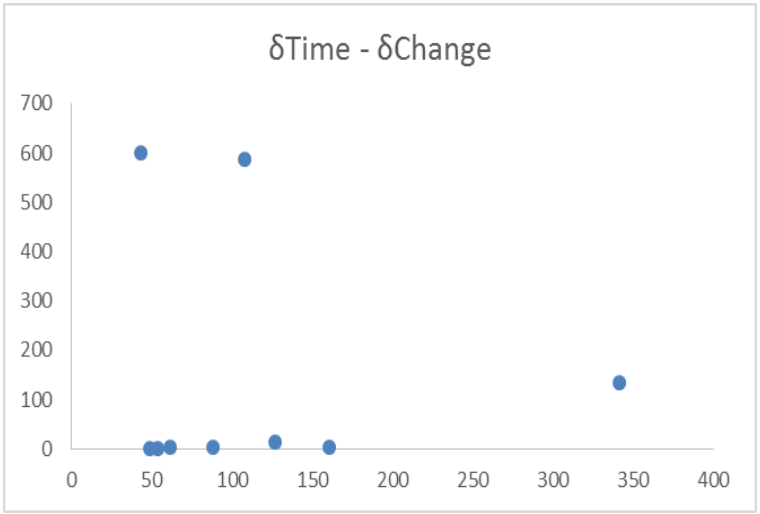


Opencart Dataset

Table A- 5: Assessment of phase extraction for Opencart

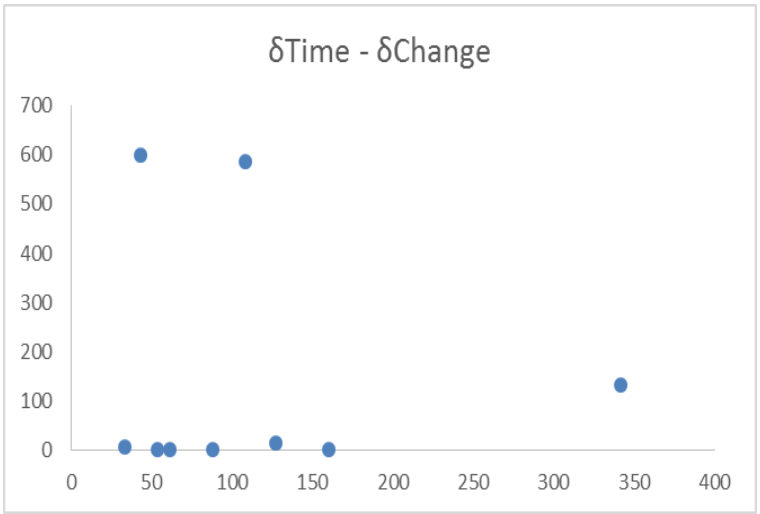
WC: 0.0 WT: 1.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	43.11	600
1@2	108.18	586
2@3	127.09	14
3@4	53.53	1
4@5	87.99	2
5@6	61.47	2
6@7	160.39	2
7@8	341.62	134
8@9	48.77	0



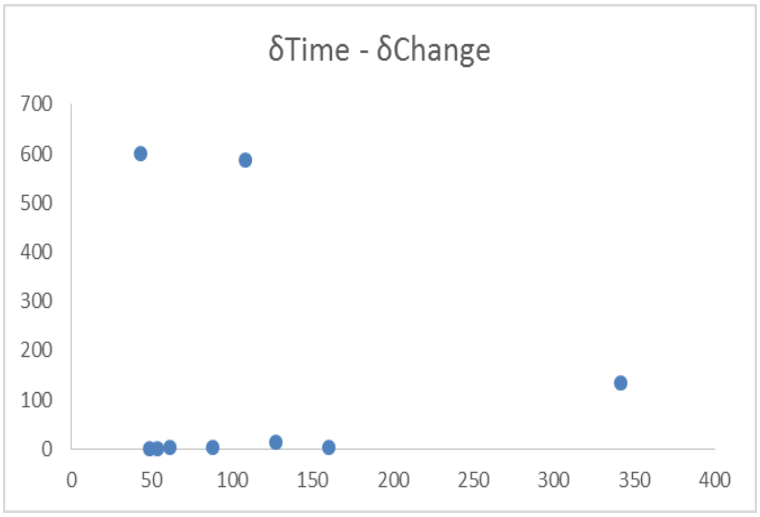
WC: 0.0 WT: 1.0
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	43.11	600
1@2	33.65	7
2@3	108.18	586
3@4	127.09	14
4@5	53.53	1
5@6	87.99	2
6@7	61.47	2
7@8	160.39	2
8@9	341.62	134



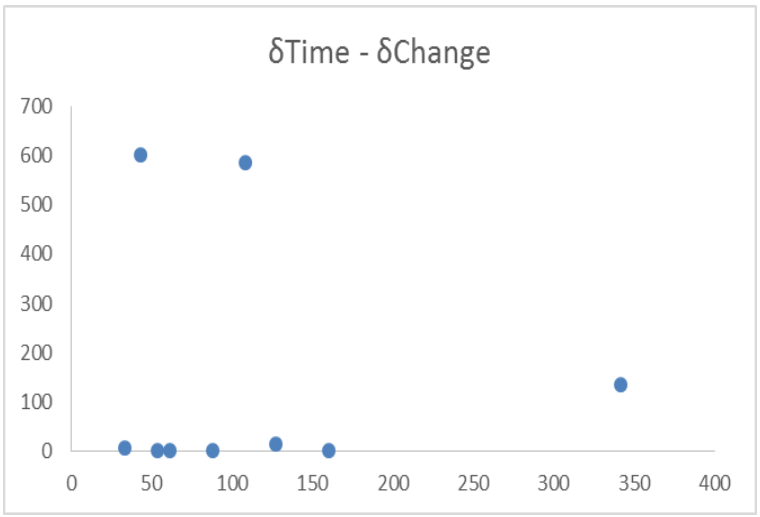
WC: 0.0 WT: 1.0
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	43.11	600
1@2	108.18	586
2@3	127.09	14
3@4	53.53	1
4@5	87.99	2
5@6	61.47	2
6@7	160.39	2
7@8	341.62	134
8@9	48.77	0



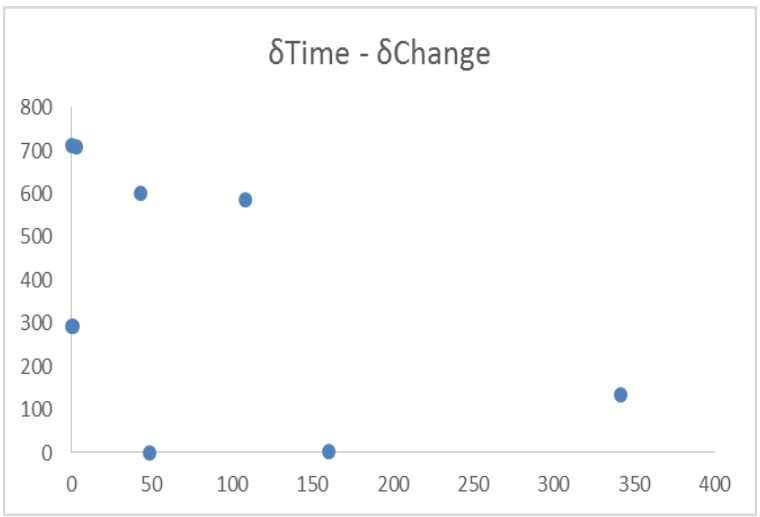
WC: 0.0 WT: 1.0
PPC:ON PPT:ON

Phases	δtime	δc
0@1	43.11	600
1@2	33.65	7
2@3	108.18	586
3@4	127.09	14
4@5	53.53	1
5@6	87.99	2
6@7	61.47	2
7@8	160.39	2
8@9	341.62	134



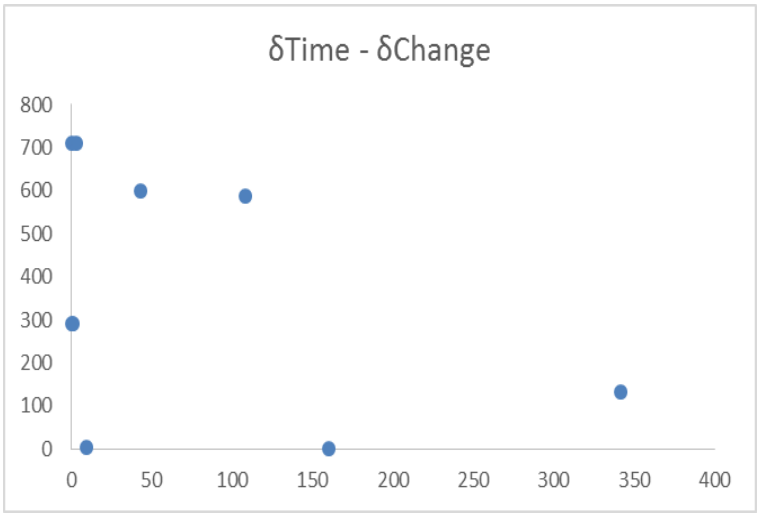
WC: 0.5 WT: 0.5
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	1.07	292
1@2	0.01	292
2@3	43.11	600
3@4	108.18	586
4@5	0.03	711
5@6	3.23	710
6@7	160.39	2
7@8	341.62	134
8@9	48.77	0



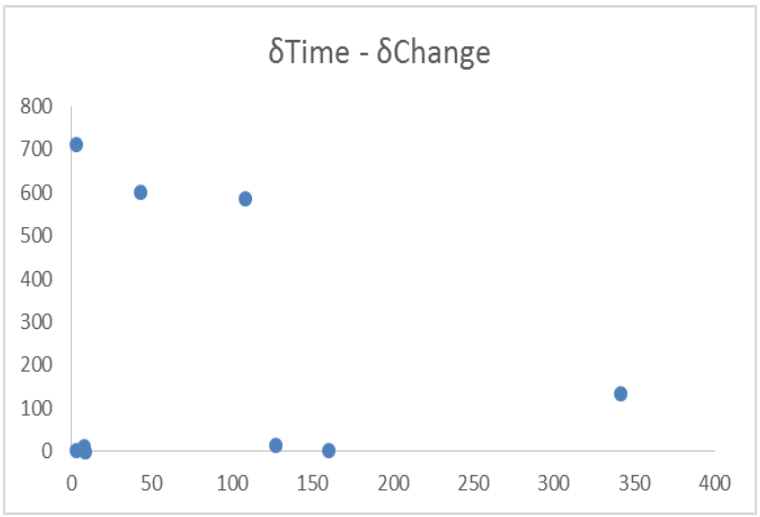
WC: 0.5 WT: 0.5
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	1.07	292
1@2	0.01	292
2@3	43.11	600
3@4	108.18	586
4@5	0.03	711
5@6	3.23	710
6@7	160.39	2
7@8	341.62	134
8@9	9.35	3



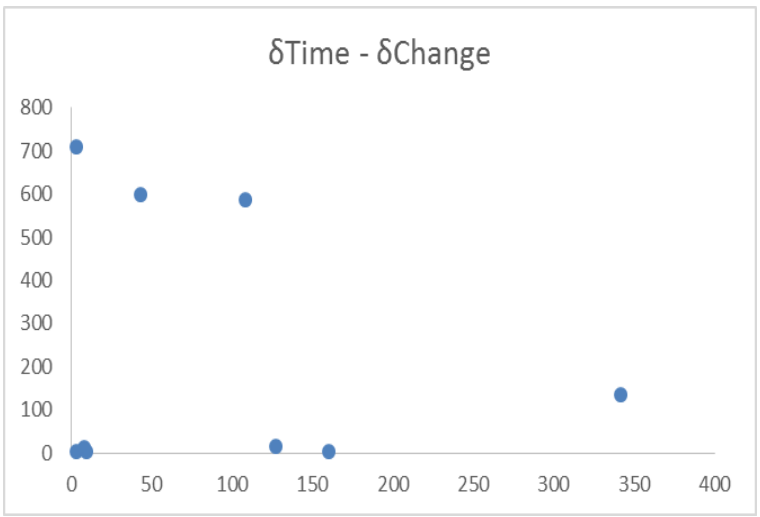
WC: 0.5 WT: 0.5
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	3.35	2
1@2	43.11	600
2@3	108.18	586
3@4	127.09	14
4@5	3.23	710
5@6	160.39	2
6@7	8.19	12
7@8	341.62	134
8@9	8.51	0



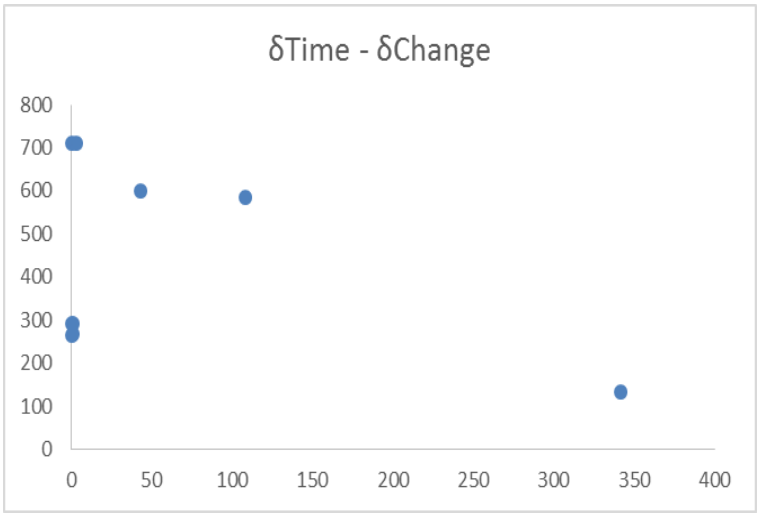
WC: 0.5 WT: 0.5
PPC:ON PPT:ON

Phases	δtime	δc
0@1	3.35	2
1@2	43.11	600
2@3	108.18	586
3@4	127.09	14
4@5	3.23	710
5@6	160.39	2
6@7	8.19	12
7@8	341.62	134
8@9	9.35	3



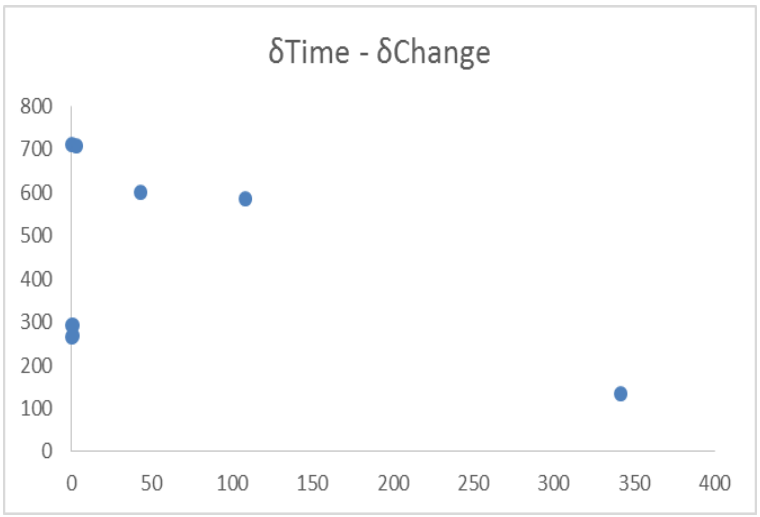
WC: 1.0 WT: 0.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	1.07	292
1@2	0.01	292
2@3	43.11	600
3@4	108.18	586
4@5	0.03	711
5@6	3.23	710
6@7	0.43	266
7@8	1.07	268
8@9	341.62	134



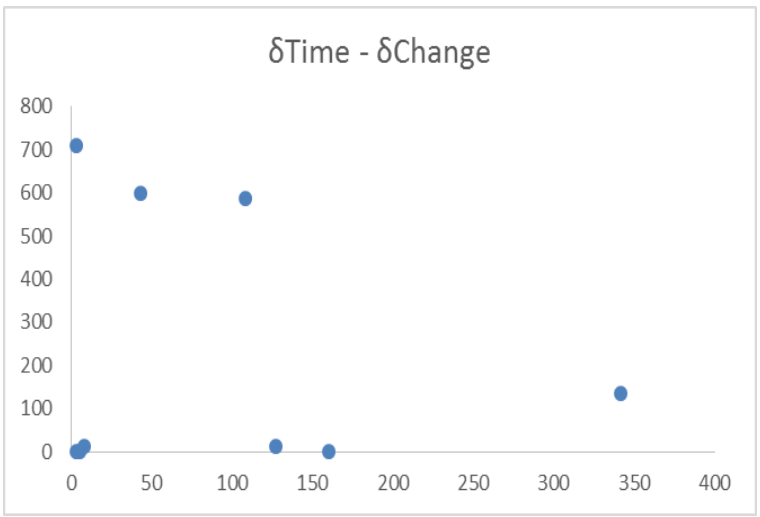
WC: 1.0 WT: 0.0
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	1.07	292
1@2	0.01	292
2@3	43.11	600
3@4	108.18	586
4@5	0.03	711
5@6	3.23	710
6@7	0.43	266
7@8	1.07	268
8@9	341.62	134



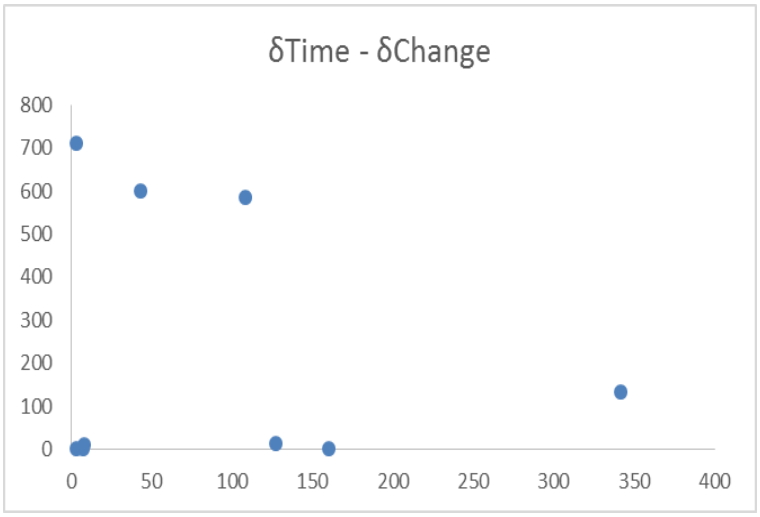
WC: 1.0 WT: 0.0
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	3.35	2
1@2	43.11	600
2@3	108.18	586
3@4	127.09	14
4@5	3.23	710
5@6	160.39	2
6@7	8.19	12
7@8	341.62	134
8@9	5.21	0



WC: 1.0 WT: 0.0
PPC:ON PPT:ON

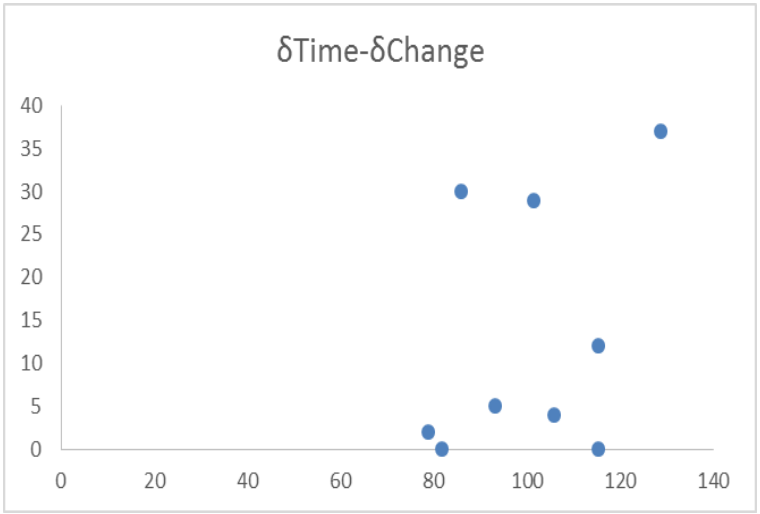
Phases	δtime	δc
0@1	3.35	2
1@2	43.11	600
2@3	108.18	586
3@4	127.09	14
4@5	3.23	710
5@6	160.39	2
6@7	8.19	12
7@8	341.62	134
8@9	7.40	2



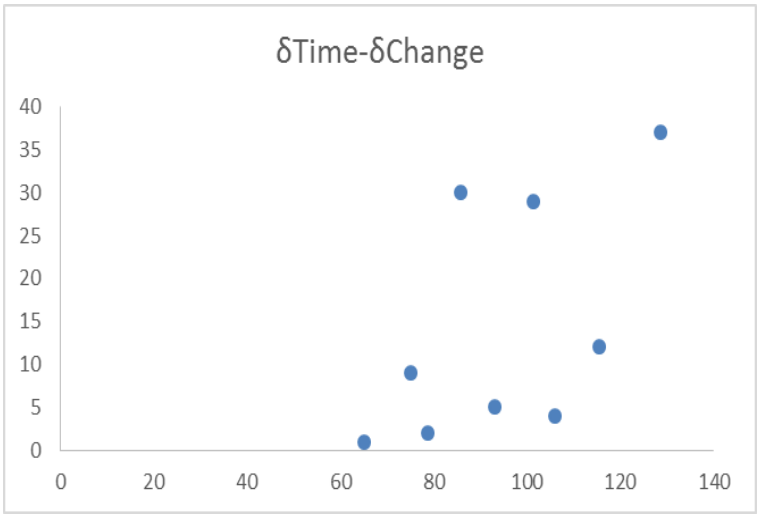
PhpBB Dataset

Table A- 6: Assessment of phase extraction for phpBB

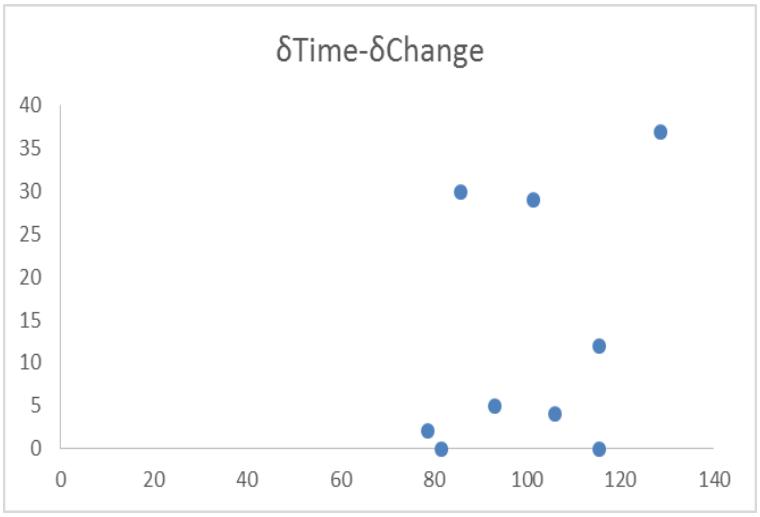
WC: 0.0	WT: 1.0	
PPC:OFF	PPT:OFF	
Phases	δtime	δc
0@1	93.09	5
1@2	115.39	12
2@3	81.67	0
3@4	115.42	0
4@5	128.67	37
5@6	105.90	4
6@7	85.88	30
7@8	101.33	29
8@9	78.75	2



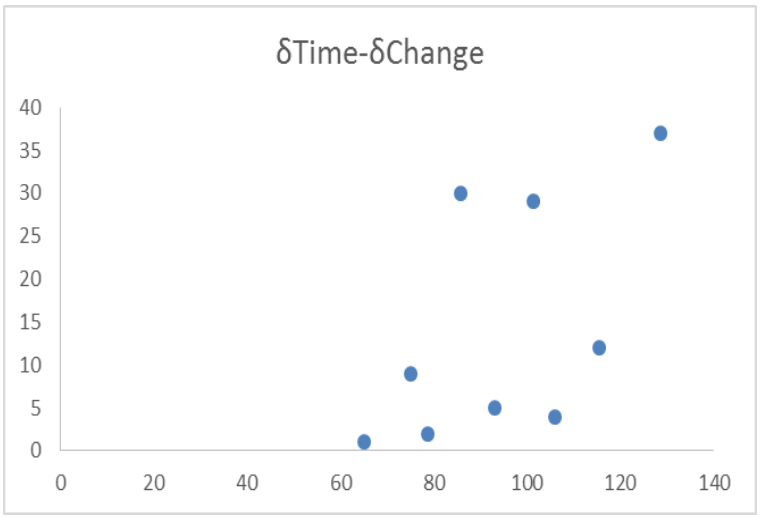
WC: 0.0	WT: 1.0	
PPC:ON	PPT:OFF	
Phases	δtime	δc
0@1	65.13	1
1@2	93.09	5
2@3	115.39	12
3@4	128.67	37
4@5	105.90	4
5@6	85.88	30
6@7	101.33	29
7@8	78.75	2
8@9	75.06	9



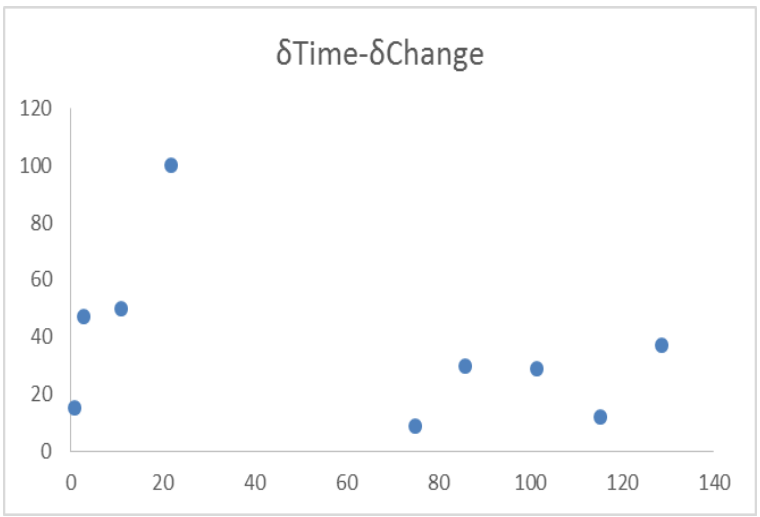
WC: 0.0	WT: 1.0	
PPC:OFF	PPT:ON	
Phases	δtime	δc
0@1	93.09	5
1@2	115.39	12
2@3	81.67	0
3@4	115.42	0
4@5	128.67	37
5@6	105.90	4
6@7	85.88	30
7@8	101.33	29
8@9	78.75	2



WC: 0.0	WT: 1.0	
PPC:ON	PPT:ON	
Phases	δtime	δc
0@1	65.13	1
1@2	93.09	5
2@3	115.39	12
3@4	128.67	37
4@5	105.90	4
5@6	85.88	30
6@7	101.33	29
7@8	78.75	2
8@9	75.06	9

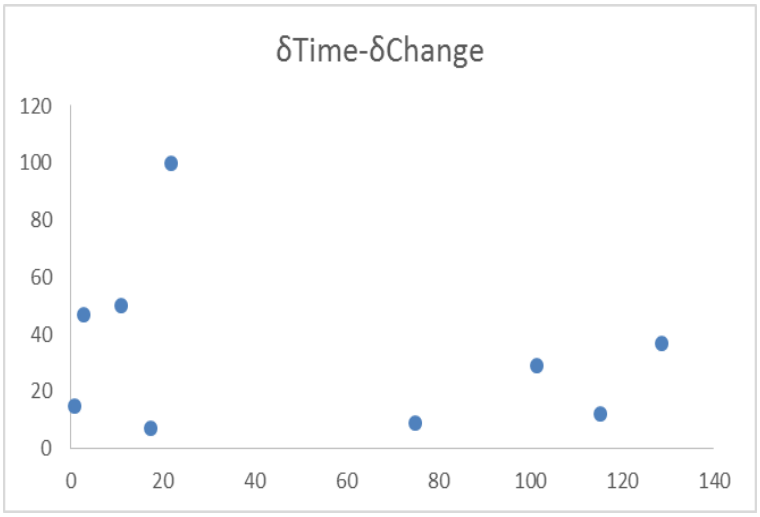


WC: 0.5	WT: 0.5	
PPC:OFF	PPT:OFF	
Phases	δtime	δc
0@1	21.68	100
1@2	115.39	12
2@3	128.67	37
3@4	85.88	30
4@5	101.33	29
5@6	10.93	50
6@7	2.69	47
7@8	75.06	9
8@9	0.85	15



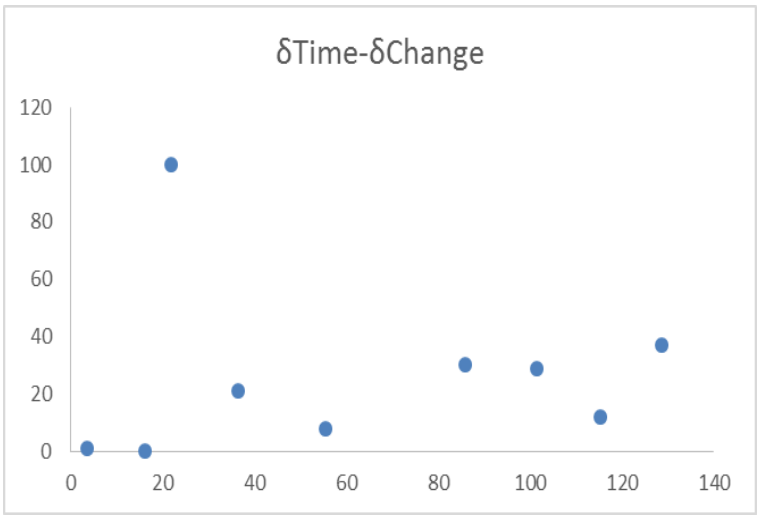
WC: 0.5 WT: 0.5
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	21.68	100
1@2	115.39	12
2@3	128.67	37
3@4	101.33	29
4@5	10.93	50
5@6	2.69	47
6@7	75.06	9
7@8	17.29	7
8@9	0.85	15



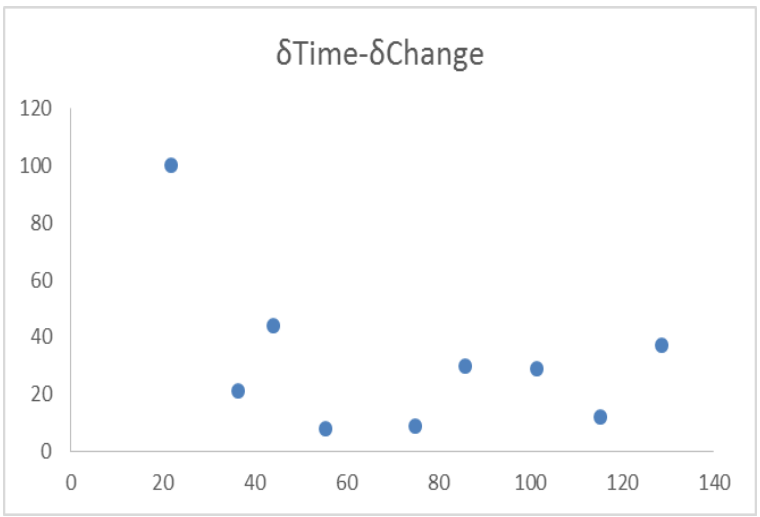
WC: 0.5 WT: 0.5
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	21.68	100
1@2	115.39	12
2@3	128.67	37
3@4	85.88	30
4@5	101.33	29
5@6	36.41	21
6@7	15.97	0
7@8	3.42	1
8@9	55.41	8



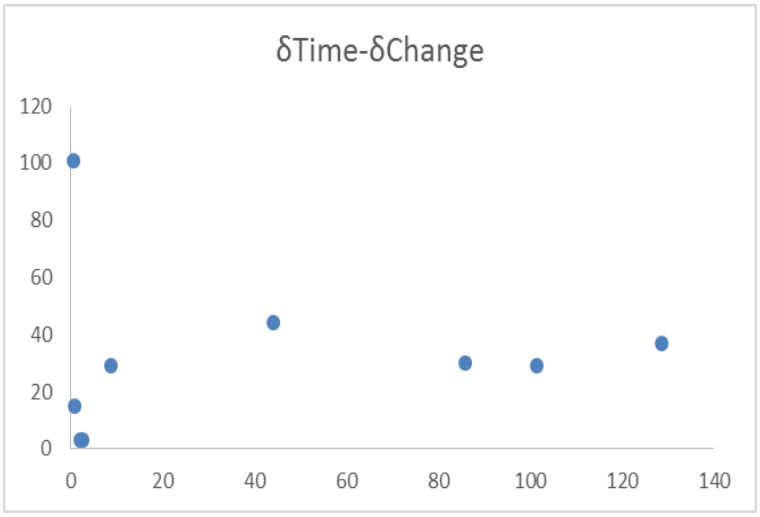
WC: 0.5 WT: 0.5
PPC:ON PPT:ON

Phases	δtime	δc
0@1	21.68	100
1@2	115.39	12
2@3	128.67	37
3@4	85.88	30
4@5	101.33	29
5@6	36.41	21
6@7	43.94	44
7@8	75.06	9
8@9	55.41	8



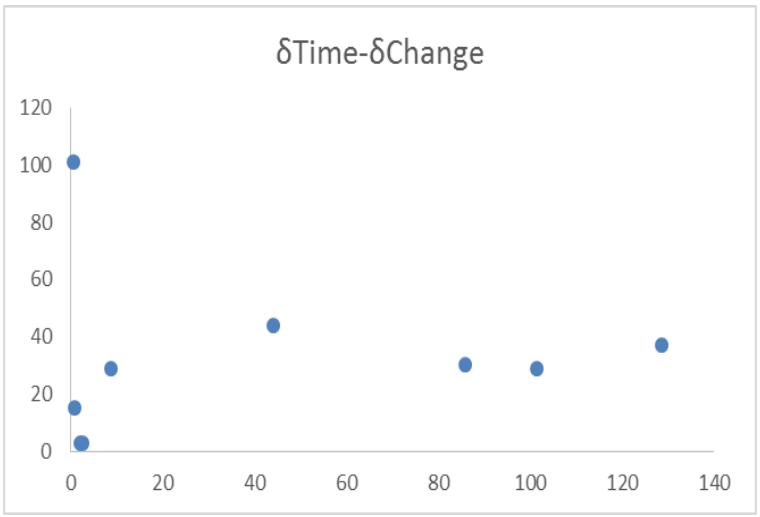
WC: 1.0 WT: 0.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	0.49	101
1@2	2.40	3
2@3	2.03	3
3@4	128.67	37
4@5	85.88	30
5@6	101.33	29
6@7	8.77	29
7@8	43.94	44
8@9	0.85	15



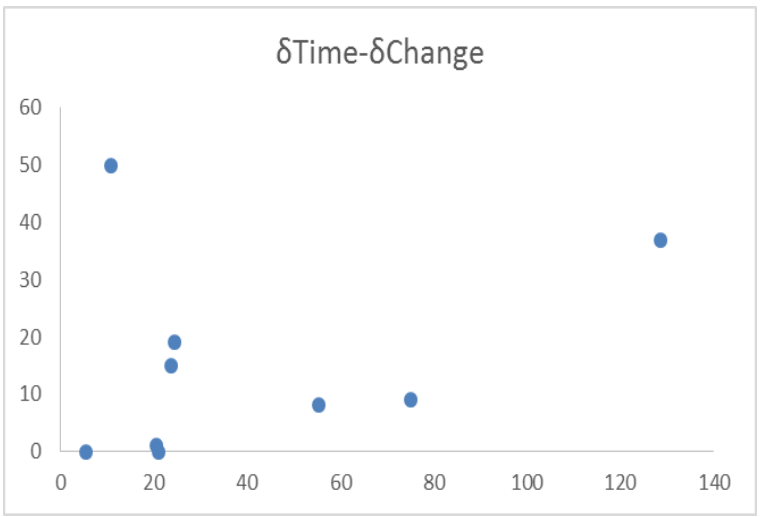
WC: 1.0 WT: 0.0
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	0.49	101
1@2	2.40	3
2@3	2.03	3
3@4	128.67	37
4@5	85.88	30
5@6	101.33	29
6@7	8.77	29
7@8	43.94	44
8@9	0.85	15



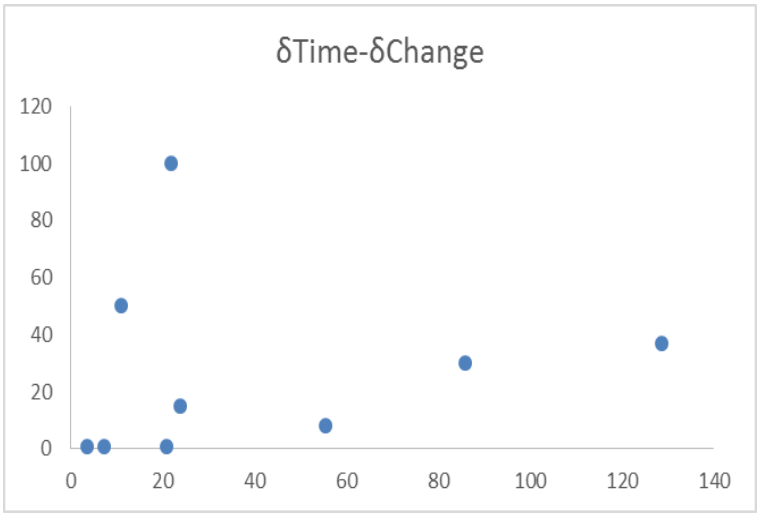
WC: 1.0 WT: 0.0
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	21.10	0
1@2	5.44	0
2@3	128.67	37
3@4	20.68	1
4@5	10.93	50
5@6	23.77	15
6@7	75.06	9
7@8	24.55	19
8@9	55.41	8



WC: 1.0 WT: 0.0
PPC:ON PPT:ON

Phases	δtime	δc
0@1	21.68	100
1@2	7.13	1
2@3	128.67	37
3@4	85.88	30
4@5	20.68	1
5@6	10.93	50
6@7	23.77	15
7@8	3.42	1
8@9	55.41	8

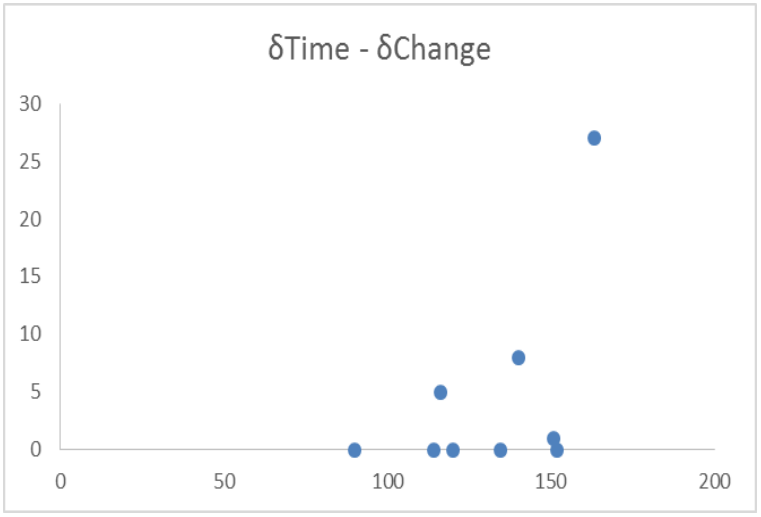


Typo3 Dataset

Table A- 7: Assessment of phase extraction for Typo3

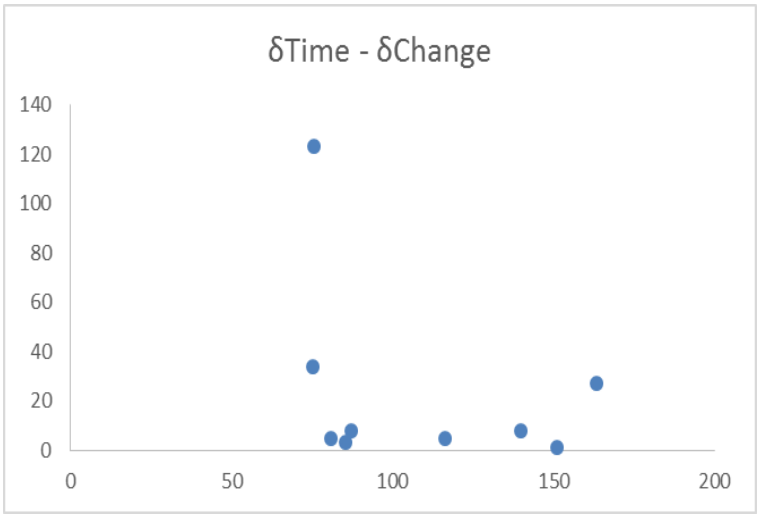
WC: 0.0 WT: 1.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	151.07	1
1@2	163.49	27
2@3	116.21	5
3@4	152.03	0
4@5	134.67	0
5@6	140.06	8
6@7	120.31	0
7@8	90.19	0
8@9	114.39	0

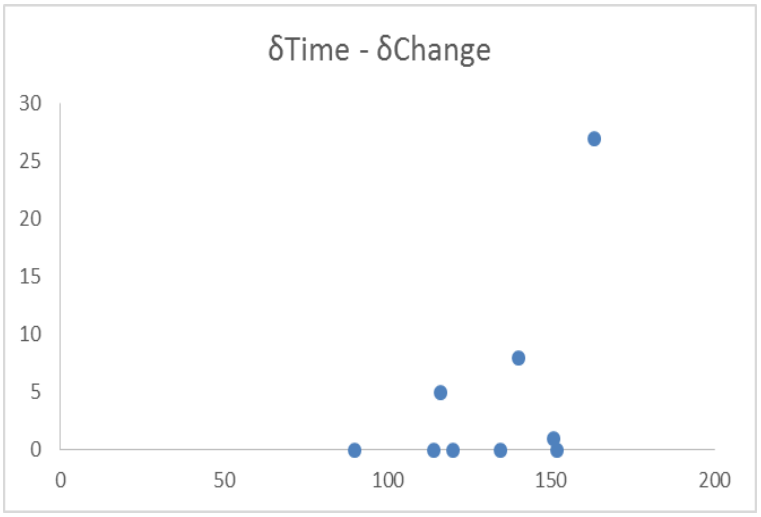


WC: 0.0 WT: 1.0
PPC:ON PPT:OFF

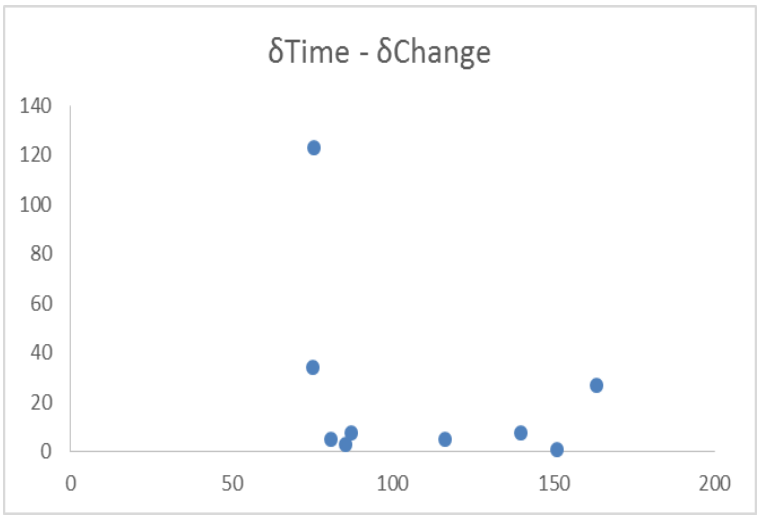
Phases	δtime	δc
0@1	151.07	1
1@2	85.31	3
2@3	163.49	27
3@4	116.21	5
4@5	80.76	5
5@6	140.06	8
6@7	87.32	8
7@8	75.33	34
8@9	75.72	123



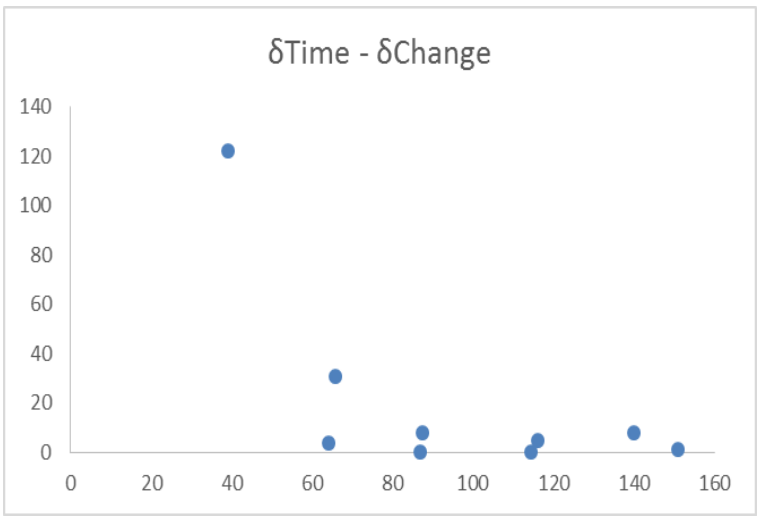
WC: 0.0	WT: 1.0	
PPC:OFF	PPT:ON	
Phases	δtime	δc
0@1	151.07	1
1@2	163.49	27
2@3	116.21	5
3@4	152.03	0
4@5	134.67	0
5@6	140.06	8
6@7	120.31	0
7@8	90.19	0
8@9	114.39	0



WC: 0.0	WT: 1.0	
PPC:ON	PPT:ON	
Phases	δtime	δc
0@1	151.07	1
1@2	85.31	3
2@3	163.49	27
3@4	116.21	5
4@5	80.76	5
5@6	140.06	8
6@7	87.32	8
7@8	75.33	34
8@9	75.72	123

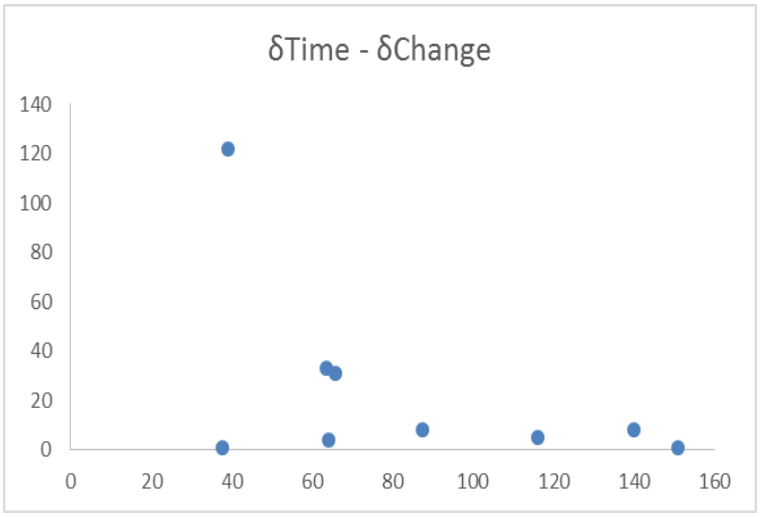


WC: 0.5	WT: 0.5	
PPC:OFF	PPT:OFF	
Phases	δtime	δc
0@1	151.07	1
1@2	116.21	5
2@3	65.86	31
3@4	140.06	8
4@5	114.39	0
5@6	64.24	4
6@7	87.32	8
7@8	39.20	122
8@9	86.79	0



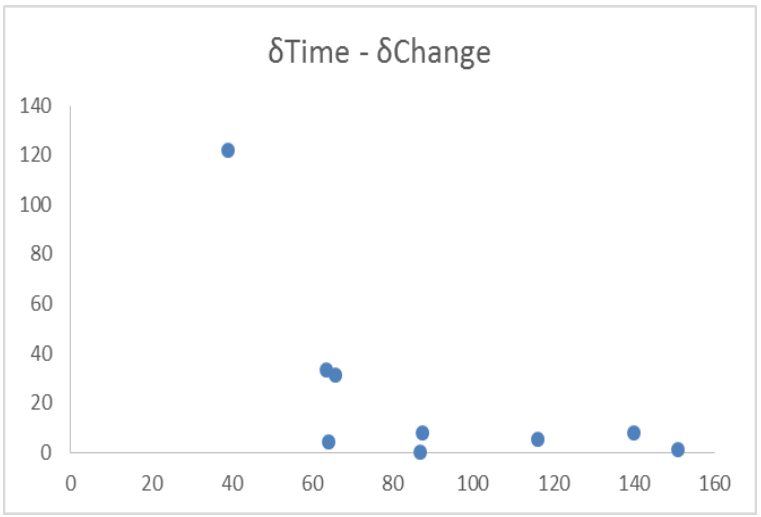
WC: 0.5 WT: 0.5
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	151.07	1
1@2	116.21	5
2@3	65.86	31
3@4	140.06	8
4@5	37.77	1
5@6	64.24	4
6@7	87.32	8
7@8	63.61	33
8@9	39.20	122



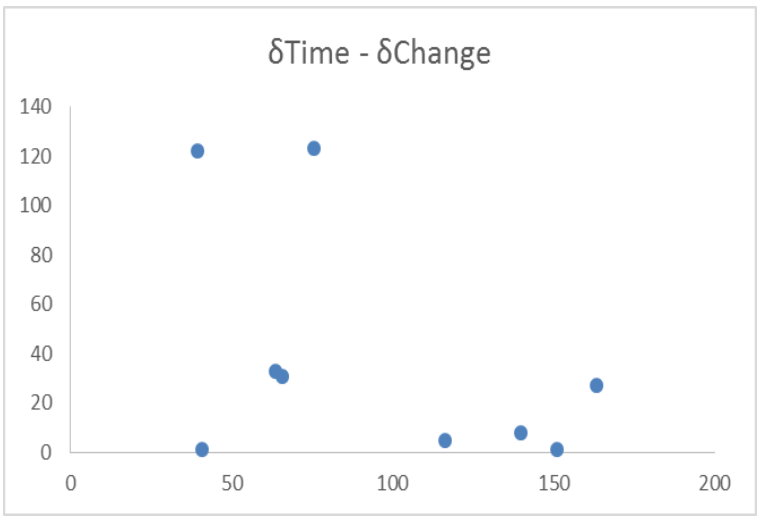
WC: 0.5 WT: 0.5
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	151.07	1
1@2	116.21	5
2@3	65.86	31
3@4	140.06	8
4@5	64.24	4
5@6	87.32	8
6@7	63.61	33
7@8	39.20	122
8@9	86.79	0



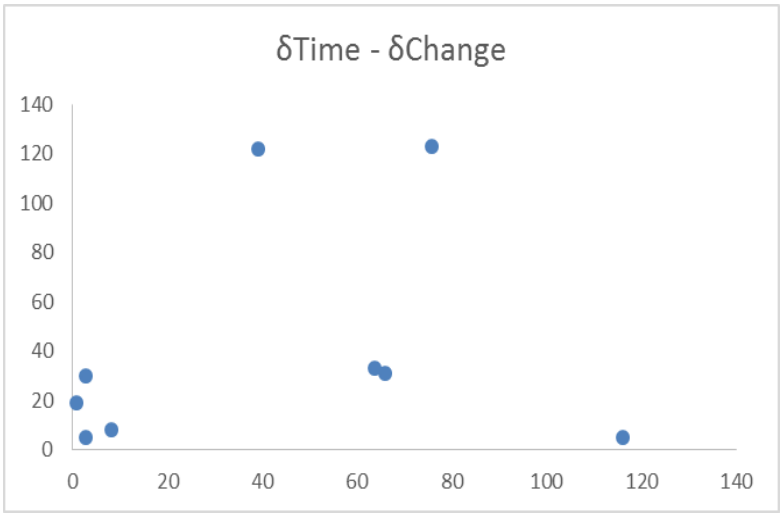
WC: 0.5 WT: 0.5
PPC:ON PPT:ON

Phases	δtime	δc
0@1	151.07	1
1@2	163.49	27
2@3	116.21	5
3@4	65.86	31
4@5	140.06	8
5@6	40.77	1
6@7	63.61	33
7@8	39.20	122
8@9	75.72	123



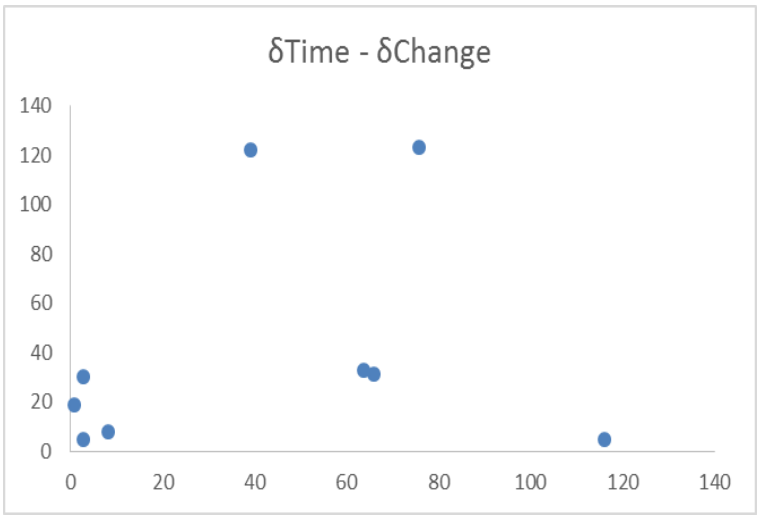
WC: 1.0 WT: 0.0
PPC:OFF PPT:OFF

Phases	δtime	δc
0@1	116.21	5
1@2	65.86	31
2@3	8.06	8
3@4	2.78	5
4@5	0.71	19
5@6	63.61	33
6@7	39.20	122
7@8	75.72	123
8@9	2.76	30



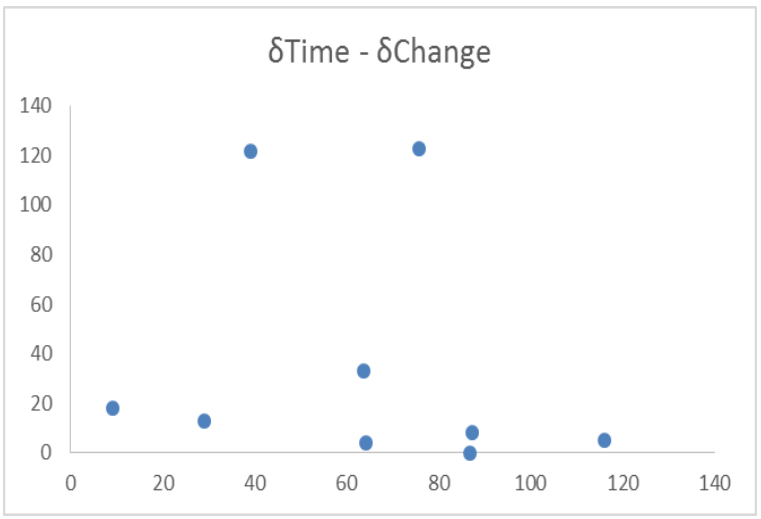
WC: 1.0 WT: 0.0
PPC:ON PPT:OFF

Phases	δtime	δc
0@1	116.21	5
1@2	65.86	31
2@3	8.06	8
3@4	2.78	5
4@5	0.71	19
5@6	63.61	33
6@7	39.20	122
7@8	75.72	123
8@9	2.76	30



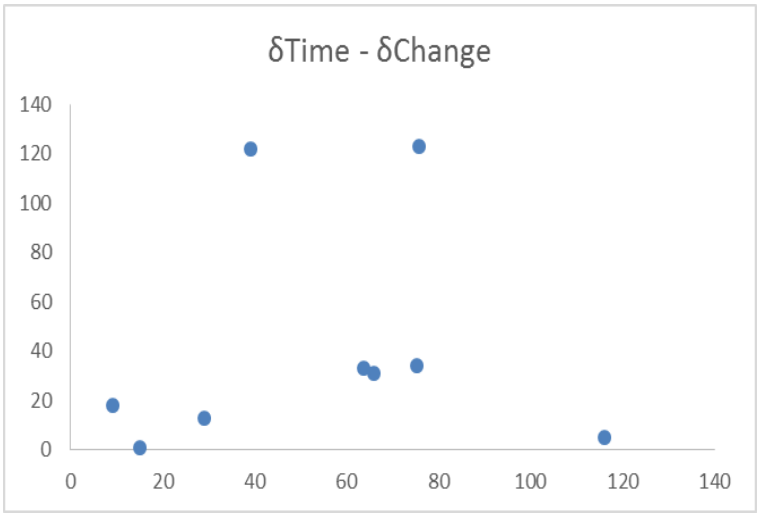
WC: 1.0 WT: 0.0
PPC:OFF PPT:ON

Phases	δtime	δc
0@1	116.21	5
1@2	9.00	18
2@3	29.06	13
3@4	64.24	4
4@5	87.32	8
5@6	63.61	33
6@7	39.20	122
7@8	75.72	123
8@9	86.79	0



WC: 1.0 WT: 0.0
PPC:ON PPT:ON

Phases	δtime	δc
0@1	116.21	5
1@2	65.86	31
2@3	9.00	18
3@4	29.06	13
4@5	15.04	1
5@6	75.33	34
6@7	63.61	33
7@8	39.20	122
8@9	75.72	123



Assessment of table clustering

Atlas Dataset

Table A- 8: Assessment of table clustering for Atlas

Wb: 0.333	Wd: 0.333	Wc: 0.333			
	Class 1	Class 2	Class 3	Class 4	Class 5
Precision					
Cluster 0	0.00	1.00	0.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00	0.00
Cluster 2	0.83	0.02	0.00	0.17	0.00
Cluster 3	0.00	0.00	0.74	0.11	0.16
Cluster 4	0.00	0.00	1.00	0.00	0.00
Recall					
Cluster 0	0.00	0.09	0.00	0.00	0.00
Cluster 1	0.00	0.82	0.00	0.00	0.00
Cluster 2	1.00	0.09	0.00	0.83	0.00
Cluster 3	0.00	0.00	0.93	0.17	1.00
Cluster 4	0.00	0.00	0.07	0.00	0.00
F-Measure					
Cluster 0	0.00	0.17	0.00	0.00	0.00
Cluster 1	0.00	0.90	0.00	0.00	0.00
Cluster 2	0.91	0.03	0.00	0.29	0.00
Cluster 3	0.00	0.00	0.82	0.13	0.27
Cluster 4	0.00	0.00	0.13	0.00	0.00

Wb: 0.0	Wd: 1.0	Wc: 0.0			
	Class 1	Class 2	Class 3	Class 4	Class 5
Precision					
Cluster 0	0.00	1.00	0.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00	0.00
Cluster 2	0.00	1.00	0.00	0.00	0.00
Cluster 3	0.40	0.20	0.00	0.00	0.60
Cluster 4	0.63	0.00	0.21	0.16	0.00
Recall					
Cluster 0	0.00	0.09	0.00	0.00	0.00
Cluster 1	0.00	0.73	0.00	0.00	0.00
Cluster 2	0.00	0.09	0.00	0.00	0.00
Cluster 3	0.04	0.09	0.00	0.00	1.00
Cluster 4	0.96	0.00	1.00	1.00	0.00
F-Measure					
Cluster 0	0.00	0.17	0.00	0.00	0.00
Cluster 1	0.00	0.84	0.00	0.00	0.00
Cluster 2	0.00	0.17	0.00	0.00	0.00
Cluster 3	0.08	0.13	0.00	0.00	0.75
Cluster 4	0.76	0.00	0.34	0.28	0.00

Wb: 0.0	Wd: 0.5	Wc: 0.5			
	Class 1	Class 2	Class 3	Class 4	Class 5
Precision					
Cluster 0	0.00	1.00	0.00	0.00	0.00
Cluster 1	0.66	0.00	0.22	0.13	0.00
Cluster 2	0.29	0.48	0.00	0.19	0.10
Cluster 3	0.00	0.00	0.00	0.00	1.00
Cluster 4	0.00	0.00	1.00	0.00	0.00
Recall					
Cluster 0	0.00	0.09	0.00	0.00	0.00
Cluster 1	0.88	0.00	0.93	0.67	0.00
Cluster 2	0.13	0.91	0.00	0.33	0.67
Cluster 3	0.00	0.00	0.00	0.00	0.33
Cluster 4	0.00	0.00	0.07	0.00	0.00
F-Measure					
Cluster 0	0.00	0.17	0.00	0.00	0.00
Cluster 1	0.75	0.00	0.35	0.21	0.00
Cluster 2	0.17	0.62	0.00	0.24	0.17
Cluster 3	0.00	0.00	0.00	0.00	0.50
Cluster 4	0.00	0.00	0.13	0.00	0.00

Wb: 0.0	Wd: 0.0	Wc: 1.0			
	Class 1	Class 2	Class 3	Class 4	Class 5
Precision					
Cluster 0	1.00	0.00	0.00	0.00	0.00
Cluster 1	0.61	0.13	0.13	0.13	0.04
Cluster 2	0.55	0.09	0.00	0.36	0.00
Cluster 3	0.52	0.13	0.21	0.10	0.04
Cluster 4	0.00	0.00	1.00	0.00	0.00
Recall					
Cluster 0	0.02	0.00	0.00	0.00	0.00
Cluster 1	0.29	0.27	0.20	0.25	0.33
Cluster 2	0.13	0.09	0.00	0.33	0.00
Cluster 3	0.56	0.64	0.73	0.42	0.67
Cluster 4	0.00	0.00	0.07	0.00	0.00
F-Measure					
Cluster 0	0.04	0.00	0.00	0.00	0.00
Cluster 1	0.39	0.18	0.16	0.17	0.08
Cluster 2	0.20	0.09	0.00	0.35	0.00
Cluster 3	0.54	0.22	0.33	0.16	0.07
Cluster 4	0.00	0.00	0.13	0.00	0.00

Wb: 0.5	Wd: 0.5	Wc: 0.0			
	Class 1	Class 2	Class 3	Class 4	Class 5
Precision					
Cluster 0	0.00	1.00	0.00	0.00	0.00
Cluster 1	0.75	0.16	0.00	0.11	0.00
Cluster 2	0.00	0.00	0.00	1.00	0.00
Cluster 3	0.00	0.00	0.00	0.00	1.00
Cluster 4	0.00	0.00	1.00	0.00	0.00
Recall					
Cluster 0	0.00	0.09	0.00	0.00	0.00
Cluster 1	1.00	0.91	0.00	0.58	0.00
Cluster 2	0.00	0.00	0.00	0.42	0.00
Cluster 3	0.00	0.00	0.00	0.00	1.00
Cluster 4	0.00	0.00	1.00	0.00	0.00
F-Measure					
Cluster 0	0.00	0.17	0.00	0.00	0.00
Cluster 1	0.86	0.27	0.00	0.18	0.00
Cluster 2	0.00	0.00	0.00	0.59	0.00
Cluster 3	0.00	0.00	0.00	0.00	1.00
Cluster 4	0.00	0.00	1.00	0.00	0.00

Wb: 0.5	Wd: 0.0	Wc: 0.5			
	Class 1	Class 2	Class 3	Class 4	Class 5
Precision					
Cluster 0	0.00	1.00	0.00	0.00	0.00
Cluster 1	0.72	0.15	0.00	0.15	0.00
Cluster 2	0.00	0.00	0.76	0.06	0.18
Cluster 3	0.00	0.00	0.50	0.50	0.00
Cluster 4	0.00	0.00	1.00	0.00	0.00
Recall					
Cluster 0	0.00	0.09	0.00	0.00	0.00
Cluster 1	1.00	0.91	0.00	0.83	0.00
Cluster 2	0.00	0.00	0.87	0.08	1.00
Cluster 3	0.00	0.00	0.07	0.08	0.00
Cluster 4	0.00	0.00	0.07	0.00	0.00
F-Measure					
Cluster 0	0.00	0.17	0.00	0.00	0.00
Cluster 1	0.83	0.26	0.00	0.25	0.00
Cluster 2	0.00	0.00	0.81	0.07	0.30
Cluster 3	0.00	0.00	0.12	0.14	0.00
Cluster 4	0.00	0.00	0.13	0.00	0.00

Wb: 1.0	Wd: 0.0	Wc: 0.0			
	Class 1	Class 2	Class 3	Class 4	Class 5
Precision					
Cluster 0	0.80	0.18	0.00	0.03	0.00
Cluster 1	0.00	0.00	0.00	1.00	0.00
Cluster 2	0.00	0.00	0.00	0.63	0.38
Cluster 3	0.00	0.00	1.00	0.00	0.00
Cluster 4	0.00	0.00	1.00	0.00	0.00
Recall					
Cluster 0	1.00	1.00	0.00	0.17	0.00
Cluster 1	0.00	0.00	0.00	0.42	0.00
Cluster 2	0.00	0.00	0.00	0.42	1.00
Cluster 3	0.00	0.00	0.93	0.00	0.00
Cluster 4	0.00	0.00	0.07	0.00	0.00
F-Measure					
Cluster 0	0.89	0.31	0.00	0.06	0.00
Cluster 1	0.00	0.00	0.00	0.59	0.00
Cluster 2	0.00	0.00	0.00	0.50	0.55
Cluster 3	0.00	0.00	0.97	0.00	0.00
Cluster 4	0.00	0.00	0.13	0.00	0.00

phpBB Dataset

Table A- 9: Assessment of table clustering for phpBB

Wb: 0.333	Wd: 0.333	Wc: 0.333		
	Class 1	Class 2	Class 3	Class 4
Precision				
Cluster 0	0.00	1.00	0.00	0.00
Cluster 1	0.08	0.90	0.00	0.02
Cluster 2	0.00	0.00	1.00	0.00
Cluster 3	0.00	0.00	0.00	1.00
Recall				
Cluster 0	0.00	0.02	0.00	0.00
Cluster 1	1.00	0.98	0.00	0.20
Cluster 2	0.00	0.00	1.00	0.00
Cluster 3	0.00	0.00	0.00	0.80
F-Measure				
Cluster 0	0.00	0.03	0.00	0.00
Cluster 1	0.15	0.94	0.00	0.03
Cluster 2	0.00	0.00	1.00	0.00
Cluster 3	0.00	0.00	0.00	0.89

Wb: 0.0	Wd: 1.0	Wc: 0.0		
	Class 1	Class 2	Class 3	Class 4
Precision				
Cluster 0	0.60	0.00	0.40	0.00
Cluster 1	0.67	0.00	0.33	0.00
Cluster 2	0.00	0.98	0.00	0.02
Cluster 3	0.00	0.00	0.00	1.00
Recall				
Cluster 0	0.60	0.00	0.67	0.00
Cluster 1	0.40	0.00	0.33	0.00
Cluster 2	0.00	1.00	0.00	0.20
Cluster 3	0.00	0.00	0.00	0.80
F-Measure				
Cluster 0	0.60	0.00	0.50	0.00
Cluster 1	0.50	0.00	0.33	0.00
Cluster 2	0.00	0.99	0.00	0.03
Cluster 3	0.00	0.00	0.00	0.89

Wb: 0.0	Wd: 0.5	Wc: 0.5		
	Class 1	Class 2	Class 3	Class 4
Precision				
Cluster 0	1.00	0.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00
Cluster 2	0.00	1.00	0.00	0.00
Cluster 3	0.05	0.83	0.05	0.08
Recall				
Cluster 0	0.40	0.00	0.00	0.00
Cluster 1	0.00	0.02	0.00	0.00
Cluster 2	0.00	0.02	0.00	0.00
Cluster 3	0.60	0.96	1.00	1.00
F-Measure				
Cluster 0	0.57	0.00	0.00	0.00
Cluster 1	0.00	0.03	0.00	0.00
Cluster 2	0.00	0.03	0.00	0.00
Cluster 3	0.08	0.89	0.09	0.14

Wb: 0.0	Wd: 0.0	Wc: 1.0		
	Class 1	Class 2	Class 3	Class 4
Precision				
Cluster 0	0.00	1.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00
Cluster 2	0.00	1.00	0.00	0.00
Cluster 3	0.07	0.81	0.04	0.07
Recall				
Cluster 0	0.00	0.02	0.00	0.00
Cluster 1	0.00	0.02	0.00	0.00
Cluster 2	0.00	0.02	0.00	0.00
Cluster 3	1.00	0.95	1.00	1.00
F-Measure				
Cluster 0	0.00	0.03	0.00	0.00
Cluster 1	0.00	0.03	0.00	0.00
Cluster 2	0.00	0.03	0.00	0.00
Cluster 3	0.14	0.87	0.09	0.14

Wb: 0.5	Wd: 0.5	Wc: 0.0		
	Class 1	Class 2	Class 3	Class 4
Precision				
Cluster 0	1.00	0.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00
Cluster 2	0.00	0.00	1.00	0.00
Cluster 3	0.00	0.00	0.00	1.00
Recall				
Cluster 0	1.00	0.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00
Cluster 2	0.00	0.00	1.00	0.00
Cluster 3	0.00	0.00	0.00	1.00
F-Measure				
Cluster 0	1.00	0.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00
Cluster 2	0.00	0.00	1.00	0.00
Cluster 3	0.00	0.00	0.00	1.00

Wb: 0.5	Wd: 0.0	Wc: 0.5		
	Class 1	Class 2	Class 3	Class 4
Precision				
Cluster 0	0.00	1.00	0.00	0.00
Cluster 1	0.00	1.00	0.00	0.00
Cluster 2	0.07	0.82	0.04	0.06
Cluster 3	0.00	0.00	0.00	1.00
Recall				
Cluster 0	0.00	0.02	0.00	0.00
Cluster 1	0.00	0.02	0.00	0.00
Cluster 2	1.00	0.96	1.00	0.80
Cluster 3	0.00	0.00	0.00	0.20
F-Measure				
Cluster 0	0.00	0.03	0.00	0.00
Cluster 1	0.00	0.03	0.00	0.00
Cluster 2	0.14	0.89	0.09	0.11
Cluster 3	0.00	0.00	0.00	0.33

Wb: 1.0	Wd: 0.0	Wc: 0.0		
	Class 1	Class 2	Class 3	Class 4
Precision				
Cluster 0	0.08	0.92	0.00	0.00
Cluster 1	0.00	0.00	1.00	0.00
Cluster 2	0.00	0.00	0.00	1.00
Cluster 3	0.00	0.00	0.00	1.00
Recall				
Cluster 0	1.00	1.00	0.00	0.00
Cluster 1	0.00	0.00	1.00	0.00
Cluster 2	0.00	0.00	0.00	0.80
Cluster 3	0.00	0.00	0.00	0.20
F-Measure				
Cluster 0	0.15	0.96	0.00	0.00
Cluster 1	0.00	0.00	1.00	0.00
Cluster 2	0.00	0.00	0.00	0.89
Cluster 3	0.00	0.00	0.00	0.33

Coppermine Dataset

Table A- 10: Assessment of table clustering for Coppermine

Wb: 0.333	Wd: 0.333	Wc: 0.333
	Class 1	Class 2
Precision		
Cluster 0	1.00	0.00
Cluster 1	0.86	0.14
Recall		
Cluster 0	0.05	0.00
Cluster 1	0.95	1.00
F-Measure		
Cluster 0	0.10	0.00
Cluster 1	0.90	0.24

Wb: 0.0	Wd: 1.0	Wc: 0.0
	Class 1	Class 2
Precision		
Cluster 0	0.86	0.14
Cluster 1	1.00	0.00
Recall		
Cluster 0	0.95	1.00
Cluster 1	0.05	0.00
F-Measure		
Cluster 0	0.90	0.24
Cluster 1	0.10	0.00

Wb: 0.0	Wd: 0.5	Wc: 0.5
	Class 1	Class 2
Precision		
Cluster 0	1.00	0.00
Cluster 1	0.86	0.14
Recall		
Cluster 0	0.05	0.00
Cluster 1	0.95	1.00
F-Measure		
Cluster 0	0.10	0.00
Cluster 1	0.90	0.24

Wb: 0.0	Wd: 0.0	Wc: 1.0
	Class 1	Class 2
Precision		
Cluster 0	1.00	0.00
Cluster 1	0.86	0.14
Recall		
Cluster 0	0.05	0.00
Cluster 1	0.95	1.00
F-Measure		
Cluster 0	0.10	0.00
Cluster 1	0.90	0.24

Wb: 0.5	Wd: 0.5	Wc: 0.0
	Class 1	Class 2
Precision		
Cluster 0	0.86	0.14
Cluster 1	1.00	0.00
Recall		
Cluster 0	0.95	1.00
Cluster 1	0.05	0.00
F-Measure		
Cluster 0	0.90	0.24
Cluster 1	0.10	0.00

Wb: 0.5	Wd: 0.0	Wc: 0.5
	Class 1	Class 2
Precision		
Cluster 0	0.95	0.05
Cluster 1	0.33	0.67
Recall		
Cluster 0	0.95	0.33
Cluster 1	0.05	0.67
F-Measure		
Cluster 0	0.95	0.09
Cluster 1	0.09	0.67

Wb: 1.0	Wd: 0.0	Wc: 0.0
	Class 1	Class 2
Precision		
Cluster 0	1.00	0.00
Cluster 1	0.00	1.00
Recall		
Cluster 0	1.00	0.00
Cluster 1	0.00	1.00
F-Measure		
Cluster 0	1.00	0.00
Cluster 1	0.00	1.00