Derivation of state diagrams for database schema evolution

A Thesis

submitted to the designated
by the Assembly
of the Department of Computer Science and Engineering
Examination Committee

by

Christina Trialoni

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN DATA AND COMPUTER SYSTEMS ENGINEERING

WITH SPECIALIZATION
IN ADVANCED COMPUTER SYSTEMS

University of Ioannina
School of Engineering
Ioannina 2022

Examining Committee:

- **Panos Vassiliadis,** Professor, Department of Computer Science and Engineering, University of Ioannina (Advisor)
- **Evaggelia Pitoura,** Professor, Department of Computer Science and Engineering, University of Ioannina
- **Apostolos Zarras,** Professor, Department of Computer Science and Engineering, University of Ioannina

DEDICATION

I would like to dedicate the thesis to my family and everyone else that has been close to me.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Professor Panos Vassiliadis, for my master thesis. He was always present throughout the duration of the thesis and without his invaluable help I would not have made it this far. He always offered his insights and help during the research and writing phases of the thesis.

I would also like to thank my friends for all of their support and understanding during this time.

Finally, I would love to thank my family, for all the love and support they have showed me, as well as for helping me believe in myself, that I can achieve greater goals.

Ioannina, July 2022

Christina Trialoni

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

ABSTRACT

Christina Trialoni, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, July 2022
Derivation of state diagrams for database schema evolution
Advisor: Panagiotis Vassiliadis, Professor

Schema evolution is the process of altering the structure of a database, also known as "schema", via the insertion, deletion or update of schema constructs, such as tables, attributes and constraints, in the process of developing, or maintaining the structure of the data, in order to service the surrounding applications that -- as all software modules do -- evolve too.
The goal of this thesis is to extract the various phases that a schema of a project enters during its lifecycle, and create "signatures" of frequently encountered sequences of phases in the lives of relational database schemata.
Using a publicly available corpus of schema evolution histories from Free Open-Source Projects, we organize the history of corpus' projects in monthly quanta as time units and assess change via a cumulative metric of monthly change. Starting with each time-unit as a different phase, the PhaseSeries of schema evolution for a project is then, a sequence of unit-phases, linked via transitions marking the amount of change (measured as the sum of inserted, deleted and updated attributes) that occurred between two units. A transition is the bridge that connects two phases and it is also labelled with respect to the amount of change between the two neighboring phases. Then, we merge subsequent unit-phases into larger phases depending on a similarity criterion that takes into consideration transition labels, and mark the resulting phases accordingly. We introduce different algorithms for merging phases, by altering the similarity criterion with the goal of finding the sweet spot between

having too many transitions (contributing a high level of accuracy) and conciseness (as having fewer transitions improves readability of the description of a project's life, at the price of reducing accuracy). We refer to the description of the life of a schema via these phases and transitions as the "signature" of the schema's evolution. Once all signatures for the entire corpus have been computed, we group them into frequently encountered signatures, introducing, thus, frequent patterns of schema lives. These patterns can be visually demonstrated via state diagrams.

Εκτεταμενη περιληψη

Χριστίνα Τριαλώνη, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, Ιούλιος 2022
Δημιουργία διαγραμμάτων καταστάσεων για την εξέλιξη σχήματος σε βάσεις δεδομένων
Επιβλέπων: Παναγιώτης Βασιλειάδης, Καθηγητής

Εξέλιξη σχήματος βάσεων δεδομένων, είναι η διαδικασία αλλαγής της δομής μιας βάσης, ή αλλιώς "σχήματος", μέσω προσθήκης, διαγραφής ή αλλαγής στοιχείων του σχήματος, όπως πίνακες, χαρακτηριστικά και περιορισμοί κατά τη διάρκεια της φάσης ανάπτυξης, ή η συντήρηση της δομής των δεδομένων, έτσι ώστε να εξυπηρετηθούν όλες οι συσχετιζόμενες εφαρμογές, οι οποίες -- όπως και όλα τα κομμάτια λογισμικού -- εξελίσσονται. Ο στόχος της διπλωματικής είναι η εξαγωγή διαφόρων φάσεων, στις οποίες μπαίνει ένα σχήμα ενός πρότζεκτ κατά τη διάρκεια του κύκλου ζωής του, και η δημιουργία "υπογραφών" των πιο συχνά εμφανιζόμενων ακολουθιών φάσεων κατά τη διάρκεια ζωής των σχεσιακών σχημάτων βάσεων. Χρησιμοποιώντας μια δημόσια συλλογή από ιστορίες εξέλιξης σχημάτων από ελεύθερα έργα ελεύθερου κώδικα, οργανώνουμε την ιστορία των συστημάτων σε μηνιαία κβάντα ως σημεία χρόνου και ορίζουμε τις αλλαγές μέσω σωρευτικών μετρικών μηνιαίων αλλαγών. Ξεκινώντας με κάθε χρονική μονάδα ως μια διαφορετική φάση, οι χρονοσειρές της εξέλιξης σχήματος για ένα έργο είναι η σειρά μονάδων φάσεων, συνδεδεμένων μέσω μεταβάσεων που ορίζουν την ποσότητα αλλαγής (μετρημένη ως το άθροισμα των χαρακτηριστικών που εισάχθηκαν, διαγράφτηκαν και αλλάχθηκαν) που συνέβησαν μεταξύ δύο φάσεων. Μια μετάβαση είναι η γέφυρα η οποία ενώνει δύο φάσεις και επισημειώνεται ανάλογα με τον αριθμό αλλαγών μεταξύ δύο γειτονικών φάσεων. Έπειτα, συνενώνουμε γειτονικές μονάδες φάσεων σε μεγαλύτερες φάσεις, στη βάση ενός κριτηρίου ομοιότητας το οποίο λαμβάνει υπ᾽ όψιν

τους χαρακτηρισμούς των μεταβάσεων και καθορίζει τις φάσεις που προκύπτουν. Παρουσιάζουμε διαφορετικούς αλγορίθμους για την ένωση φάσεων, εναλλάσσοντας το κριτήριο ομοιότητας με στόχο την εύρεση της χρυσής τομής μεταξύ του να έχουμε πολλές μεταβάσεις (το οποίο συμβάλλει στο να έχουμε μεγάλη ακρίβεια) και περιεκτικότητα (το να έχουμε λιγότερες μεταβάσεις βελτιώνει την αναγνωσιμότητα της περιγραφής της ζωής ενός έργου, με κόστος την μείωση ακρίβειας). Αναφερόμαστε στην περιγραφή της ζωής ενός σχήματος μέσω αυτών των φάσεων και των μεταβάσεων ως την "υπογραφή" της εξέλιξης του σχήματος. Μόλις όλες οι υπογραφές για όλη τη συλλογή των έργων έχουν υπολογιστεί, τις ομαδοποιούμε σε υπογραφές που εμφανίζονται συχνά, παρουσιάζοντας έτσι, τα πιο συχνά πρότυπα της ζωής των σχημάτων. Αυτά τα πρότυπα μπορούν να αναπαρασταθούν οπτικά μέσω διαγραμμάτων καταστάσεων.

CHAPTER 1

INTRODUCTION

1.1 Goals

The world around us keeps evolving with a rapid pace. The market is growing offering new technologies and creating new needs. This impact has also been apparent in the computer science field and specifically in software development and maintenance.

Software development is the process of designing, creating and maintaining applications and everything else that directly impacts the application, such as the database used to store and access the data. The contents of a database, such as the tables, fields, relationships, functions, procedures and constraints make up the schema. Schema evolution is how any database can be altered with respect to its internal structure from the development phase up to the maintenance phase of the application. Changes can consist of insertions, deletions, additions, merges and more, and they can happen during any part of the development process or during the maintenance timeframe. They can also happen frequently, or more scarcely, or never.

The field of schema evolution is relatively new, however in the last few years more and more studies emerge, despite the small number of open-source databases. Most of these studies focus on figuring the various states a schema can enter and aim to help developers "predict" the course of an application during its lifecycle, in order

to organize and distribute time and resources evenly and with as little risk as possible. In one of the studies [5], a tool called UML-Diff was created in order to help figure all changes in the classes of a system. Using this information, the researchers managed to extract various phases the development can enter and group them into categories of similar traits. In another study [13], one of the goals was the extraction of different characteristics of schema evolution types, which are called taxa. Taxa basically label the number of changes during the schema evolution of a database.
Both of the aforementioned studies have provided methods and tools to extract phases [5] and categorize schema evolution types based on the history of activity [13], respectively. However, they haven't managed to find specific phases and categories of schema evolution patterns. In this thesis we combine the general idea and findings in studies [5] and [13], in order to find evolution categories in schemata.
Using the results of [13] and the histories of its 195 databases, the goals of the thesis are the following:

- The representation of histories of the schemata in phases.
- The clustering of the "histories with phases" into homogenous clusters.
- For each cluster, the creation of a state diagram.

To accomplish that, 3 steps are followed.

- Firstly, the PhaseSeries of the project are created. To do so, the Phases and Transitions are extracted from the result files of each project.

  The Phases are the different states during specific timeframes, while the schema evolves. Regarding to that, a Phase can consist of one or more timeframes with their respective changes, which are called Atomic Measurements. In other words, an Atomic Measurement includes the percentage of total number of changes and percentage of time.

  The Transitions are the bridges that connect the neighboring Phases. In more detail, a Transition is the passage of a Phase or state of a project into a different Phase state. So, for example if a project has two Phases, where during the first one a lot of changes occurred, while in the second there were no changes at all, then those Phases create a Transition between them. Each Phase is a cohesive group of continuous time points with similar change rates and the transition between the phases marks a change in the change rate. A

transition can be labelled depending on the difference between the Phases it connects.

The labels are the following:

- Flat: Indicates no changes between the Phases
- Low: The neighboring Phases have a small number of changes between them
- Regular: A normal number of changes between the Phases
- Steep: A significant number of changes between the Phases.

– The second step includes merging the various Phases and Transitions in order to create a shorter version of the PhaseSeries, while keeping the most important details. The merging process can happen under various conditions, which can include the degree of similarity of labels, or certain labels themselves, or duration of Phases depending on the project's duration. In this project, 4 different conditions are used in the algorithms and they are analyzed further. One point that has to be considered, is the cost of each approach. If many Phases are merged, then accuracy and information is lost, however if the opposite approach is used then we end up with a lot of accuracy, meaning more Phases, with extra Transitions.
– The third step includes the grouping of all similar PhaseSeries, in order to create the state diagrams and categorize them.

In a nutshell, the final goal is to reach a merging condition that allows us to create PhaseSeries that do not include a lot of information on each schema (Phases/Transitions), while not losing a lot of accuracy. When this condition is met, the various state diagrams are created and categorized. The result is the various categories of state diagrams that indicate the different characteristics of database evolution.

1.2 Structure of the Thesis

The thesis consists of 5 main chapters:
In chapter 1, the current one, we discuss the main goals of the thesis.
In chapter 2, the related work is presented.
In chapter 3, the whole process is presented in detail, while providing various examples on the different procedures.
In chapter 4, the experiments and results are presented, as well as the conclusions reached.
In chapter 5, the conclusions and future work are presented.

CHAPTER 2

RELATED WORK

2.1 Introduction

What is schema evolution and why is it important to observe it?
Schema evolution is the process of altering the structure of a database from an older to a newer version, typically done by inserting, removing, or altering the constituent elements of the schema of the database [14].
All applications require databases in order to store, use, create data and their entire functionality depends on them. Schema evolution affects all the applications, since the databases they use, are altered. Such changes can lead to errors during run time or errors that require alteration of the code. So, it is highly important for developers to be able to predict, expect and understand any possible changes of their applications' schemata during their lifetime.

2.2 Related Work on Software and Schema Evolution

2.2.1 Lehman Laws

Concerning the field of software evolution, studies begun around 50 years ago and the most important findings appeared in the mid-2010s. The first extensive study had as a result a set of rules called Lehman Laws [1] [2] [3]. These rules focus on feedback-based systems on software evolution and include the following laws:

- Law of continuing change - A software, or system, must be evolving continuously in order to fulfill any new requirements.
- Law of Increasing Complexity - A system that is evolving is also increasing in complexity, unless the process of refactoring is done to it.
- Law of Self-Regulation - This means that the system is regulated by feedback.
- Law of Conservation of Organizational Stability - The work rate of an organization that evolves the system, tends to be continuous over the lifetime of the system.
- Law of Familiarity - The growth ratio depends on the need to maintain familiarity.
- Law of Continuing Growth - User satisfaction must be maintained all times; therefore, a system needs to be growing during all of its lifetime, in order to fulfill the users' needs.
- Law of Declining Quality - If not all needs are met and there's no evolution or adaptation of a system, then its use and quality also decline.
- Law of Feedback System - The evolution processes are multi-level, multi-loop, multi-agent feedback systems.

Schema and database evolution is a field which has not been explored extensively.

2.2.2 Thesaurus Tool and Impact of Schema Changes on Systems

In 1993, a research paper was published by Dag Sjøberg [4], in which the goal was to create a method in order to define and measure the categories and types of changes which occur during the evolution of databases/application systems, as well as the consequences. To achieve this goal, a tool called Thesaurus was used and a big scale functional database was monitored. The database belonged to a health

management system and it was tested for 18 months. The findings of the research effort showed the fact that schema changes have a big impact on an application throughout its lifecycle, including the development and gone-live phase. Changes include a big number of relation and field additions, but a smaller number of deletions. With so many alterations in a system, a change management tool was needed to monitor the situation. This is the reason Thesaurus was used for this specific study. This tool analyzed the database schema and application programs, extracted all information about the application changes made by the developers and estimated any changes created by the schema changes.

2.2.3 UMLDiff Tool

In 2005, a new tool called UMLDiff, was introduced [5]. This tool could recognize the design-level changes that have happened during the system's evolution. Based on the results of this research, analyses on 3 different development steps are performed in order to recognize patterns and gain an insight on potential development sequences like additions or deletions.

- The first step begins during the development process. Just an input of the systems versions is required and then the reverse engineering of the models in initiated. The last part is a product of the UMLDiff differentiations. This whole process is implemented in the JDE, which is a plugin of the Eclipse IDE.
- The second step includes the detection of changes from one version to another. The researchers of this study had already made a case study on JFreeChart and smaller ones on the UMLDiff algorithm which helped them with this phase. So, they used that in order to detect the version changes that are relevant to the software development process.
- During the third and final phase, the various class-evolution profiles have been obtained and discretized and each set of version changes is classified in terms of a five-class taxonomy. With these results and using 3 analyses, it is possible to gain insight for the development of the system. These analyses include:
    - Phasic analysis. In this analysis, the evolution profiles are segmented into coherent phases. These Phases include "Start with" Phase and

"End with" Phase, which indicate the creation and the final alteration of a class respectively. Additionally, if a class gets removed, then its "End with Phase" is also its "Remove with" Phase. Every alteration that occurs in between is included include "In the middle" Phase. These Phases can also be characterized by the number of changes the classes included. For example, any of these Phases may have included classes with "Intense evolution", "Rapid development", "Restructures", "Slow development" or "Steady State".

- Gamma analysis. In this analysis the patterns are identified. To reach this goal, the relative order of the various phase types in the class evolution profiles is examined, in order to figure the relationships between them. So, for each possible Phase pairing, a score is calculated and finally a gamma map is created. The gamma map displays all of the phases in their precedence order and abstracts the overall sequential pattern from a phase map.
- Optimal matching analysis. In this analysis the most frequent patterns during the evolution of the system are identified. To achieve this goal, clusters of phases of classes that evolved in a similar manner for a specific period or not.

2.2.4 VTracker and Evolution of Webservices

In 2011 a paper on the evolution on web services [6] studied, how the changes in WSDL files affect client applications. In order to define those impacts, a tool called VTracker was used and 5 web services were being monitored. The VTracker tool is a domain agnostic tree differencing algorithm. As a result of this study, 2 points were made.

- First, the tool was successful in identifying changes and determining effective and ineffective maintenance scenarios.
- Finally, web services are usually growing, rather than being altered. Generally, additions of new features or changes don't affect the clients. However, deletions should be avoided as they can cause crashes on client applications.

2.2.5 How the Development Process is Affected by the Schema Changes

In 2013, a study [7] which had as a main goal to figure how code development is affected by schema changes, in order to provide data for the creation of an assistive tool, to be used during database evolution, was introduced. The approach of this research was focused on 3 key points.

- The frequency of the database evolution
- The volume and types of changes of database evolution
- The code co-evolution with the database

To this end, a certain process was followed in order to retrieve information needed. First of all, the schema files were located and then the databases were extracted. Out of those, only the valid ones were kept, which contain the most changes. After that, the atomic changes were extracted by manually figuring the differences of the schemata versions. Lastly, analysis on co-evolution can be done. The results of this research gave answers to the first key questions. More specifically, turns out that the schemas evolve frequently and mostly during early development, they grow in size and gain columns as they evolve. Regarding the way databases evolve, 6 different categories were observed.

- Transformations
- Structure Refactoring
- Data Quality Refactoring
- Referential Integrity Refactoring
- Method Refactoring
- Architectural Refactoring

Out of these categories, the first 3 appeared more frequently and the final one relatively infrequently. Regarding the atomic changes, additions of tables and alteration/additions of columns were the most frequent. Data also showed very little impact of the existence of constraints of procedures. Finally, all additions and changes were used mostly during evolution and as for a final point the results showed that schema changes can impact code and the development process greatly.

2.2.6 Autoregressive Moving Average Models

In 2014 a study [8] used Autoregressive Moving Average (ARMA) models to analyze and model stationary Time Series. Time Series are sequential measurements or observations of a phenomenon of interest through time. During this, 3 phases were needed to be implemented to create the models.

- The first phase includes the model identification, which requires the data preparation and then the model selection by using various estimations out of the autocorrelation and partial correlation functions.
- The second Phase includes model estimation and Testing. 3 steps are being followed, such as estimation, diagnostics and model selection.
- The final Phase is about forecasting and simulation. It is extremely important to choose the best model for forecasting and simulation of the dynamics.

The approach mentioned above, was used on various systems and the evaluation showed that the models can predict accurately with minor errors. The results of this research can be used to predict the time required and budget needed for the creation of applications.

2.2.7 Study on Lehman Laws Applications

A study emerged during 2014 and it focused on the Lehman Laws and their application to the database evolution [9]. During this study, 9 Open-Source databases where being monitored and 3 rules were extracted.

- The first rule is called "Feedback-Based Behavior for Schema Evolution". It indicates that the size of schema grows and the changes happening are dependent on the user needs.
- The second one, "HeartBeat of change" explains that changes come in outbursts, therefore not a linear process.
- And the last one is "Schema Growth is Small ", which is almost self-explanatory. Basically, schema overall shows minimal changes during its lifetime.

2.2.8 Patterns Derived from Schema Evolution Properties

The next study was published in 2015, focused on determining the patterns regarding table properties [10]. 4 patterns were extracted from the research.

- The Γ pattern. This pattern implies that schemata that are large are less likely to be deleted and they have long lifetime.

- Comet pattern. This pattern explains that tables with medium sized schemata tend to have the most updates.
- The inverse Γ pattern. This pattern indicates that tables that don't survive long, produce less updates than the ones that survive the longest.
- The Empty Triangle pattern is a mix of all of the above. Deleted tables appear to have the shortest life duration, whereas older tables have a very low probability of deletion.

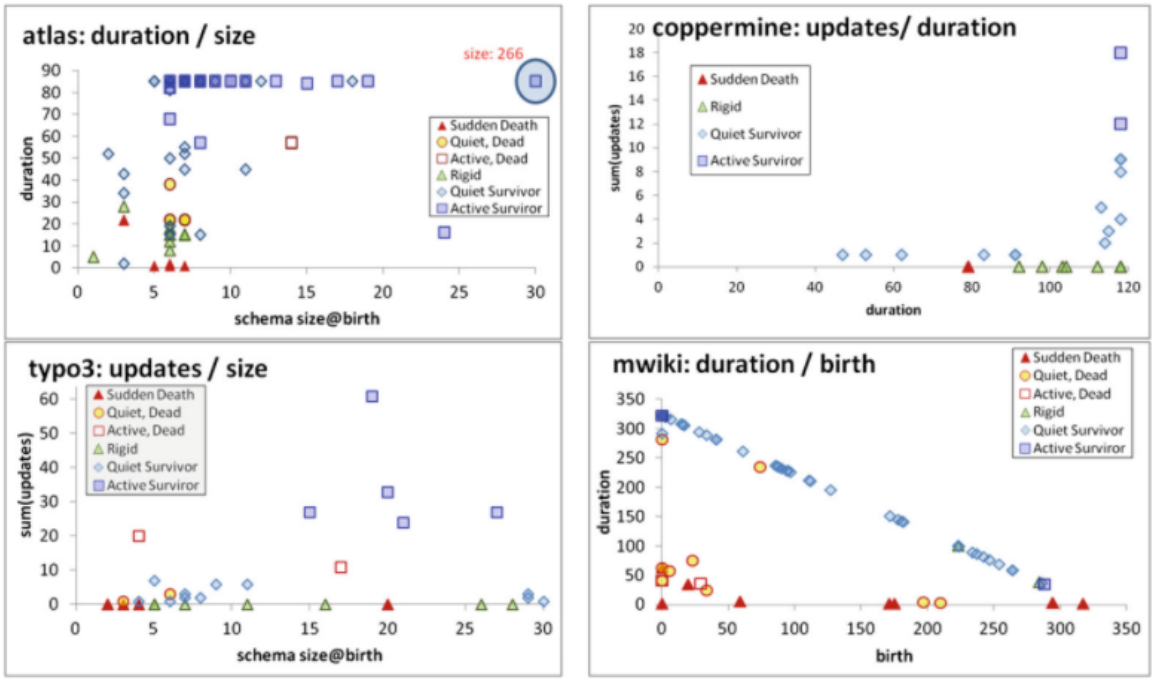

Figure 2.1 The 4 patterns of: Gamma (top left), inverse Gamma (top right), comet (bottom left) and empty triangle (bottom right). [10] (figure reproduced with author permission)

2.2.9 Electrolysis Pattern

The study of 2017 focused more on how survival is related to the duration of tables as well as their activity profile [11].

Just one pattern was derived from this study and it focuses on the antitheses that appear between the active tables and their life cycle versus inactive tables. It is called Electronics pattern. It explains that tables that have had a short life cycle with very few updates are most likely dead, whereas, active tables have lived longer and attracted plenty of updates.

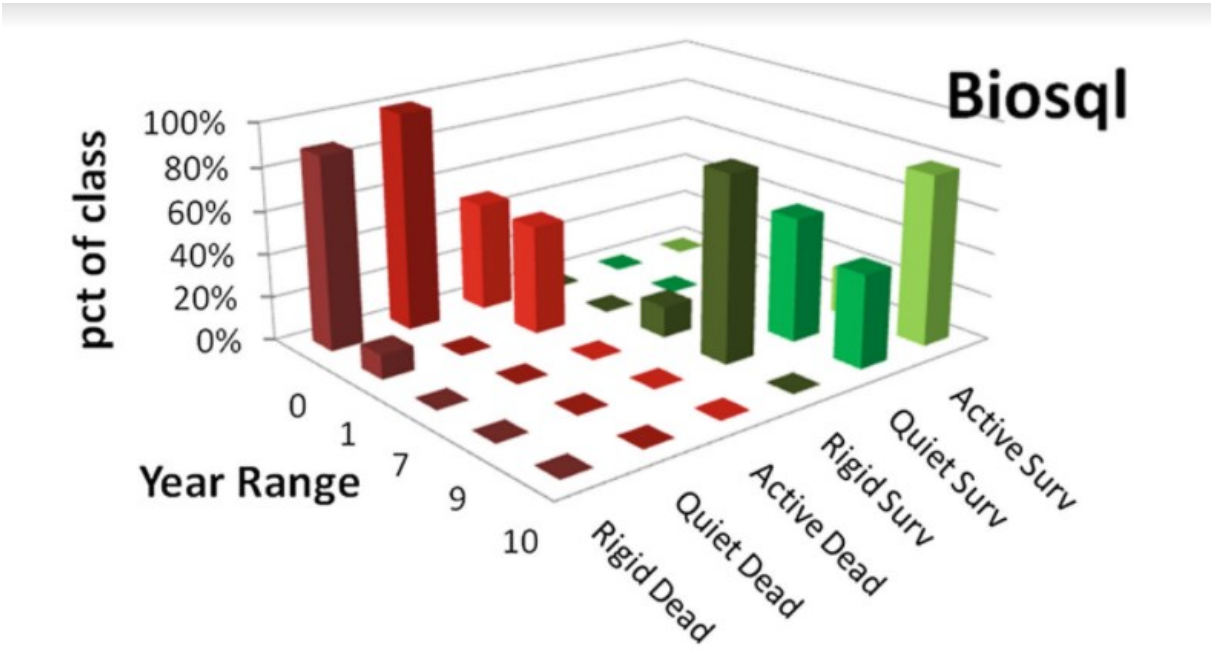

Figure 2.2 The Electrolysis pattern: the left axis shows the durations in years; the right axis shows the level of activity of tables; the vertical axis shows the percentage of tables with respect to their activity class. [11] (figure reproduced with author permission)

2.2.10 EVO-NET

The problem of how relations among states reflect the evolution of temporal data, was researched during 2021 [12]. A state graph was used to portray the time – varying relations during various time series. In order to figure the various patterns created, a GNN based model called Evo-Net (Evolutionary State Graph Network) was used on real world databases. The framework can transform any given time series into a dynamic graph based on characteristics, such as states or weights and creates a neural network to predict correlations and various events.

2.2.11 Schema Evolution and Taxa

Finally, the most recent research happened during 2021 and it's considered as the largest empirical study on the Free Open-Source Software projects [13]. This research was conducted on 195 projects and focused on understanding the characteristics of schema evolution in order to answer 3 queries.

- The first is whether the schema evolution is a continuous process
- The second is whether there could be a categorization or patterns of schema lives.

– The final one is about the properties of schema evolution.

The findings about the first query implied that schema evolution is not a continuous activity. However, only a small number of projects are fairly active. This absence of evolution is related to the difficulty of the adjustments and maintenance required. The second finding revealed the existence of "taxa" of schemata, which characterize their activity, by monitoring their heartbeat, or in other words how active they were throughout their lifespan. These taxa are the following:

– History-less - They contain 1 commit on the initial file.
– Frozen - 0 changes and 0 active commits.
– Almost Frozen- Very few commits and little changes.
– Focused Shot & Frozen - Very few commits, focused change in a single commit.
– Moderate - Moderate rate of heartbeat (active commits), moderate volume of activity.
– Focused Shot & Low - A couple of reeds and a few active commits, focused change.
– Active - Frequent rate of heartbeat (active commits), high volume of activity.

The last finding focused on the volume, frequency and important characteristics of changes on the schemata. Volume of change is measured in affected attributes as well as table creations, alterations and deletions. The number of any kind of commits characterizes the frequency of changes. The density of change is characterized by the special kind of commits called reeds and turfs. Last but not least, the change of size of the schema on a coarse level is characterized by the tables inserted or deleted at the start and at the end of the life cycle of the schema.

2.3 Comparison to Related Work and Thesis Outline

So far, the research in the schema evolution field is still evolving. Even though there are quite a few studies on this field, there has been a number of important revelations on this field. These include patterns during evolution as well as categories depending on number of changes or taxa. Some of these studies relied on the various laws or comparative methods that are used for software evolution. This thesis is aiming to

continue from where the latest schema evolution research left off [13], while also using some of the software evolution methods that have been mentioned, using a tool to define changes and development phases [5] and model creation [8]. The goal of this thesis is to extract the various states a schema of a project enters during its development lifecycle over a substantial corpus of schema histories and represent them as a sequence of phases. Then, the final goal is to come up with a concise, informative set of common patterns of these phase sequences and visually represent them as state diagrams. To the best of our knowledge this is the first time that an attempt towards this problem is made in the field of schema evolution.

Using the findings of the research [13], we aim to extract the metrics that will help identify the various stages a schema of a project entered during its development period. The number of changes, which can be found for each month, will be classified into labeled states. These states contain Atomic Measurements and can be grouped into a phase. In this research the initial goal is to group and merge all the Atomic Measurements depending on a factor, such as similarity of label or condition, into a phase and link the neighboring phases using a transition. A transition is the bridge that connects two phases and can also be labelled by the difference the two neighboring phases create. All the Transitions, Phases and Atomic Measurements of the project make up the PhaseSeries.

Our goal is to find the silver lining between having many transitions, which gives a lot of accuracy and information, or having fewer transitions and reducing accuracy. After we gain the data for one database of a project, we aim to do the same for more projects and group all of them into categories, based on the resulting PhaseSeries and overall behavior. With this approach, we aim to extract concrete categories and their frequency of appearance on different taxa.

CHAPTER 3

PHASE EXTRACTION AND MERGING ALGORITHMS

3.1 Introduction

The goal of this thesis is to extract information from schemata of various databases, organize them into a PhaseSeries and group the PhaseSeries in order to extract various categories of schema evolution patterns. To achieve those goals, we break the process into 3 steps:

- The first step includes the ingestion of data, in order to extract the Phases and Transitions and create the initial PhaseSeries of the schema. The definitions of these terms are:
    - A Phase is a state of the schema during schema evolution, during a specific timeframe with a similar rate of change. This state can have the span of one Atomic Measurement or more.
    - An Atomic Measurement contains the (x,y) metrics. The x is the percentage of time metric and y is the percentage of total activity metric.

    - A Transition is the link that connects two neighboring phases; the Phase-From, which is the initial phase and the Phase-To which is the phase it ends in.
    - A PhaseSeries contains all the Phases and Transitions of a project.
- The second step includes the merging of the Phases and Transitions given a specific condition. In this way we aim to create a compact PhaseSeries that gives us all the information needed without losing a lot of accuracy. The merging conditions can be similarity of Phases or Transitions, or duration of a Phase in comparison with the project's duration.
- The final step includes the grouping and clustering of all similar PhaseSeries in order to extract patterns of schema evolution categories.

3.2 Fundamental Concepts and Reference Algorithm

3.2.1 Original Setup

The initial step of the Phase Extractor is the ingestion of data.

The data that is required can be found in a tsv file, which contains data of the evolution history of a database. Each file includes monthly information gathered during each project's development cycle, such as insertions, deletions, overall changes and other metrics. For this project we require:

- The number of months the evolution of the schema lasted.
- The total schema changes that occurred during this period.

So, initially, the tsv file of a specific project must be loaded into the system. Two metrics are computed from the data.

- The cumulative fraction of schema evolution timespan for every month (x) using the formula: currentMonth/(TotalNumberOfMonths-1)
- The cumulative fraction of changes for every month (y) using the formula: sumOfPrevious + (currentNumberOfChanges/SumOfChanges).

3.2.2 Fundamental Concepts

Given the above data, we use each pair of (x,y) in order to create one Phase.

- x: cumulative time progress percentage.
- y: cumulative evolution activity percentage.

A *Phase* is a sequence of contiguous Atomic Measurements that demonstrate similar evolutionary behavior of the schema, during a specific timeframe. A Phase can have the span of one Atomic Measurement or more. At the initial step, each Phase consists of one Atomic Measurement.

A *Transition* is the link from one phase to another. In other words, it is the bridge that connects a Phase of the development with the next one. A Transition includes two Phases, which are the ones it connects; the initial PhaseFrom and the next PhaseTo.

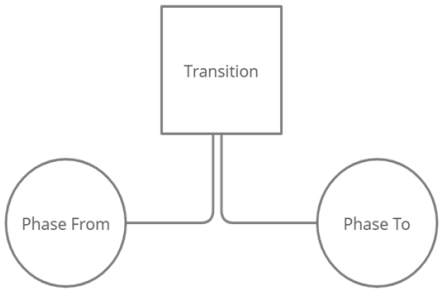

Figure 3.1 A single Transition and Phases it connects

3.2.3 Reference Algorithm

So, at first, we assume each Atomic Measurement (e.g., the amount of change per commit, or per month) to be a Phase and each pair of consecutive Phases to create a Transition. A Transition's PhaseTo will be the same phase as the PhaseFrom of the next transition. All of the Transitions and the Phases they link, are included in a list called PhaseSeries. Initially, a PhaseSeries consists of (TotalMonths-1) Transitions and (TotalMonths) Phases.

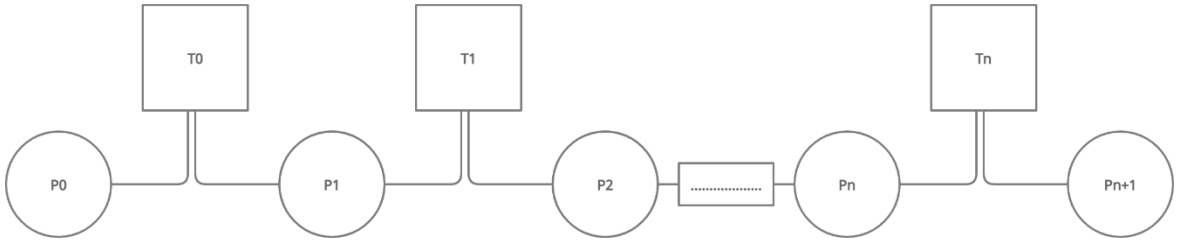

Figure 3.2 A PhaseSeries

The next step is the labeling of the Transitions. The transition from one phase to another creates a line in the (x,y) space which can be characterized by the angle it creates. The higher the degrees of an angle, the steeper the transition is. And that means that the bigger the angle, the higher the rate of change occurred during a particular transition between two phases of a project.
In order to find this angle, we go through the following steps.

- We calculate the distance of the y metric created between the phases of the transition. This can be achieved by distracting the y metric of the last Atomic Measurement of the Phase-To by the y metric of the first Atomic Measurement of Phase-From. So, dy = yLast-yFirst.
- The same has to be done for the x metric. So dx=xLast-xFirst.
- The next step is to measure the division of dy/dx and calculate the atan(dy/dx) which gives us the angle in degrees.
- Depending on the degrees the angle can be characterized as:
    - FLAT: $\varphi <= 0$ degrees - no change rate during the transition
    - LOW: $0 < \varphi <= 30$ degrees - a small change rate during the transition
    - REGULAR: $30 < \varphi <= 60$ degrees - a normal change rate during a transition
    - STEEP: $60 < \varphi$ degrees - a significant change rate during a transition

The next step is the iteration of the PhaseSeries in order to merge neighboring Transitions by following a certain condition. The conditions can be the similarity of labels, or duration of Phases and overall Project. When these conditions are met, the merging of the selected Transitions can occur. During each cycle of iterations, we are comparing a pair of neighboring Transitions, e.g., T0-T1, T1-T2, etc. If T1 can be merged, it will be merged to the T0, so the next pair to be examined in the iteration will be T0-T2. In more detail, merging can be achieved by appending the Phase-To of the second Transition to the Phase-To of the preceding Transition. That includes all of the Atomic Measurements of the Phase that is being appended. In other words, we add all of the Atomic Measurements of the Phase-To of the second Phase in the list of Atomic Measurements of the PhaseTo of the first Phase. In this way, the number of phases and transitions is reduced. After this process is done, it is crucial to recalculate the label of the merged Transition.

While the merging process is in action, more and more Phases will be merged together. That means that Phases will contain more than one Atomic Measurements. It is crucial to identify the internal angles that are created during this process. The angles can be characterized by labels similarly as the Transitions' labels. To find the internal angle of a Phase we follow similar same steps as finding the Transition's labels:

- We calculate the distance of the y metric created between the Atomic Measurements of the Phase. This can be achieved by distracting the y metric of the last Atomic Measurement by the y metric of the first Atomic Measurement. So, dy = yLast-yFirst.
- The same has to be done for the x metric. So dx=xLast-xFirst.
- The next step is to measure the division of dy/dx and calculate the atan (dy/dx) which gives us the angle in degrees.
- Depending on the degrees the angle can be characterized as:
    - FLAT: $\varphi <= 0$ degrees - no changes during the Phase
    - LOW: $0 < \varphi <= 30$ degrees - a low rate of changes during the Phase
    - REGULAR: $30 < \varphi <= 60$ degrees-a normal rate of change during a Phase
    - STEEP: $60 < \varphi$ degrees - a significant change rate during a Phase

At the beginning of the iterations the Phases will contain one Atomic Measurement, so they will be classified as MONADIC POINT (M).

Once the initial PhaseSeries model is ready, we can proceed with the merging algorithms. A general algorithm can have the following form:

- Input: The PhaseSeries which is a transition list L = {T0,...,Tn}. Every Transition contains two single Atomic Measurement Phases, Phase.From and Phase.To. Every Transition contains a label.
- Output: A new PhaseSeries which is the new Transition list L' where consecutive transitions from L have been merged according to a specific condition.
- Variables:
    - L: A list that contains all of the transitions.
    - L': A list that contains all of the new transitions.
    - firstTransition: First transition of the list L.
    - secondTransition: Second transition of the list L.
    - counter: Counts the number of iterations.

Algorithm 3.1 General Merging Algorithm.

```
Require: L,L.size>1
Ensure: L'
1:  transitionList L; transitionList L';
2:  firstTransition = L.get(0); secondTransition = L.get(1);
3:  counter = 1; //as long as we have 2 or more transitions
4:  while (counter<L.size-1){
5:    if (canMergeByCondition(firstTransition, secondTransition) ==
    TRUE){
6:      firstTransition.PhaseTo.append(secondTransition.PhaseTo)
7:    }else{
8:      L'.add(firstTransition);
9:      firstTransition = secondTransition;
10:   }
11:   counter++;
12:   secondTransition = L.get(counter);
13: }
14: return L';
```

Algorithm 3.2 Updated Merging Algorithm.

```
Require: L,L.size>1
Ensure: L'
1:  transitionList L; transitionList L';
2:  L.add(createMockTransitionFunction(L.get(0)));
3:  firstTransition = L.get(0); secondTransition = L.get(1);
4:  counter = 1; //as long as we have 2 or more transitions
5:  while (counter<L.size-1){
6:    if (canMergeByCondition(firstTransition, secondTransition) ==
    TRUE){
7:      firstTransition.PhaseTo.append(secondTransition.PhaseTo)
8:    }else{
9:      L'.add(firstTransition);
10:     firstTransition = secondTransition;
11:   }
12:   counter++;
13:   secondTransition = L.get(counter);
14: }
15: L'.get(1).PhaseFrom=L'.get(0).PhaseTo //set PhaseTo of mockTransition
    to the PhaseFrom of the first "real" transition
16: L'.remove(0); //remove mockTransition
17: return L';
```

A subtle issue with this algorithm is that the very first Transition is never examined, so it never gets merged. This can be left as is, if desirable. If not, the solution to that can be achieved by adding a "mock Transition" at the beginning of the initial PhaseSeries, which will contain the very first Phase as PhaseFrom and PhaseTo. In this way we are able to keep merging the PhaseTo, once the condition allows it. Once the merging process is complete, the Phase.To of the "mock Transition" will become the Phase.From of the first actual Transition, which is going to be the second Transition in the PhaseSeries. Once this action is done, we remove the mock transition and get the new PhaseSeries as a result. Algorithm 3.2 details this process.

Algorithm 3.3 CreateMockTransitionFunction(Transition firstTransition)

Require: firstTransition
Ensure: Transition T' //mockTransition
1: Transition T';
2: T'.PhaseFrom = firstTransition.PhaseFrom;
3: T'.PhaseTo = firstTransition.PhaseFrom;
4: **return** T';

3.3 The Merge Same Labels Algorithms

3.3.1 The Naive Merge Same Labels Algorithm

Once the PhaseSeries is labeled, it is time to merge the transitions that have the same labels. In this algorithm we simply iterate the PhaseSeries and compare the labels of each pair of transitions. The neighboring transitions that have the same labels will be merged into one.

The method explained can be accomplished by following the next steps.

- We begin the iteration by examining the first two transitions.
- We compare the labels
- If the two consecutive Transitions have the same label, they can be merged.
    - We keep the new transition and compare it to the next one in the list, using the same method
- If the two consecutive Transitions do not have the same labels, do not merge.
    - Keep the second transition and compare it to the next one.
    - The very first transition remains unchanged.

That was the first algorithm of phase merging and can be used as the initial step of other algorithms.

Algorithm 3.4 Naive Same Labels Algorithm.

```
Require: L,L.size>1
Ensure: L'
1:  transitionList L; transitionList L';
2:  L.add(createMockTransitionFunction(L.get(0)));
3:  firstTransition = L.get(0); secondTransition = L.get(1);
4:  counter = 1; //as long as we have 2 or more transitions
5:  while (counter<L.size-1){
6:    if (firstTransition.Label == secondTransition.Label){
7:      firstTransition.PhaseTo.append(secondTransition.PhaseTo)
8:    }else{
9:      L'.add(firstTransition);
10:     firstTransition = secondTransition;
11:   }
12:   counter++;
13:   secondTransition = L.get(counter);
14: }
15: L'.get(1).PhaseFrom=L'.get(0).PhaseTo //set PhaseTo of mockTransition
    to the PhaseFrom of the first "real" transition
16: L'.remove(0); //remove mockTransition
17: return L';
```

3.3.2 Post Processing for the Merge Same Labels Algorithm

Algorithm 3.5 Merge Same Labels Algorithm.

```
Require: output L from Same Label Algorithm, L.size>1
Ensure: L'
1:  transitionList L; transitionList L';
2:  L.add(createMockTransitionFunction(L.get(0)));
3:  firstTransition = L.get(0); secondTransition = L.get(1);
4:  counter = 1; //as long as we have 2 or more transitions
5:  while (counter<L.size-1){
6:    if (secondTransition.Label == FLAT){
7:      firstTransition.PhaseTo.append(secondTransition.PhaseTo)
8:    }else{
9:      L'.add(firstTransition);
10:     firstTransition = secondTransition;
11:   }
12:   counter++;
13:   secondTransition = L.get(counter);
14: }
15: L'.get(1).PhaseFrom=L'.get(0).PhaseTo //set PhaseTo of mockTransition
    to the PhaseFrom of the first "real" transition
16: L'.remove(0); //remove mockTransition
17: return L';
```

One issue that occurred with the Naive Merge Same Labels Algorithm was that it produced several Transitions that were Flats. There is no reason to keep Phases connected by flat Transitions separately, as they portray a "dead" state of the evolution. The algorithm that has been analyzed in the previous chapter can be updated so that all of the Flat Transitions will be eliminated from the PhaseSeries. So, during the iteration, if the neighboring Transition is Flat, will be merged with the previous one. The Merge Same Labels (MSL) Algorithm complements the Naive MSL by eliminating Flat Transitions.

3.3.3 Merge All but Steep Algorithm

Algorithm 3.6 Merge All but Steep Algorithm.

```
Require: output L from Post-Processed Same Label Algorithm, L.size>1
Ensure: L'
1:  transitionList L; transitionList L';
2:  L.add(createMockTransitionFunction(L.get(0)));
3:  firstTransition = L.get(0); secondTransition = L.get(1);
4:  counter = 1; //as long as we have 2 or more transitions
5:  while (counter<L.size-1){
6:    if (secondTransition.Label != STEEP){
7:      firstTransition.PhaseTo.append(secondTransition.PhaseTo)
8:    }else{
9:      L'.add(firstTransition);
10:     firstTransition = secondTransition;
11:   }
12:   counter++;
13:   secondTransition = L.get(counter);
14: }
15: L'.get(1).PhaseFrom=L'.get(0).PhaseTo //set PhaseTo of mockTransition
    to the PhaseFrom of the first "real" transition
16: L'.remove(0); //remove mockTransition
17: return L';
```

Algorithm MSL is the reference algorithm to provide a sequence of cohesive homogenous Phases for the evolution of schema. However, it occasionally produces large sequences with many Phases. To shrink the number of individual Phases we need to merge Phases further. To this end, Algorithm Merge All But Steep (MABS) is introduced.

In this algorithm the condition is that any Transition that is not Steep will be merged with the previous one. Additionally, to that, the phases themselves must not be steep either. This basically applies to phases that have more than one time points. Their

internal angle is calculated and if the outcome is over 60 degrees, then the phase itself can be labeled as steep.

- The initial step of the algorithm is to use the post processed same-label-merge method in order to reduce the number of transitions that will be evaluated.
- After this process, a method similar to the previous algorithm is used. The PhaseSeries is iterated and the neighboring transitions are being evaluated.
    - In this case if the next transition is FLAT, LOW or REGULAR, it gets merged with the previous transition. Since the phase-to of the previous transition and the phase-from of the next transition is the same, we append the phase-to of the next transition to the phase-to of the previous transition.
    - If any of the conditions mentioned above are not met, then the iteration moves to the next pair of transitions.
    - If the conditions are met, then the previous transition (with the merged phase) is being compared with the next transition.
- This process is followed until the last transition.

3.3.4 Examples

The outcome of the algorithms can also be visible through an example.
The examples that will be used are derived from the evaluation of TheWhiteTulip project.

Original Extraction

The White Tulip Project lasted 10 months and had 34 changes in total as shown at the Figure 3.3.

Month	TotalAttributeActivity
0	7
1	0
2	3
3	9
4	0
5	0
6	14
7	0
8	0
9	1

Figure 3.3 White Tulip: Initial data

Before any merging we need to find the x-y metrics.

- x: cumulative time progress percentage.
- y: cumulative evolution activity percentage.

The metric x is calculated by dividing the current month by the final month. For example, to find the progress percentage of month 6, we divide 6/9 and the outcome is 0.67.

The y metric is calculated by dividing the totalAttributeActivity by the total number of changes and adding the previous percentage, if it exists. For example the project percentage of month 6 is calculated by dividing 14/34 and adding the project percentage of month 5. In the case of month zero, there is no percentage for a previous month so we do not add anything. Results are shown in the Figure 3.4.

INITIAL DATA			
Month	TotalAttributeActivity	x	y
0	7	0.00	0.21
1	0	0.11	0.21
2	3	0.22	0.29
3	9	0.33	0.56
4	0	0.44	0.56
5	0	0.56	0.56
6	14	0.67	0.97
7	0	0.78	0.97
8	0	0.89	0.97
9	1	1.00	1.00
Totals	10 months	34 changes	

Figure 3.4 White Tulip: Initial metrics

From the data above we can conclude that there are 9 transitions and 10 phases initially. After this step we have enough data to calculate the degrees and therefore decide the labels of the PhaseSeries. For this calculation we compute dy/dx initially, which is (yFinal-yFirst)/(xFinal-xFirst). After that, we calculate the atan of the product in degrees and decide the label. The results are shown in figures 3.5 and 3.6.

BEFORE ANY MERGE					
Transitions	Phase From	Phase To	dy/dx	atan	LABEL
1	B3	B4	0.00	0.00	Flat
2	B4	B5	0.79	38.45	Regular
3	B5	B6	2.38	67.23	Steep
4	B6	B7	0.00	0.00	Flat
5	B7	B8	0.00	0.00	Flat
6	B8	B9	3.71	74.90	Steep
7	B9	B10	0.00	0.00	Flat
8	B10	B11	0.00	0.00	Flat
9	B11	B12	0.26	14.83	Low
Totals	9 transitions	10 phases			

Figure 3.5 White Tulip: Transitions, Phases and Labels

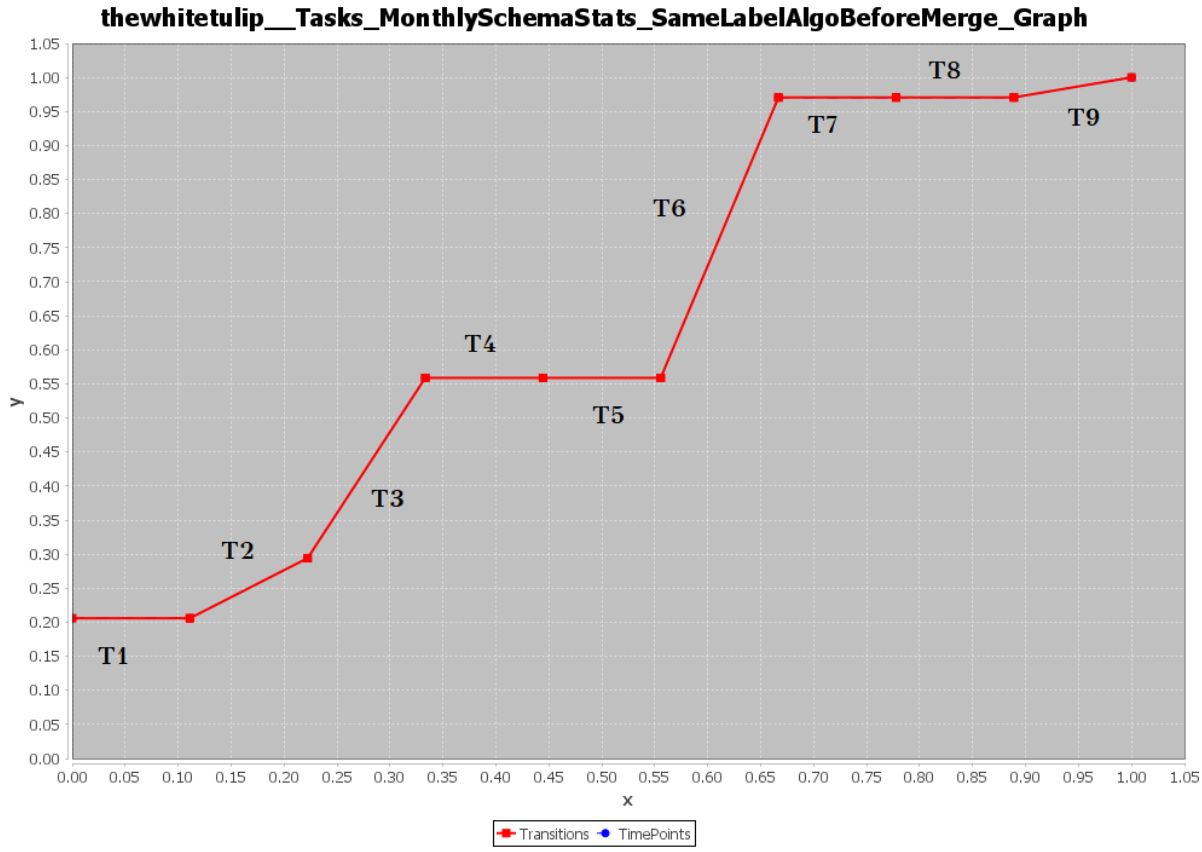

Figure 3.6 White Tulip: Initial x-y axis representation

Each red continuous solid segment indicates a transition. Each group of continuous blue dashed segments indicate a phase and the points inside the phase indicate the group of time points.

We can now proceed with the algorithms.

Same Label Merge

The very first algorithm that must be implemented merges all the transitions with the same labels.

- First, we add the T0 mock transition which by default is Flat. It's Phase-From is B3 and its Phase-To is also B3
- We compare T0 (Flat) with T1 (Flat) and realize that they can be merged. The new T01 transition's Phase-From will consist of the Atomic Measurement B3 and the Atomic Measurements B4-B5 will consist of Phase-To. The new transition's label is Flat. That is because we calculate by taking the last Atomic Measurement of Phase-From(B3) and the first one of Phase-To(B4), which results in Flat.
- Continuing from T01 (Flat), we compare with T2 (Regular). Their labels are not the same, so we compare the next pair.
- T2's (Regular) and T3's (Steep) labels are also different so they cannot be merged.
- The same applies to T3 (Steep) and T4 (Flat).
- However, T4 (Flat) and T5 (Flat) have the same labels. So they will be merged. The new T45 transition's Phase-From will consist of the Atomic Measurement B6 and the Atomic Measurements B7-B8 will consist of Phase-To. The new transition's label is Flat. That is because we calculate by taking the last Atomic Measurement of Phase-From(B6) and the first one of Phase-To(B7), which results in Flat, just like transition T4.
- Comparing the new T45(Flat) to the next one T6(Steep) we notice that they do not have the same labels, therefore cannot be merged
- T6(Steep) and T7(Flat) cannot be merged either.
- T7(Flat) and T8(Flat) can be merged. The new transition's Phase-From will include the Atomic Measurement B9 and Phase-To will include B10 and B11 Atomic Measurements. The new transition's label is Flat. That is because we calculate by taking the last Atomic Measurement of Phase-From(B9) and the first one of Phase-To(B10), which results in Flat, just like transition T9.
- The new transition (Flat) cannot be merged with T9(Low).

– The final step is the removal of the "mock Transition". This is contained in T01. Its neighboring transition is T2, so the new Phase-From of T2 will be T01's Phase-To (B3-B4) and T01 will be eliminated.
– Detailed results are shown in figures 3.7 and 3.8

SAME LABEL MERGE					
Transitions	Phase From	Phase To	dy/dx	atan	LABEL
1	B3-B4	B5	0.79	38.45	Regular
2	B5	B6	2.38	67.23	Steep
3	B6	B7-B8	0.00	0.00	Flat
4	B8	B9	3.71	74.90	Steep
5	B9	B10-B11	0.00	0.00	Flat
6	B11	B12	0.26	14.83	Low
Totals	6 transitions	7 phases			

Figure 3.7 White Tulip: Same Label detailed results

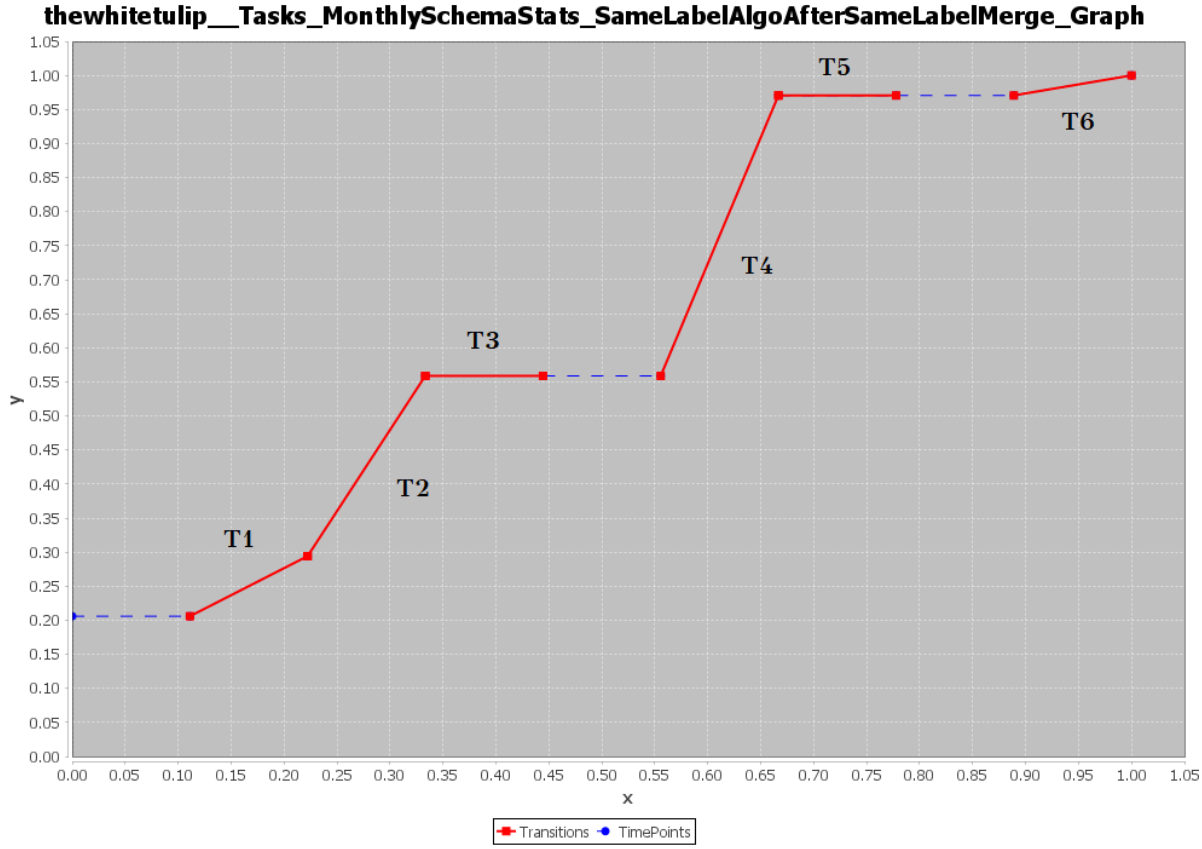

Figure 3.8 White Tulip: Same Label output x-y axis representation

Post-Processed Same Label Merge

Now the goal is to merge all the transitions that are flat.

- First, we add the T0 mock transition which by default is Flat. It's Phase-From is B3-B4 and it is Phase-To is also B3-B4
- We compare T0 (Flat) with T1 (Regular) and realize that they cannot be merged.
- Comparing T1 (Regular) and T2 (Steep), we realize T2 is not Flat so they cannot be merged.
- The T2 (Steep) and T3 (Flat) can be merged. So, the new Transition will be T23(Steep), where the Phase-To will consist of B6-B7-B8 Atomic Measurements.
- T23(Steep) cannot be merged with T4(Steep).
- T4 (Steep) and T5 (Flat) can be merged. So, the new Transition will be T45(Steep), where the Phase-To will consist of B9-B10-B11 Atomic Measurements.
- T45(Steep) cannot be merged with T6(Low).
- The final step is the removal of the "mock Transition" T0. It's neighboring transition is T1, so the new Phase-From of T1 will be T0's Phase-To (B3-B4) and T0 will be eliminated.
- Detailed results are shown in figures 3.9 and 3.10.

POST PROCESSED SAME LABEL MERGE					
Transitions	Phase From	Phase To	dy/dx	atan	LABEL
1	B3-B4	B5	0.79	38.45	Regular
2	B5	B6-B7-B8	2.38	67.23	Steep
3	B8	B9-B10-B11	3.71	74.90	Steep
4	B11	B12	0.26	14.83	Low
Totals	4 transitions	5 phases			

Figure 3.9 White Tulip: Post Processed Same Label detailed results

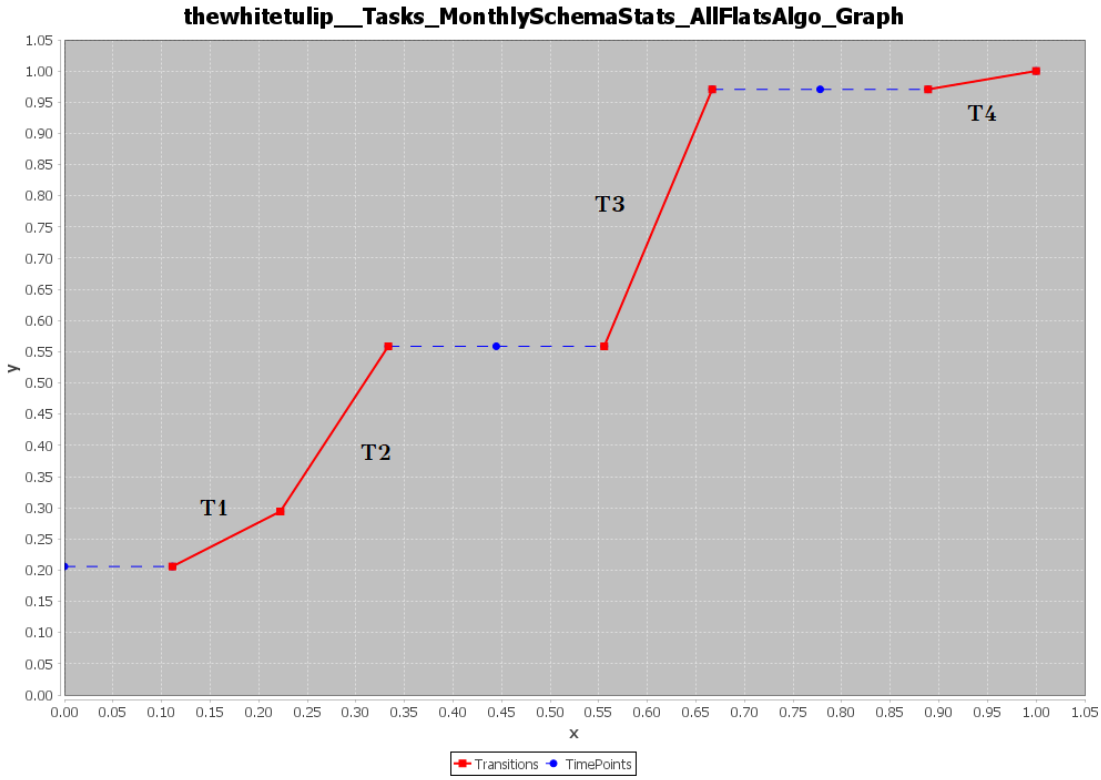

Figure 3.10 White Tulip: Post Processed Same Label output x-y axis representation

Merge All But Steep

Now the goal is to merge all the transitions that are not steep.

- First, we add the T0 mock transition which by default is Flat. It's Phase-From is B3-B4 and it's Phase-To is also B3-B4.
- We compare it with T1 (Regular) and notice that it can be merged. The new T01 transition's Phase-From will consist of the Atomic Measurements B3-B4 and the Atomic Measurements B3-B4-B5 will consist the Phase-To. The new transition's label is Flat. That is because we calculate by taking the last Atomic Measurement of Phase-From(B4) and the first one of Phase-To(B3), which results in Flat.
- The next transition T2 is Steep, so it cannot be merged.
- T3 is also Steep, so it cannot be merged either.
- T4 is Low so it can be merged with T3. The new T34 transition's PhaseFrom contains B8 and PhaseTo contains B9-B10-B11-B12.
- Now it's time to remove the mockTransition. T01's PhaseFrom will become T2's PhaseFrom, which will include the Atomic Measurements B3-B4-B5.

– Detailed Results are shown in figures 3.11 and 3.12.

NOT STEEP LABEL MERGE					
Transitions	Phase From	Phase To	dy/dx	atan	LABEL
1	B3-B4-B5	B6-B7-B8	2.38	67.23	Steep
2	B8	B9-B10-B11-B12	3.71	74.90	Steep
Totals	2 transitions	3 phases			

Figure 3.11 White Tulip: Merge All but Steep details

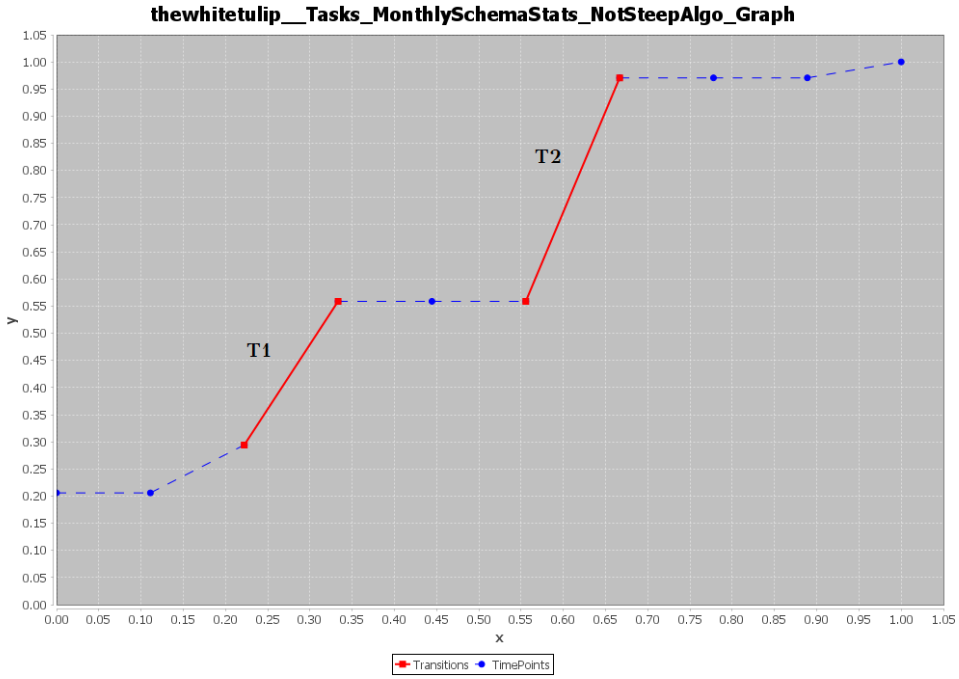

Figure 3.12 White Tulip: Merge All but Steep x-y axis representation

3.4 Signatures

We need to be able to represent a PhaseSeries, in order to further process it, as well as be able understand it, without losing information. A concise and intuitive way to achieve this is the representation of PhaseSeries via *signatures*. A signature is a series of symbols that inform us of all the Phases and Transitions between them and they can contain the labels and duration of each component. This representation is a compact but detailed way to understand the resulting series after a merging process. So, focusing on "The white tulip" project's examples we can derive data for each algorithm. The detailed signatures for each algorithm, appear in Figure 3.13.

ALGORITHM	SIGNATURE
MERGE SAME LABEL	P(F:2)-T(R)-P(M:1)-T(S)-P(M:1)-T(F)-P(F:2)-T(S)-P(M:1)-T(F)-P(F:2)-T(L)-P(M:1)
MERGE SAME LABEL+	P(F:2)-T(R)-P(M:1)-T(S)-P(F:3)-T(S)-P(F:3)-T(L)-P(M:1)
MERGE ALL BUT STEEP	P(L:3)-T(S)-P(F:3)-T(S)-P(L:4)

Figure 3.13 White Tulip: Fully Detailed Signatures

For the naive Same Label Merge Algorithm, in the first line of Figure 3.13, observe the P(F:2). This means that the initial Phase consists of 2 Atomic Measurements and the Phase's Label is Flat. Similarly, the T(R) shows the Transition which connects neighboring Phases (Transitions do not have durations). In this case it is labeled as Regular. The next Phase, P(M), indicates a Phase with a single point, which means that this point is between two transitions.

In the case of the post-processing of Merge Same Label where Flat phases are removed from the signature, we observe that the signature has become a lot more compact. That is because all single points and Flat Transitions have been merged and eliminated, as shown in the 3.3.2 chapter.

In the case of the Merge All But Steep Algorithm, observe that the signature has become significantly shorter, at the price of accuracy. In this case, all of the Transitions that were labelled as Flat, Low and Regular have been merged, leaving only Steep transitions behind.

The signature representation is very helpful to understand the state of a project, after a merging process and it can become a lot more compact to help us understand the Phases and Transitions if we remove the details about the duration of Phases. This is shown in the figure 3.14. The reason for reducing the amount of information in the signature is that this compression facilitates the grouping of different projects into groups with identical, or at least significantly similar, signatures.

ALGORITHM	SIGNATURE
MERGE SAME LABEL	P(F)-T(R)-P(M)-T(S)-P(M)-T(F)-P(F)-T(S)-P(M)-T(F)-P(F)-T(L)-P(M)
MERGE SAME LABEL+	P(F)-T(R)-P(M)-T(S)-P(F)-T(S)-P(F)-T(L)-P(M)
MERGE ALL BUT STEEP	P(L)-T(S)-P(F)-T(S)-P(L)

Figure 3.14 White Tulip: Signatures without durations

Finally, we can represent signatures without the Transition levels. This can allow us to derive the structure of the Phases and Transitions a project entered during its evolution, through the various merging algorithm variants. This facilitates the identification of similar signatures even further. The new signatures for our running example are shown in Figure 3.15.

ALGORITHM	SIGNATURE
MERGE SAME LABEL	P(F)-T()-P(M)-T()-P(M)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(M)
MERGE SAME LABEL+	P(F)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(M)
MERGE ALL BUT STEEP	P(L)-T()-P(F)-T()-P(L)

Figure 3.15 White Tulip: Signatures without Transition Labels

CHAPTER 4

EXPERIMENTS

4.1 Experimental Setup

Several experiments were conducted in order to test the effectiveness and efficiency of the algorithms that were analyzed in the previous chapters. We tested on the 195 databases of ICDE 2021 [13]. These datasets are part of a collection that facilitate the study of Schema Evolution. Each dataset refers to the history of a database schema as a sequence of releases. The collection of this data has been compiled and processed by Panos Vassiliadis, in May 2019.

The algorithms that were tested were "The Naive Merge Same Labels Algorithm", "Merge Same Labels Algorithm" and "Merge All But Steep". More specifically, on the first algorithm, the method that iterates, merges and returns the new PhaseSeries has been timed and tested, and regarding the second algorithm we timed the post-processing part, which reiterates and merges all flat Transitions. So, to test the full functionality of the "Same Labels Merge" Algorithm, we summed both of the aforementioned algorithms' findings. The second algorithm that was tested was the

"Merge All but Steep" Algorithm. Similarly, the timing results of this algorithm must be added to the previous results so that we can have an idea of the full picture.
All the experiments were run 5 times on the Eclipse IDE platform, in order to find average timings. The tests were conducted on a laptop with the specifications as shown in the tables 4.1 and 4.2:

Device ID	IdeaPad Gaming 3 15ACH6
Processor	AMD Ryzen 7 5800H with Radeon Graphics
Installed RAM	16 GB (13.9 GB usable)
System type	64-bit operating system, x64-based processor

Table 4.1 Computer Hardware Specifications

OS Edition	Windows 11 Home
Version	21H2
Installed on	12/20/2021
OS build	22000.739

Table 4.2 Computer Software Specifications

The results of the experiments for each algorithm include:

- The amount of the effectiveness of each algorithm in terms of the frequent phase sequences it produces.
- The results of execution time that make up the efficiency for each algorithm.
- The results of execution time per number of merges for each algorithm.
- Examples of resulting state diagrams.

4.2 Effectiveness Assessment

4.2.1 Algorithm Effectiveness

In this Section, we discuss the effectiveness of the proposed algorithms with respect to the discovery of common patterns in the lives of database schemata.

How do we measure the effectiveness of our algorithms? For each schema history of our corpus, we execute each algorithm, and, in each case, the result is a Phase Series, i.e., a summary of the schema's life in phases. Each phase has a duration and label representing its angle in the cumulative schema evolution chart. Each transition is also characterized by its angle-related label.

We can produce various signatures for each Phase Series as a text string. For example, we can have the following degrees of detail in the signature: (a) both label and duration for phases, label for transitions, (b) only labels for both phases and transitions, (c) only phase labels. We will employ the following terminology:

- P(.) denotes phases and T(.) denotes transitions
- Labels: M=single point, F=Flat, L=Low, R=Regular, S=Steep
- P(x:d) denotes a phase with a label of x and a duration of d

For example, for the whiteTulip project, we can have the following signatures, with increasing level of conciseness as shown in the table 4.3:

Conciseness	Algo	Signature
Full	SLM	P(F:2)-T(R)-P(M:1)-T(S)-P(M:1)-T(F)-P(F:2)-T(S)-P(M:1)-T(F)-P(F:2)-T(L)-P(M:1)
	SLM+	P(F:2)-T(R)-P(M:1)-T(S)-P(F:3)-T(S)-P(F:3)-T(L)-P(M:1)
	MABS	P(L:3)-T(S)-P(F:3)-T(S)-P(L:4)
Labels-only	SLM	P(F)-T(R)-P(M)-T(S)-P(M)-T(F)-P(F)-T(S)-P(M)-T(F)-P(F)-T(L)-P(M)
	SLM+	P(F)-T(R)-P(M)-T(S)-P(F)-T(S)-P(F)-T(L)-P(M)
	MABS	P(L)-T(S)-P(F)-T(S)-P(L)
Labels-phases	SLM	P(F)-T()-P(M)-T()-P(M)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(M)
	SLM+	P(F)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(M)
	MABS	P(L)-T()-P(F)-T()-P(L)

Table 4.3 Signatures for white tulip

The goal of our algorithms, and hence, the method to evaluate their effectiveness, is to produce signatures that are as few and accurate, as possible.

4.2.2 Same Label Merge variants

The algorithm Same Label Merge (SLM) and its extension that merges flat transitions into their preceding phases, which we call SLM+ for short, takes the most detailed

approach to signature production. We will restrict discussion to SLM+ as it is a straightforward extension of SLM.

Effectiveness

MSL+ produces (a) 120 signatures with full annotation, (b) 74 when duration is removed and (c) 64 signatures with labels only for the phases. Once we sort them for popularity in the corpus, the first 15 of them cover 76% of the corpus and include all the signatures with 0 and 1 transitions (See Table 4.4).

Signature	Count of projects	#transitions
P(M)	39	0
P(F)-T()-P(M)	33	1
P(F)	24	0
P(M)-T()-P(M)	15	1
P(F)-T()-P(F)	7	1
P(F)-T()-P(F)-T()-P(M)	6	2
P(M)-T()-P(F)	4	1
P(M)-T()-P(L)	4	1
P(M)-T()-P(F)-T()-P(M)	4	2
P(F)-T()-P(R)	2	1
P(M)-T()-P(M)-T()-P(M)	2	2
P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)	2	5
P(M)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(M)	2	5
P(F)-T()-P(L)	1	1
P(F)-T()-P(S)	1	1
P(F)-T()-P(M)-T()-P(M)	1	2
P(F)-T()-P(M)-T()-P(F)	1	2
P(M)-T()-P(F)-T()-P(R)	1	2
P(M)-T()-P(L)-T()-P(M)	1	2
P(F)-T()-P(F)-T()-P(F)	1	2
P(M)-T()-P(F)-T()-P(F)	1	2
P(F)-T()-P(F)-T()-P(F)-T()-P(F)	1	3
P(M)-T()-P(M)-T()-P(F)-T()-P(F)	1	3
P(M)-T()-P(F)-T()-P(M)-T()-P(M)	1	3
P(F)-T()-P(F)-T()-P(F)-T()-P(L)	1	3
P(M)-T()-P(M)-T()-P(M)-T()-P(L)	1	3
P(M)-T()-P(F)-T()-P(L)-T()-P(M)	1	3
P(F)-T()-P(L)-T()-P(M)-T()-P(L)	1	3

P(M)-T()-P(M)-T()-P(F)-T()-P(L)	1	3
P(M)-T()-P(S)-T()-P(F)-T()-P(M)	1	3
P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)	1	4
P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)	1	4
P(M)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(M)	1	4
P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)	1	4
P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(R)	1	4
P(F)-T()-P(F)-T()-P(L)-T()-P(F)-T()-P(M)	1	4
P(M)-T()-P(F)-T()-P(L)-T()-P(M)-T()-P(M)	1	4
P(M)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(L)	1	4
P(F)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(M)	1	4
P(M)-T()-P(M)-T()-P(M)-T()-P(F)-T()-P(R)	1	4
P(F)-T()-P(F)-T()-P(F)-T()-P(R)-T()-P(L)-T()-P(M)	1	5
P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)	1	5
P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(L)	1	5
F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)	1	6
P(M)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(L)-T()-P(M)	1	6
P(F)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(M)	1	6
P(M)-T()-P(R)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(R)-T()-P(L)-T()-P(L)	1	7
P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(F)	1	7
P(F)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(L)-T()-P(L)-T()-P(M)-T()-P(F)-T()-P(M)	1	8
P(M)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(S)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)	1	8
F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(S)-T()-P(R)-T()-P(M)	1	8
P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(R)	1	8
P(F)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(L)-T()-P(L)-T()-P(R)-T()-P(F)-T()-P(F)-T()-P(M)	1	10
P(F)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(L)-T()-P(M)	1	10
P(M)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(R)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(M)	1	10

M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)	1	12
P(M)-T()-P(M)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(L)-T()-P(M)-T()-P(F)-T()-P(L)-T()-P(M)	1	12
P(F)-T()-P(F)-T()-P(M)-T()-P(S)-T()-P(M)-T()-P(M)-T()-P(R)-T()-P(M)-T()-P(L)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)	1	14
P(M)-T()-P(M)-T()-P(M)-T()-P(F)-T()-P(R)-T()-P(L)-T()-P(M)-T()-P(L)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)	1	14
P(M)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(L)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)	1	15
P(F)-T()-P(L)-T()-P(R)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(L)-T()-P(L)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(R)-T()-P(S)-T()-P(F)-T()-P(M)-T()-P(L)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(M)	1	19
P(M)-T()-P(M)-T()-P(L)-T()-P(F)-T()-P(M)-T()-P(L)-T()-P(F)-T()-P(L)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(S)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(F)	1	19
P(M)-T()-P(M)-T()-P(L)-T()-P(S)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(L)-T()-P(M)-T()-P(R)-T()-P(F)-T()-P(L)-T()-P(M)-T()-P(R)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(L)-T()-P(M)-T()-P(M)-T()-P(L)-T()-P(F)-T()-P(L)-T()-P(L)-T()-P(R)-T()-P(L)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(L)-T()-P(M)-T()-P(M)	1	39
P(M)-T()-P(M)-T()-P(R)-T()-P(M)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(F)-T()-P(M)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(S)-T()-P(M)-T()-P(M)-T()-P(L)-T()-P(L)-T()-P(F)-T()-P(L)-T()-P(F)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(R)-T()-P(S)-T()-P(L)-T()-P(F)-T()-P(F)-T()-P(M)-T()-P(M)-T()-P(M)-T()-P(S)-T()-P(M)	1	39

Table 4.4 Phase-Label-Only Signatures of Phase Series, their frequency, and their transitions for MSL+.

Taxa and signatures

Taxa behave differently with respect to the signatures -- in particular, differently with respect to the number of transitions that signatures contain. Observe Figure 4.1, clearly depicting the different groups of taxa with respect to their signatures.

Taxon / #transitions	0	1	2	>=3	Grand Total
0_FROZEN	34				**34**
1_ALMOST_FROZEN	21	39	5		**65**
1_FocusedShot_n_FROZEN	6	15	4		**25**
2_MODERATE	1	7	5	16	**29**
3_FocusedShot_n_LOW	1	6	1	12	**20**
4_ACTIVE			3	19	**22**
Grand Total	**63**	**67**	**18**	**47**	**195**

Taxon / #transitions	0	1	2	>=3	Grand Total
0_FROZEN	**100%**				**100%**
1_ALMOST_FROZEN	32%	**60%**	8%		**100%**
1_FocusedShot_n_FROZEN	24%	**60%**	16%		**100%**
2_MODERATE	3%	24%	17%	**55%**	**100%**
3_FocusedShot_n_LOW	5%	30%	5%	**60%**	**100%**
4_ACTIVE			14%	**86%**	**100%**
Grand Total	**32%**	**34%**	**9%**	**24%**	**100%**

Figure 4.1 Transition ranges and taxa for MSL+

The first group contains the family of frozen taxa. Frozen projects, by definition, contain no transitions. Almost frozen have mostly one transition and frequently zero. Practically, this means that their change does not spread in more than one month in most cases. The same applies with Focused Shot and Frozen.

The second group contains the family of moderately active taxa. Moderate and Focused Shot and Low are two taxa with similar amount of change, albeit difference in how clustered the change is in different commits. In terms of monthly change, however, they prove to be quite similar, mostly having more than 2 transitions. However, the moderate taxon includes several projects with longer phase series and number of transitions (see Figure 4.2).

Finally, the third group includes the Active projects, which typically have longer lives and several phases.

Taxon / #transitions	0	1	2	3	4	5	6	7	8	10	12	14	15	19	39	Grand
0_FROZEN	34															**34**
1_ALMOST_FROZEN	21	39	5													**65**
1_FocusedShot_n_FROZEN	6	15	4													**25**
2_MODERATE	1	7	5	4	4	2	1	1	1	1	1		1			**29**
3_FocusedShot_n_LOW	1	6	1	4	3	4	1									**20**
4_ACTIVE			3	1	3	1	1	1	3	2	1	2		2	2	**22**
Grand Total	**63**	**67**	**18**	**9**	**10**	**7**	**3**	**2**	**4**	**3**	**2**	**2**	**1**	**2**	**2**	**195**

Figure 4.2 Breakdown of projects in taxa and number of transitions for MSL+

4.2.3 Merge All But Steep

In this subsection, we will examine the Merge All but Steep Algorithm (MABS) effectiveness.

Effectiveness

MABS produces (a) 101 signatures with full annotation, (b) 44 when duration is removed and (c) 44 signatures with labels only for the phases. This is because all the remaining Transitions will be steep, so they'll have one label variant anyway. Once we sort them for popularity in the corpus, the first 15 of them cover 85.12% of the corpus and include all the signatures with 0 and 1 transitions (See Table 4.5).

Signature	# of Projects	#transitions
P(L)	61	0
P(M)	39	0
P(F)	24	0
P(F)-T()-P(M)	12	1
P(L)-T()-P(L)	6	1
P(F)-T()-P(F)	4	1
P(F)-T()-P(F)-T()-P(M)	4	2
P(R)	3	0
P(M)-T()-P(L)	3	1
P(M)-T()-P(F)	2	1
P(R)-T()-P(L)	2	1
P(F)-T()-P(F)-T()-P(F)-T()-P(L)	2	3

P(F)-T()-P(R)-T()-P(L)-T()-P(M)	2	3
P(F)-T()-P(L)	1	1
P(F)-T()-P(S)	1	1
P(L)-T()-P(F)	1	1
P(L)-T()-P(M)	1	1
P(M)-T()-P(R)	1	1
P(F)-T()-P(F)-T()-P(F)	1	2
P(L)-T()-P(F)-T()-P(L)	1	2
P(L)-T()-P(L)-T()-P(L)	1	2
P(L)-T()-P(L)-T()-P(M)	1	2
P(L)-T()-P(R)-T()-P(L)	1	2
P(M)-T()-P(F)-T()-P(F)	1	2
P(M)-T()-P(F)-T()-P(M)	1	2
P(M)-T()-P(L)-T()-P(L)	1	2
P(M)-T()-P(L)-T()-P(M)	1	2
P(L)-T()-P(F)-T()-P(F)-T()-P(M)	1	3
P(L)-T()-P(F)-T()-P(L)-T()-P(M)	1	3
P(L)-T()-P(L)-T()-P(F)-T()-P(M)	1	3
P(L)-T()-P(R)-T()-P(L)-T()-P(L)	1	3
P(M)-T()-P(F)-T()-P(L)-T()-P(L)	1	3
P(M)-T()-P(R)-T()-P(L)-T()-P(L)	1	3
P(F)-T()-P(F)-T()-P(L)-T()-P(F)-T()-P(M)	1	4
P(F)-T()-P(L)-T()-P(R)-T()-P(L)-T()-P(M)	1	4
P(L)-T()-P(F)-T()-P(L)-T()-P(L)-T()-P(L)	1	4
P(F)-T()-P(F)-T()-P(L)-T()-P(L)-T()-P(L)-T()-P(M)	1	5
P(F)-T()-P(F)-T()-P(L)-T()-P(L)-T()-P(S)-T()-P(M)	1	5
P(L)-T()-P(L)-T()-P(L)-T()-P(R)-T()-P(F)-T()-P(L)	1	5
P(M)-T()-P(L)-T()-P(L)-T()-P(F)-T()-P(L)-T()-P(L)-T()-P(F)	1	6
P(M)-T()-P(L)-T()-P(L)-T()-P(L)-T()-P(R)-T()-P(L)-T()-P(L)	1	6
P(M)-T()-P(R)-T()-P(F)-T()-P(L)-T()-P(F)-T()-P(L)-T()-P(L)-T()-P(F)-T()-P(F)	1	8
P(L)-T()-P(R)-T()-P(R)-T()-P(L)-T()-P(R)-T()-P(L)-T()-P(R)-T()-P(L)-T()-P(L)-T()-P(R)	1	9
P(M)-T()-P(L)-T()-P(S)-T()-P(L)-T()-P(L)-T()-P(L)-T()-P(L)-T()-P(L)-T()-P(L)-T()-P(R)	1	9

Table 4.5 Phase-Label-Only Signatures of Phase Series, their frequency, and their transitions for MABS.

Taxa and signatures

Similarly to the MSL variants, in this case taxa also behave different with respect to the signatures, and in particular, with respect to the number of transitions that signatures contain. Observe Figure 4.1, clearly depicting the different groups of taxa with respect to their signatures.

Taxon / #transitions	0	1	2	>=3	Grand Total
0_FROZEN	34				34
1_ALMOST_FROZEN	51	12	2		65
1_FocusedShot_n_FROZEN	16	6	3		25
2_MODERATE	13	6	2	8	29
3_FocusedShot_n_LOW	9	5	4	2	20
4_ACTIVE	4	5	2	11	22
Grand Total	127	34	13	21	195

Taxon / #transitions	0	1	2	>=3	Grand Total
0_FROZEN	100%				100%
1_ALMOST_FROZEN	78%	18%	3%		100%
1_FocusedShot_n_FROZEN	64%	24%	12%		100%
2_MODERATE	45%	21%	7%	28%	100%
3_FocusedShot_n_LOW	45%	25%	20%	10%	100%
4_ACTIVE	18%	23%	9%	50%	100%
Grand Total	65%	17%	7%	11%	100%

Figure 4.3 Transition ranges and taxa for MABS

The first group contains the family of frozen taxa. As already mentioned, frozen projects contain no transitions by definition. Almost Frozen, Focused Shot and Frozen have mostly one transition and frequently zero.

The second group contains the family of moderately active taxa, which are moderate and Focused Shot and Low which have similar amount of change and similar signatures, mostly having more than 2 transitions. The difference this time is that Focused Shot and Low have more projects of 2 transitions, whereas Moderate have more projects of 3 Transitions (see Figures 4.3 and 4.4).

Finally, the third group includes the Active projects, which typically have longer lives and several phases.

Count of PhaseSeries	Column Labels									
Row Labels	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**8**	**9**	**Grand Total**
0_FROZEN	34									34
1_ALMOST_FROZEN	51	12	2							65
1_FocusedShot_n_FROZEN	16	6	3							25
2_MODERATE	13	6	2	5	2		1			29
3_FocusedShot_n_LOW	9	5	4	1	1					20
4_ACTIVE	4	5	2	4		3	1	1	2	22
Grand Total	**127**	**34**	**13**	**10**	**3**	**3**	**2**	**1**	**2**	**195**

Figure 4.4 Breakdown of projects in taxa and number of transitions for MABS.

4.3 Efficiency Assessment

In this subsection, we discuss the efficiency of the studied algorithms in terms of execution time. The goal is to assess the amount of time needed to execute our algorithms with respect to the length and the history of a project.

4.3.1 Same Label Merge Algorithms

APPENDIX A shows the results of execution time of the Same Labels Merge Algorithm for each project. For each project, we have also included its taxa and duration of months (in other words number of lines in the initial data files). In the sequel, we detail the patterns of behavior that we have observed.

Projects with a single line in the input file. All of the projects that had one line of data, or in other words, lasted less than a month, have the least processing time from all of the other projects. That is because the algorithm never processed that data; since datasets like that contain one month of data, there is just one phase available and there's no series to iterate. Examples like that are on top of the table.

Unexpected time execution behavior. Another pattern that is noticeable is that Frozen and Almost Frozen datasets may have needed more time to be executed than Moderate or Active Projects, even if the duration of all is similar. An example like that is the project "mapbox__mode-mbtiles" (Almost Frozen) and "mozilla__tls-observatory" (Moderate).

mapbox__node-mbtiles	1_ALMOST_FROZEN	32	126220
mozilla__tls-observatory	2_MODERATE	32	40220

Figure 4.5 "mapbox__mode-mbtiles" and "mozilla__tls-observatory" stats.

Both of these projects lasted 32 months (32 initial Phases), however "mapbox__mode-mbtiles" needed almost more than 3 times the time "mozilla__tls-observatory" needed to be executed. That is because the first project needed a lot of merging processes to be done. A lot more Atomic Measurements were needed to be appended in one Phase.

This can also be seen by the graphs in Figures 4.4 and 4.5.

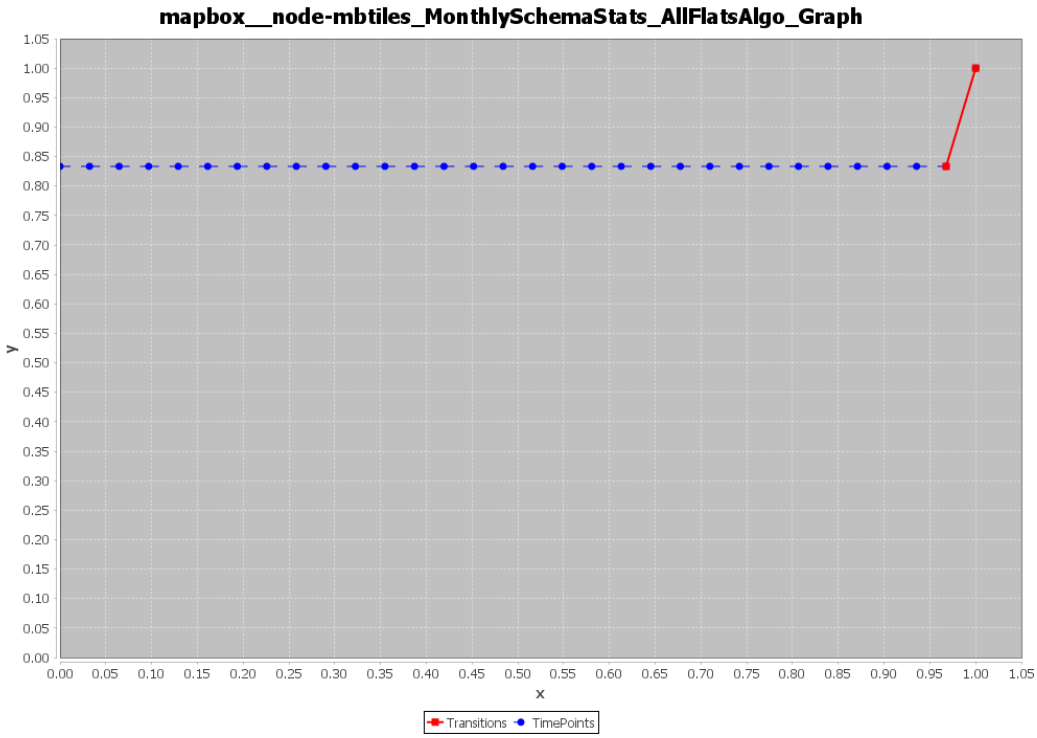

Figure 4.6 Post Processing Same Labels Algorithm Graph for the "mapbox__mode-mbtiles" project

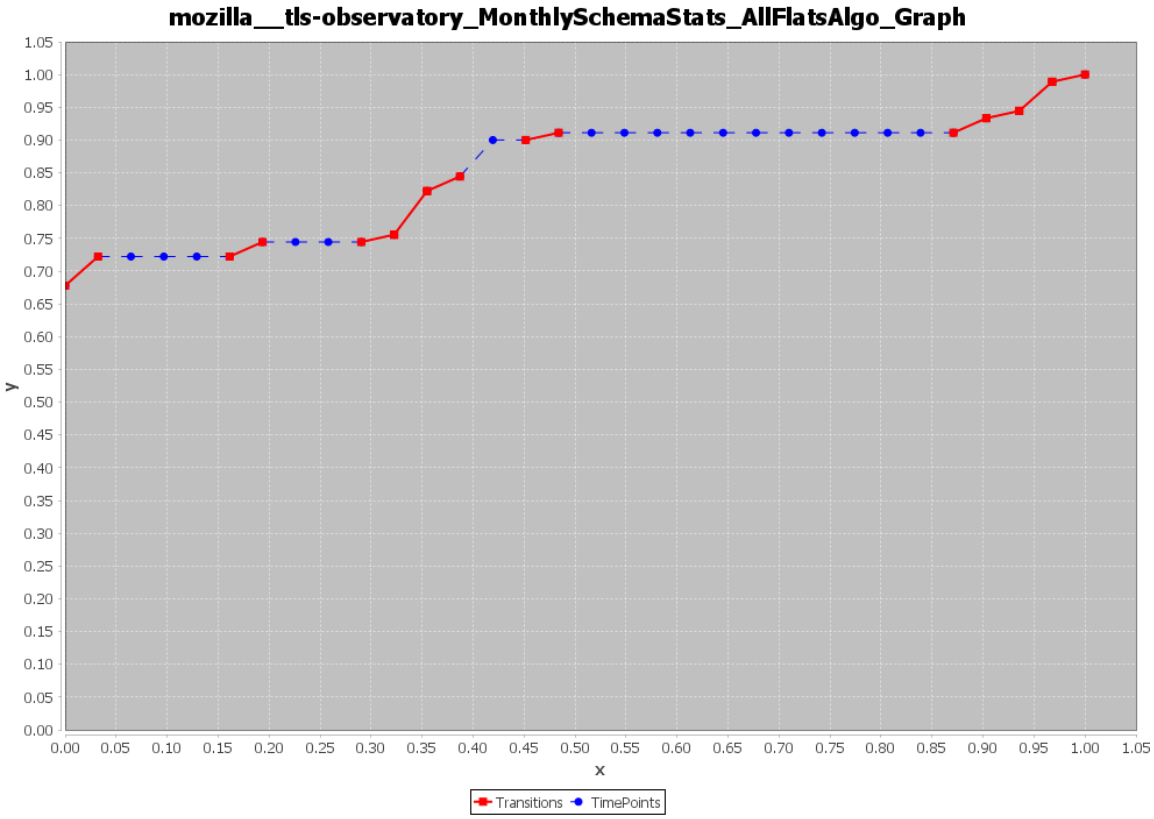

Figure 4.7 Post Processing Same Labels Algorithm Graph for the "mozilla__tls-observatory" project

We observe that, while both projects had a lot of similar Phases that could be merged, the first project keeps merging almost throughout its entire duration. The second project has noticeably less merges. So, the duration of a project is not an absolute indicator of how long the merging process can last, however it does play a small factor; more Phases require more iterations and checks.

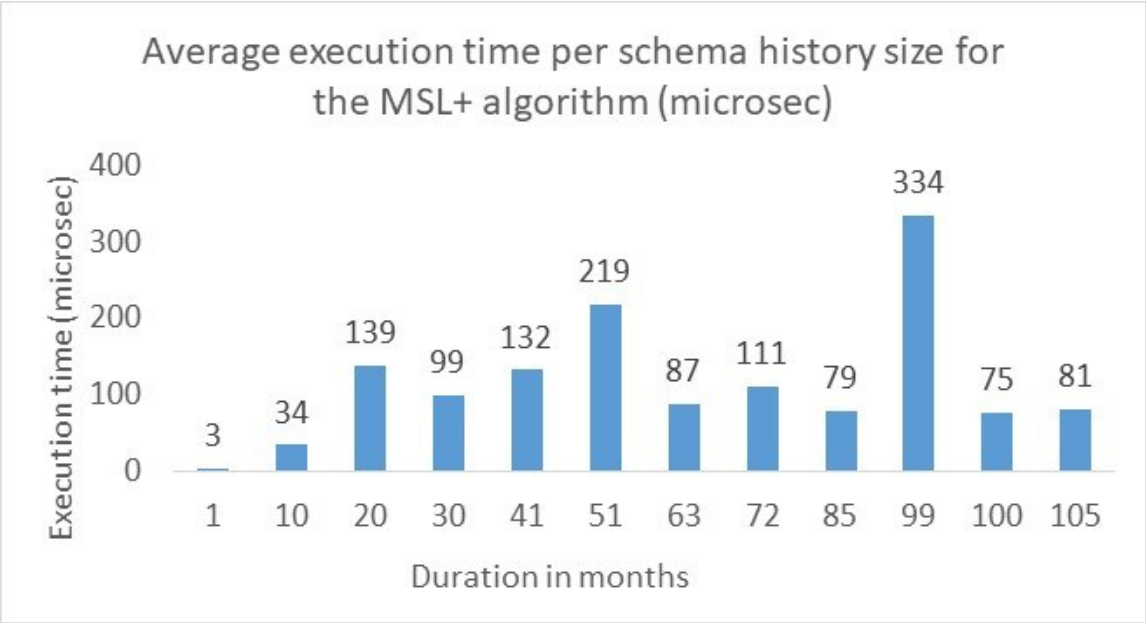

Figure 4.8 Average time (μs) taken to execute algorithm for projects that lasted 1, 10, 20, 30, 41, 51, 63, 72, 85, 99, 100 and 105 months

The effect of schema update period on execution time. In figure 4.8, we observe the behavior of the execution time as a function of the schema update period. In the horizontal axis we depict the number of months the evolution of each schema and on the vertical axis, the execution time in microseconds. We chose schemata whose histories were close to a multiple of 10 months. Each label in the horizontal axis represents all the schemata of the corpus with this duration; the corresponding value in the vertical axis is the average execution time for all the projects that pertain to the respective duration in months.

In the same figure, we observe patterns that further prove the points suggested. First of all, projects that lasted one month have very minimum execution time in average. As already mentioned, the algorithm was never executed for these projects.

Projects of longer duration required extra time. The initial execution time does not depend linearly over the number of Phases; however, the number of merges does require extra execution time. We notice projects of big duration that have small execution time and others that took longer. For example, projects that lasted 99 months had a longer execution time in average, than ones that lasted 85 or 100 months.

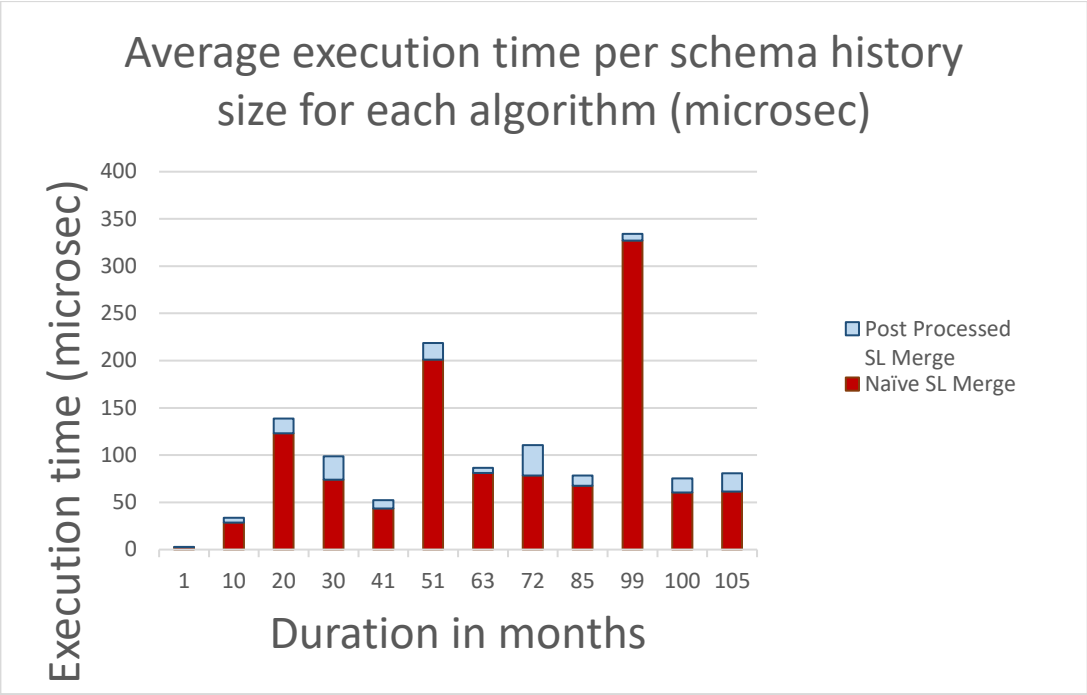

Figure 4.9 Average time (μs) taken to execute each algorithm for projects that lasted 1, 10, 20, 30, 41, 51, 63, 72, 85, 99, 100 and 105 months

In figure 4.9, we observe that the post-processed same label algorithm did not take longer than the naive one. The case of the project that lasted 99 months is interesting, as it appears that the naive algorithm required a lot of execution time, while the post-processed one required only a small fraction of time.

Most of the other projects required some time for the execution of the post-processed algorithm. In all of these cases the execution time is a lot shorter than the naive algorithm's execution time. That is because of the nature of the histories of the schemata and the algorithms. There were a lot of months with no change in activity for a lot of these projects, and this means that those initial Phases needed to be merged into one. That was all processed during the Naive Similar Label Merge Algorithm. The second algorithm, reiterated the new PhaseSeries, only to eliminate any stray Flat Transitions, so, by default, there would not be as many merges to be made.

4.3.2 Merge All but Steep Algorithm

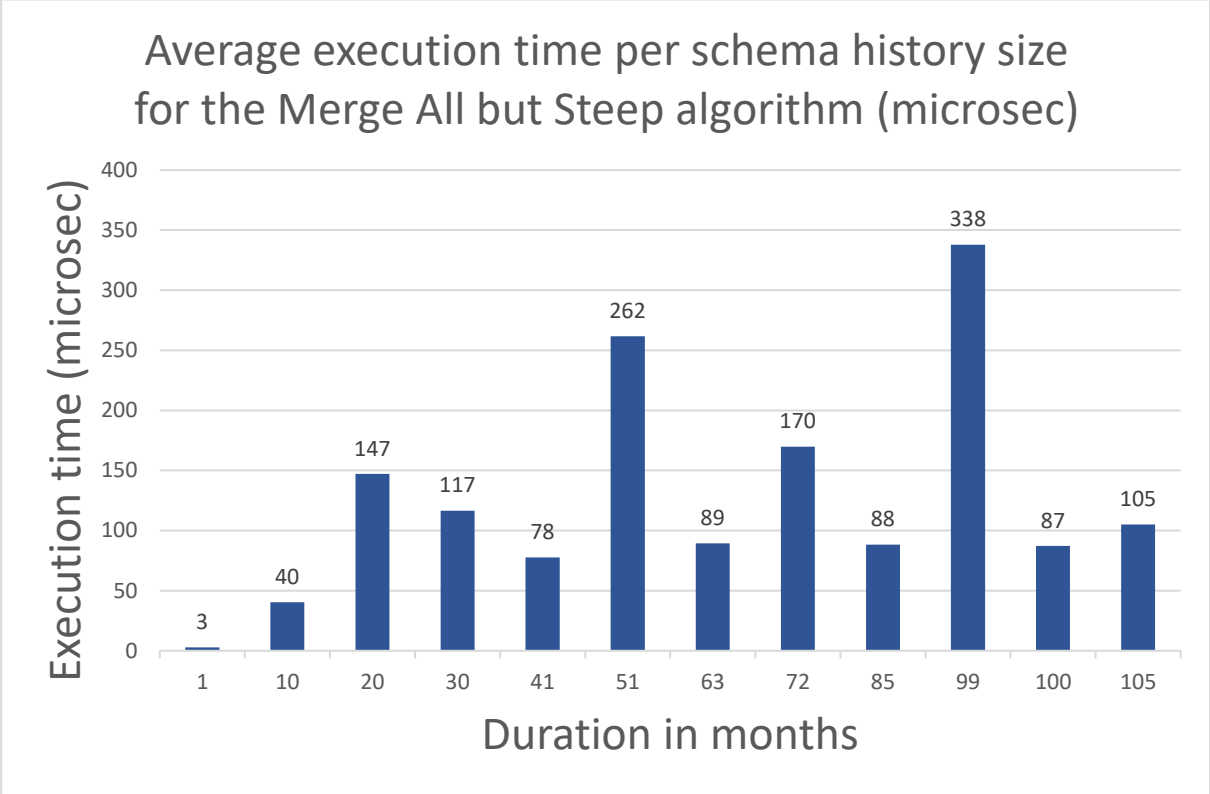

Figure 4.10 Average time (μs) taken to execute the Merge All But Steep Algorithm for projects that lasted 1, 10, 20, 30, 41, 51, 63, 72, 85, 99, 100 and 105 months

In the figure 4.10, we observe the behavior of the execution time as a function of the schema update period. In the horizontal axis we lay the number of months the evolution of each schema and on the vertical axis, the execution time in microseconds. For this algorithm, we also chose schemata whose histories were close to a multiple of 10 months.

In the same figure, we notice that projects that lasted one month have very minimum execution time in average, just like in the previous experiments.

Projects of longer duration required extra time. Similarly, as in figure 4.6, for this algorithm there were projects that lasted 99 months had a longer execution time in average, than ones that lasted 85, 100 or 105 months.

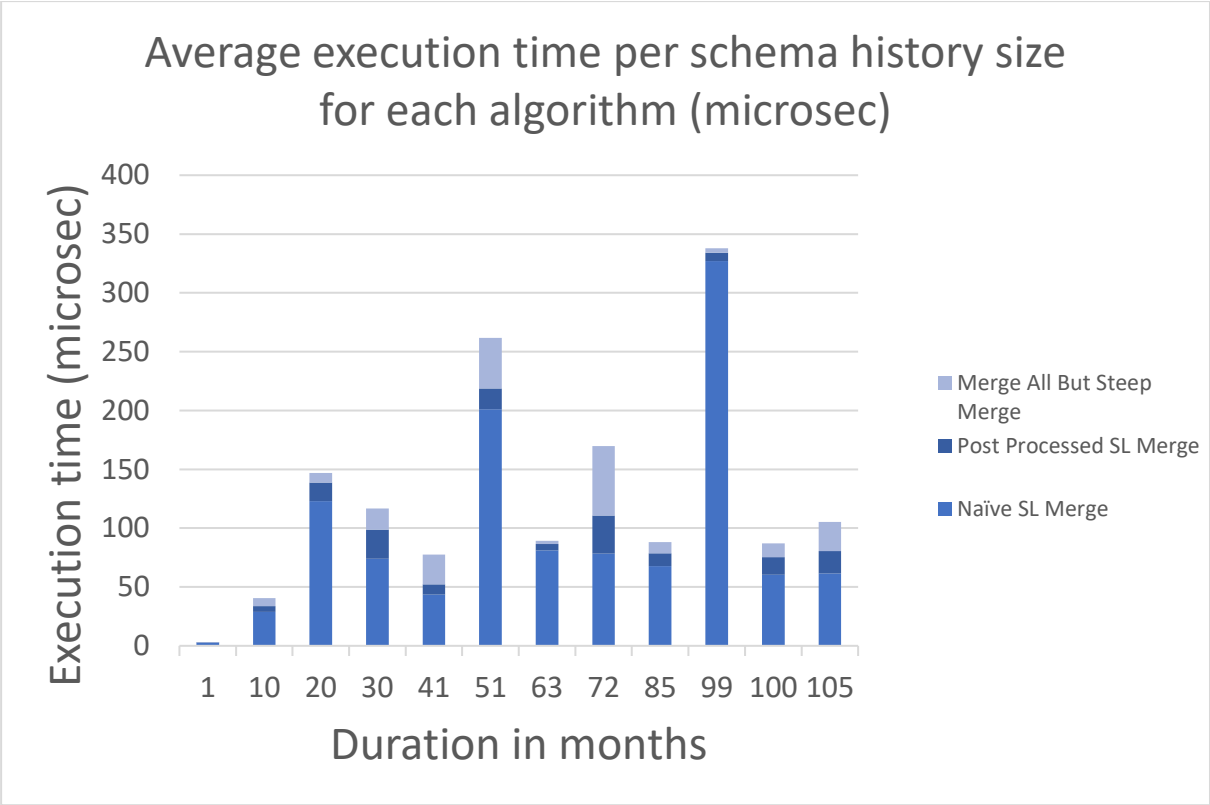

Figure 4.11 Average time (μs) taken to execute each Algorithm for projects that lasted 1, 10, 20, 30, 41, 51, 63, 72, 85, 99, 100 and 105 months

In Figure 4.11, we stack the execution time of each algorithm on top of each other so that we can compare the execution times between the algorithms. In this case we notice that the MABS algorithm requires more execution time than the MSL+ algorithm (Post Processed Same Label Merge). That is a logical result, as the MSL+ algorithm merges only the Flat leftovers of MSL, whereas MABS merges everything else that is not Steep (Low, Regular).

4.4 Correlation between Schema Duration and Merges

In this subsection, we demonstrate the correlation of the number of merges of a signature with the number of lines of the input files – equivalently the duration of the schema update period.

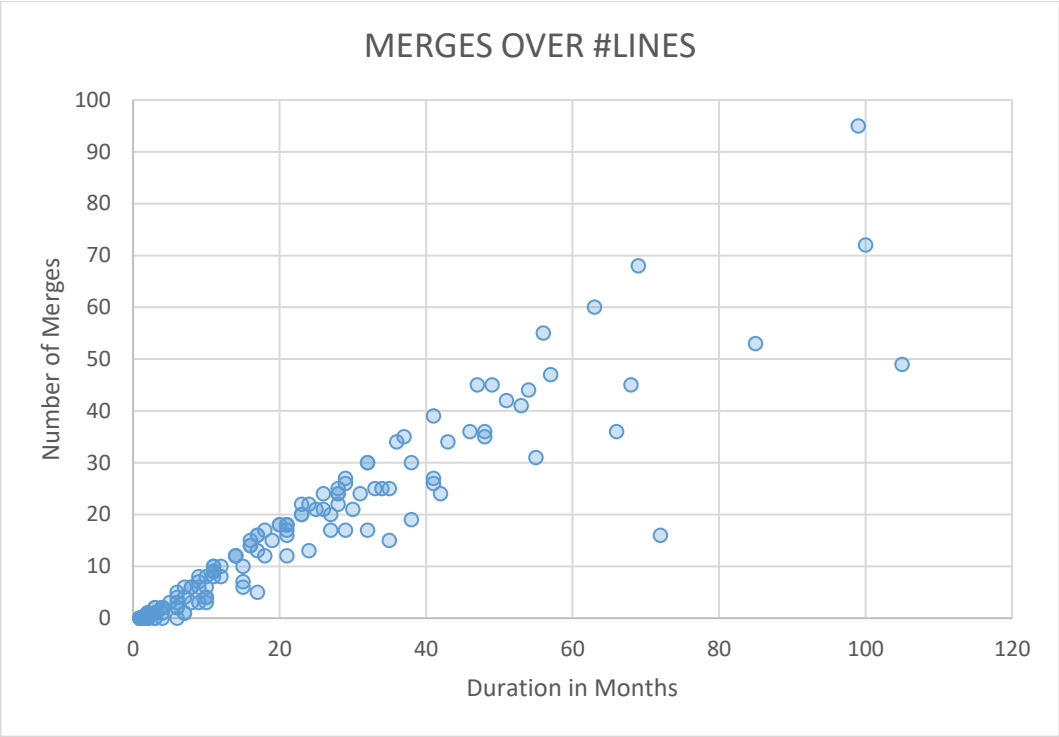

Figure 4.12 Correlation between merges and duration executing MSL.

In Figure 4.12, using the results of MSL, we examine the correlation of (a) the number of lines of the input files (i.e., the duration of the schema update period) and (b) the number of merges. We observe that they are very closely correlated. The Pearson correlation is 94% and their scatterplot is almost a straight line (with few exceptions). Thus, we cannot really discern the effect of merges from the effect of the number of lines in the execution time.

In any case, we isolated a small range of values, where the number of lines is between 63 and 72, having different number of merges and studied the effect on time. The results depicted in Fig. 4.13 are inconclusive.

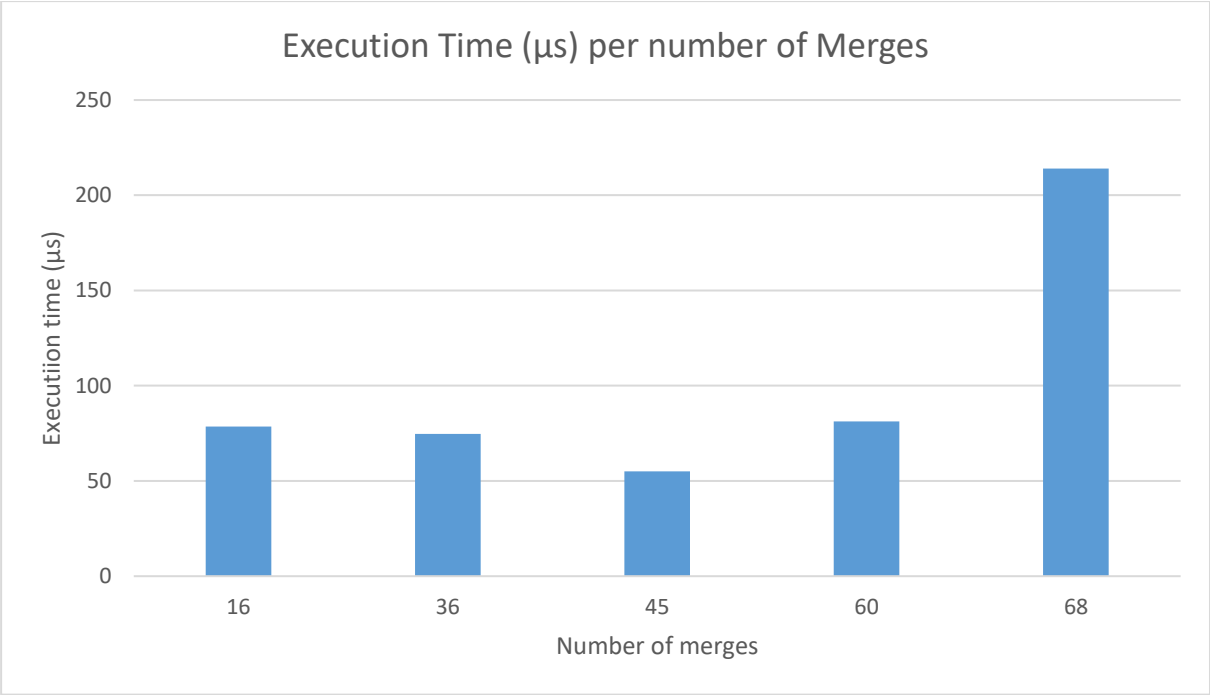

Figure 4.13 Average time (µs) taken per number of merges executing MSL.

Similar results are produced when using the MABS algorithm. In this case the Pearson correlation is 99%.

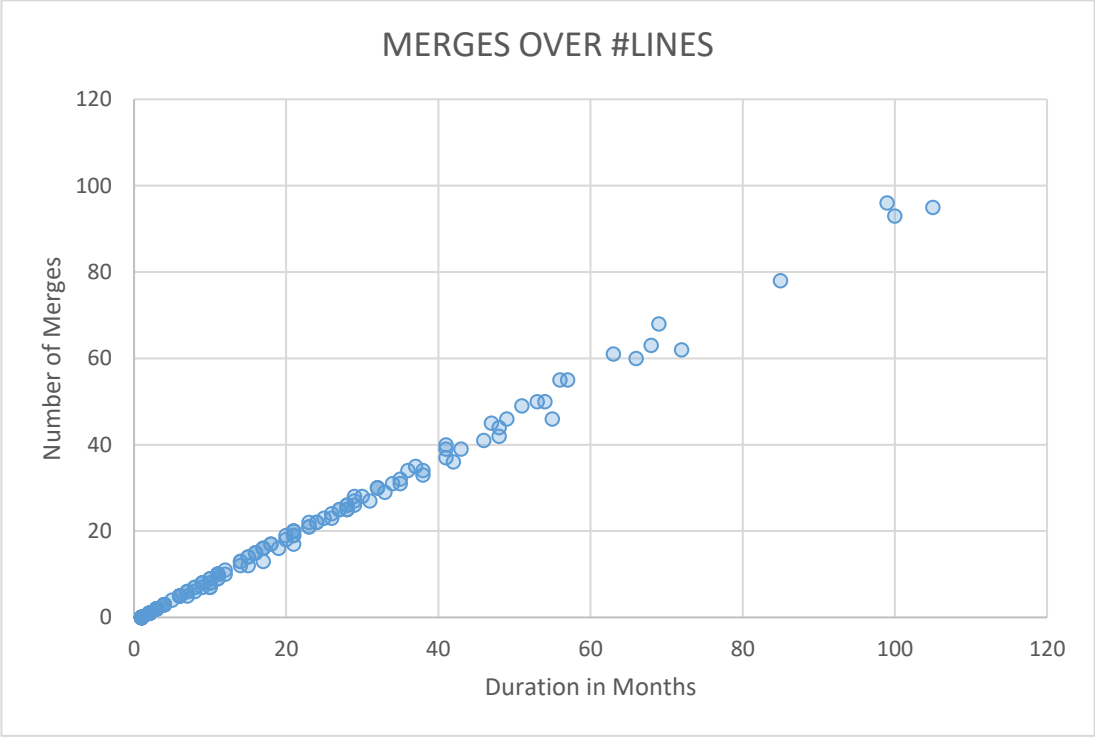

Figure 4.14 Correlation between merges and duration executing MABS.

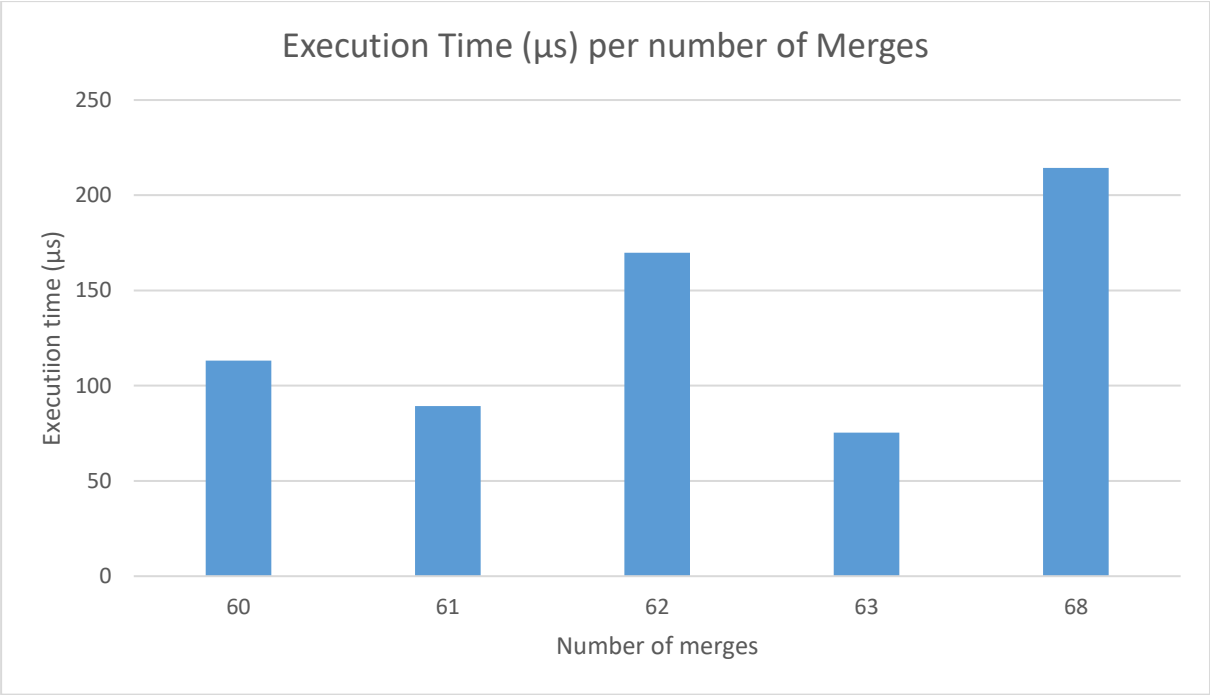

Figure 4.15 Average time (µs) taken per number of merges executing MABS, using the same projects as in figure 4.13.

4.5 State Diagrams

In this subsection, we present the state diagrams that can are derived from our experiments.

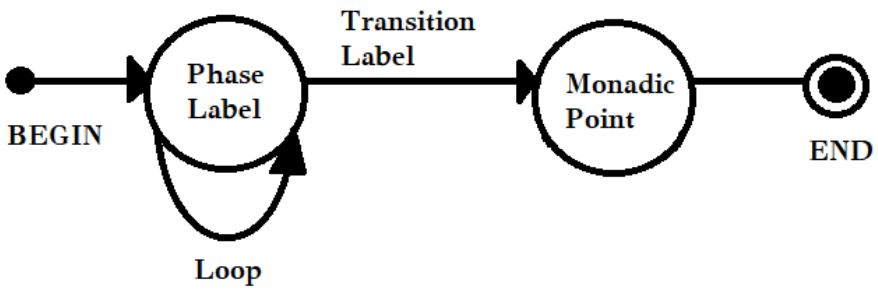

Figure 4.16 A state diagram. The first circle indicates the birth, the big round circles indicate Phases and their loops, the return to the same state. The arrows that connect the Phases are the Transitions. A circle without a loop is a Monadic point. The double circle indicates the end.

In the subsection, of the Efficiency Assessment we derived a few signatures for a good percentage of the total projects:

- MSL+ produced 15 signatures which cover 76% of the (See Table 4.4).
- MABS produced 15 signatures which cover 82.56% of the corpus (See Table 4.5).

In order to portray types of state diagrams that are derived from the aforementioned experiment, we take the top 5 signatures for each algorithm that make up:

- 60% of the projects using MLS+.
- 72.82% of the projects using MABS.

Signature of MLS+	Count of projects	#transitions
P(M)	39	0
P(F)-T()-P(M)	33	1
P(F)	24	0
P(M)-T()-P(M)	15	1
P(F)-T()-P(F)	7	1

Table 4.6 First 5 Phase-Label-Only Signatures of Phase Series, their frequency, and their transitions for MSL+.

The resulting state diagrams for MSL+ are the following in figures 4.16-4.20:

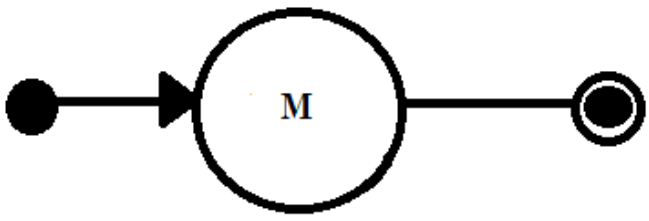

Figure 4.17 MSL+: P(M)

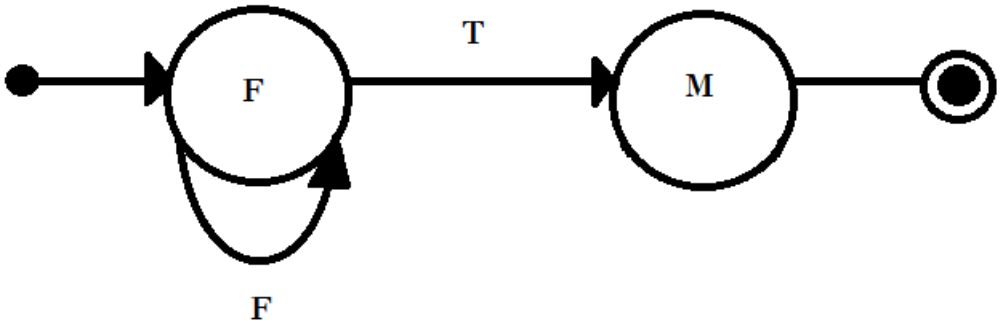

Figure 4.18 MSL+: P(F)-T()-P(M)

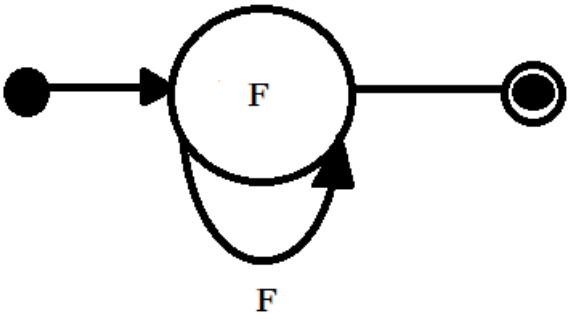

Figure 4.19 MSL+: P(F)

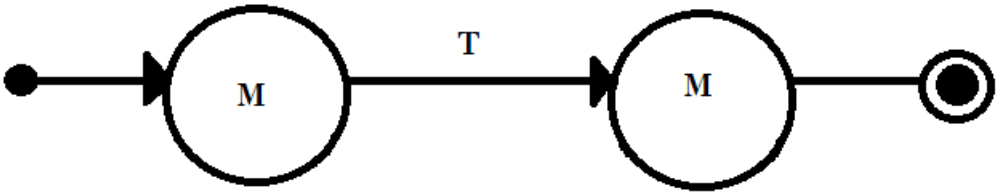

Figure 4.20 MSL+: P(M)-T()-P(M)

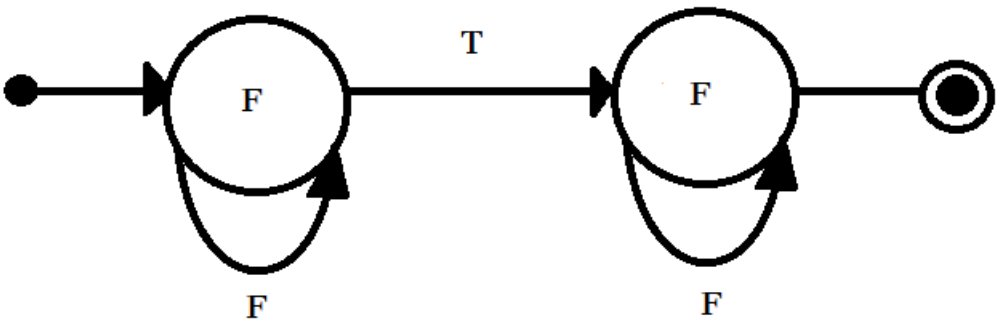

Figure 4.21 MSL+: P(F)-T()-P(F)

Signature	# of Projects	#transitions
P(L)	61	0
P(M)	39	0
P(F)	24	0
P(F)-T()-P(M)	12	1
P(L)-T()-P(L)	6	1

Table 4.7 First 5 Phase-Label-Only Signatures of Phase Series, their frequency, and their transitions for MABS.

The resulting state diagrams for MABS are the following in figures 4.21-4.25:

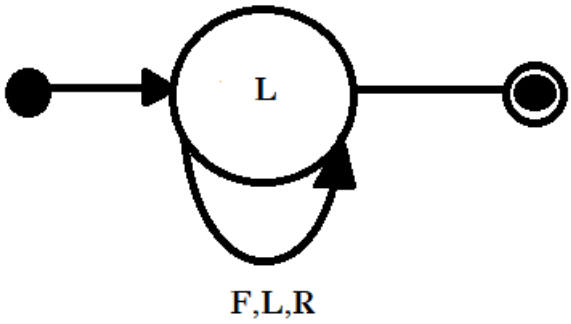

Figure 4.22 MABS: P(L)

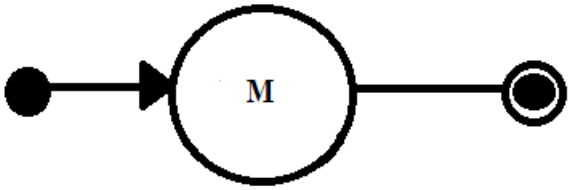

Figure 4.23 MABS: P(M)

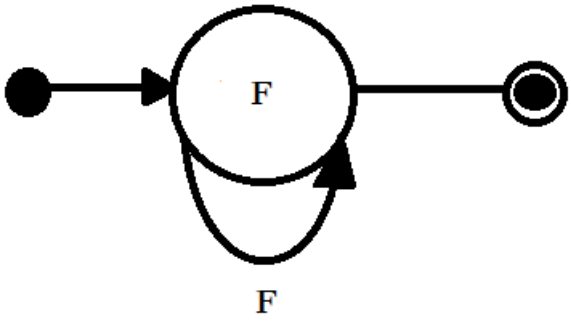

Figure 4.24 MABS: P(F)

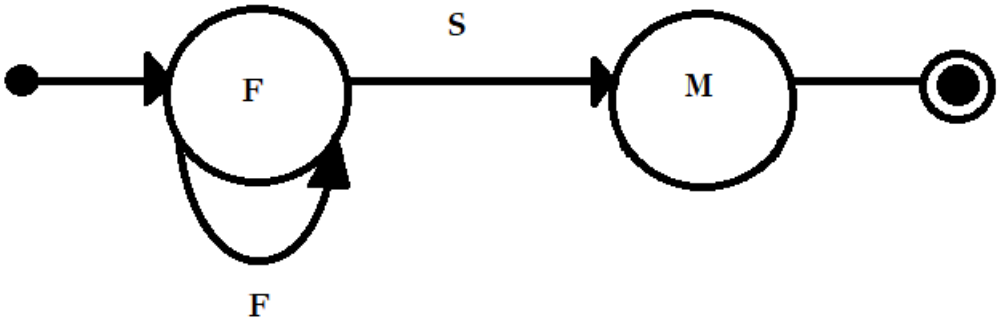

Figure 4.25 MABS: P(F)-T()-P(M)

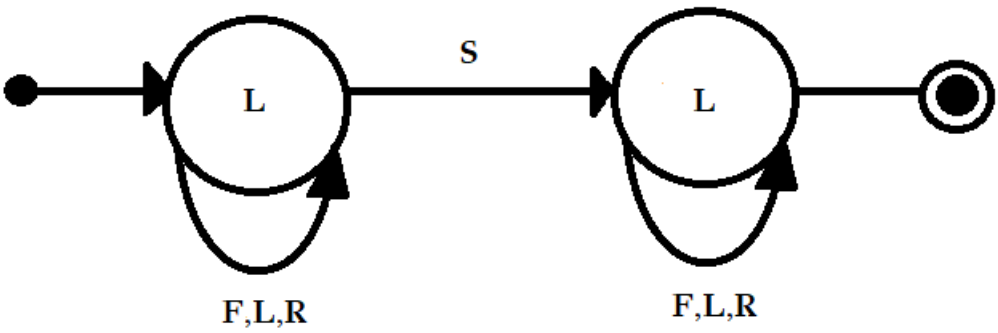

Figure 4.26 MABS: P(L)-T()-P(L)

In figures 4.17-4.26 we notice that the label of the Phase can be Flat, Regular, or Low and we observe that there are no Steep instances. Monadic Points do not include loops as they are a single point. Regarding the loops of the Phases which are marked with (?), the Phases can contain one or more Atomic Measurements, which means that in the state diagram representation, they can return to the same state. In the general case, the label of the Phase that has been produced by merging several other phases into one, can be different, depending on the algorithm.

- For the MSL, the resulting label, is the label of the continuous merged phases.
- For the MSL+, the resulting label is the label of the continuous merged phases, plus Flat Phases.
- For the MABS algorithm, the resulting label could be any label, depending on the project, although it can never be Steep.

Regarding the Transitions and their labels:

- For the MSL, the surviving label can be anything.
- For the MSL+, a transition label will never be Flat.
- For the MABS algorithm, a transition label can only be Steep.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this Section, we discuss the conclusions of the thesis. Given a set of 195 publicly available corpus of schema evolution histories from Free Open-Source Projects, we organized the history of corpus' projects in monthly quanta as time units and assess change via a cumulative metric of monthly change. Given this data we created a PhaseSeries, which consisted of Atomic Measurements, Phases and Transitions. In more detail, Atomic Measurements consist of the cumulative of the time progress percentage (x) and the cumulative evolution activity percentage (y). A Phase includes one or more of these Atomic Measurements. A Transition is the bridge that connects two neighboring Phases which, means that it connects two different sequences of contiguous Atomic Measurements of the schema. The difference between them, can be labeled by the rate of change given the time, which is calculated by the angle that is created between the Phases. So, depending on the degrees of the angle, it can be characterized as:

- FLAT: $\varphi <= 0$ degrees - no change of rate during the transition
- LOW: $0 < \varphi <= 30$ degrees - a small change of rate during the transition
- REGULAR: $30 < \varphi <= 60$ degrees - a normal change of rate during a transition

- STEEP: 60<φ degrees - a significant change of rate during a transition

Our goal was to create shorter PhaseSeries that maintain the most important changes during schema evolution without losing a lot of accuracy, meaning without omitting important information.

To achieve that, we introduced a few algorithms that aimed in merging Transitions, given a similarity metric. The first algorithm we introduced is called "Similar Labels Merge" and its goal is to iterate each pair of neighboring Phases that exist in neighboring Transitions and check if the newer Transition can be merged with the previous one. The Transitions can be merged only if their labels are the same. The outcome of this algorithm is a new PhaseSeries with no repeating neighboring Transitions. The issue that we observed from executing our first algorithm is that many Flat Transitions remained in the PhaseSeries. A Flat Transition implies no change during the evolution, so there is no reason to keep instances like that in the PhaseSeries. That is why we introduced a new algorithm, that basically processes the outcome of the previous algorithm and merges all Flat Transitions and is called Merge Same Labels+. Our final algorithm, which is called Merge All but Steep is merging all the remaining Transitions that are not Steep. In this way we keep the most "important" changes in our PhaseSeries.

In order to portray the resulting PhaseSeries, we introduced the "signatures", i.e., series of symbols that portray the Phases, Transitions and occationally the duration and labels. These help us understand and research the outcomes of the algorithms a lot easier.

After the creation and execution, we ran a couple of experiments to test the efficiency and effectiveness of the algorithms. Regarding the effectiveness we grouped the projects by the different signatures concluded that each algorithm reduces the number of different signatures and the more "active" the evolutions was during a project, the more transitions it has, thus having a higher chance of acquiring a bigger signature. Regarding the efficiency, we noticed that the very first algorithm is the most "resource-heavy" out of the three and that is because a lot of merges have to happen during the initial examination, especially on inactive projects with a lot of duration. Another thing we concluded from these experiments is that the time taken to execute the algorithms is not related to the duration of the projects.

Finally, we extracted examples of state diagrams that derived from this research.

5.2 Future Work

In this thesis we examined 3 algorithms that merge Transitions based on the similarity of labels, or category of labels.

Another approach to this could be to examine the resulting labels of the Phases additionally to the Transitions. The Phases can include one or more Atomic Measurements, so potentially the Phase could be labelled in order to be examined during the merging process. Another merging condition that could be added could be the examination of a duration of a Phase in comparison with the project duration or given a set expected percentage of change rate.

Another idea would be to completely omit the comparison between Transitions and only compare the Phases. The Transitions could be created as the outcome of an algorithm like that.

Finally, another approach could be a bit more complex method, which could require the computation of the cost that is created during the merging process. While keeping any of the examined or discussed merging conditions intact, this new condition could potentially help find the balance between having a lot of Transitions, or in other words a lot of accuracy and losing some accuracy by omitting some Transitions.

As for the creation of the state diagrams, a lot more details could be added, and deeper research could be conducted, in order to extract various categories.

REFERENCES

[1] Belady, L.A., Lehman, M.M.: A model of large program development. IBM Systems Journal 15(3), 225–252, 1976.

[2] Lehman, M.M., Ramil, J.F., Wernick, P., Perry, D.E., Turski, W.M.: Metrics and laws of software evolution - the nineties view. In: 4th IEEE International Software Metrics Symposium (METRICS 1997), p. 20, 1997.

[3] Lehman, M.M., Fernandez-Ramil, J.C.: Rules and Tools for Software Evolution Planning and Management. In: Software Evolution and Feedback: Theory and Practice. John Wiley and Sons Ltd. (2006) ISBN-13: 978-0-470-87180-5.

[4] Dag Sjøberg. Quantifying Schema Evolution. Information and Software Technology, 35(1), pp. 35-44, January 1993

[5] Zhenchang Xing and Eleni Stroulia. Analyzing the Evolutionary History of the Logical Design of Object-Oriented Software, IEEE Transactions on Software Engineering, Vol 13, No 10, October 2005

[6] Marios Fokaefs, Rimon Mikhaiel, Nikolaos Tsantalis, Eleni Stroulia and Alex Lau. An Empirical Study on Web Service Evolution. IEEE International Conference on Web Services, (ICWS) 2011, Washington DC, USA, July 4-9, 2011, 2011

[7] Dong Qiu, Bixin Li, Zhendong Su. An Empirical Analysis of the Co-evolution of Schema and Code in Database Applications. ESEC/FSE '13, 2013.

[8] Hamed Shariat Yazdi, Mahnaz Mirbolouki, Pit Pietsch, Timo Kehrer and Udo Kelter. Analysis and Prediction of Design Model Evolution Using Time Series. CAiSE 20124 WorkShops, LNBIP 178, pp. 1-15, 2014.

[9] Skoulis, I., Vassiliadis, P., Zarras, A.: Open-source databases: within, outside, or beyond Lehman's laws of software evolution? In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 379–393. Springer, Heidelberg, 2014.

[10] Panos Vassiliadis, Apostolos Zarras, Ioannis Skoulis. How is Life for a Table in an Evolving Relational Schema? Birth, Death & Everything in Between. 34th International Conference on Conceptual Modeling (ER 2015). 19-22 October 2015, Stockholm, Sweden.

[11] Panos Vassiliadis , Apostolos V. Zarras , Ioannis Skoulis. Gravitating to Rigidity: Patterns of Schema Evolution -and its Absence- in the Lives of Tables. Information Systems, Volume 63, January 2017, Pages 24-46, ISSN 0306-4379, doi:10.1016/j.is.2016.06.010.

[12] Wenjie Hu, Yang Yang, Ziqiang Cheng, Carl Yang, Xiang Ren. Time-Series Event Prediction with Evolutionary State Graph. Arxiv, available at https://arxiv.org/abs/1905.05006, doi: 10.48550/ARXIV.1905.05006, Last update: 2020.

[13] Panos Vassiliadis. Profiles of Schema Evolution in Free Open Source Software Projects. 37th IEEE International Conference on Data Engineering (ICDE '21), Chania, Crete, Greece, 19-22 April 2021.

[14] Schema Biographies: https://www.cs.uoi.gr/~pvassil/projects/schemaBiographies/info.html

APPENDIX A

DURATION OF SAME LABEL MERGE+ FOR EACH ICDE 2021 DATABASE

Project	TAXON	#months	Sum of Duration (ns)
aiyi__go-user	2_MODERATE	1	6700
APTrust__exchange	1_ALMOST_FROZEN	1	4840
azzlack__Sentinel.OAuth	0_FROZEN	1	4820
colbygk__ARS	1_ALMOST_FROZEN	1	7400
dneustadt__majima	1_ALMOST_FROZEN	1	4300
eldersantos__winston-postgre	1_ALMOST_FROZEN	1	6440
EricDepagne__Astrodb	1_FocusedShot_n_FROZEN	1	5680
fastpress__fastpress	1_FocusedShot_n_FROZEN	1	5160
goproj__note	0_FROZEN	1	7660
ichthus-soft__bible-api	0_FROZEN	1	3700
jasdel__harvester	3_FocusedShot_n_LOW	1	3200
jessemillar__stalks	1_ALMOST_FROZEN	1	3800
jingweno__jqplay	1_ALMOST_FROZEN	1	5300
jmcneese__bitmasked	0_FROZEN	1	3200
knightliao__disconf	0_FROZEN	1	3520
leapp-to__prototype	1_ALMOST_FROZEN	1	6880
mbilbille__jpnforphp	0_FROZEN	1	1100
mgilangjanuar__slimedoo	1_ALMOST_FROZEN	1	960
mozilla__ichnaea	0_FROZEN	1	1120
protosam__hostcontrol	0_FROZEN	1	1000
purefn__hipbot	1_ALMOST_FROZEN	1	1000
remind101__empire	1_ALMOST_FROZEN	1	980

RichMercer__ContentMetadata	0_FROZEN	1	980
rill-event-sourcing__rill	0_FROZEN	1	920
rogeriopvl__nodo	0_FROZEN	1	1020
royzhao__prot-coderun	1_FocusedShot_n_FROZEN	1	1100
rvadym__languages	1_ALMOST_FROZEN	1	960
saltzm__yadi	0_FROZEN	1	1060
shiftcurrency__shift	1_ALMOST_FROZEN	1	1140
skarllot__netpaper	1_ALMOST_FROZEN	1	1240
starbs__yeh	0_FROZEN	1	1140
taskrabbit__empujar	0_FROZEN	1	920
theskyinflames__bpulse-go-client	0_FROZEN	1	960
tracer__tracer	1_ALMOST_FROZEN	1	1120
travis-ci__jupiter-brain	1_ALMOST_FROZEN	1	1280
UlricQin__beego-blog	1_ALMOST_FROZEN	1	980
wanlitao__HangfireExtension	1_ALMOST_FROZEN	1	1100
webinverters__win-with-logs	1_ALMOST_FROZEN	1	920
webnuts__post_json	1_FocusedShot_n_FROZEN	1	980
ankitjain28may__registration-module	1_ALMOST_FROZEN	2	63060
archan937__cached_record	1_FocusedShot_n_FROZEN	2	92380
curt-labs__GoSurvey	3_FocusedShot_n_LOW	2	43420
flynn__flynn-subdomainer	1_ALMOST_FROZEN	2	38460
HXLStandard__hxl-proxy	0_FROZEN	2	48920
jadekler__git-go-d3-concertsap	1_FocusedShot_n_FROZEN	2	37080
jaybennett89__thorium-go	2_MODERATE	2	32760
jgauffin__griffin.mvccontrib	0_FROZEN	2	34400
joyplus__o2oadmin	3_FocusedShot_n_LOW	2	39380
JRonak__OnlineJudge	1_FocusedShot_n_FROZEN	2	41280
liujianping__scaffold	1_FocusedShot_n_FROZEN	2	37220
magnus-lycka__gocddash	1_FocusedShot_n_FROZEN	2	35160
marmelab__comfygure	0_FROZEN	2	38220
mattinsler__work-it	1_ALMOST_FROZEN	2	35820
milogert__ocdns	3_FocusedShot_n_LOW	2	33700
mozilla-services__autograph	1_ALMOST_FROZEN	2	28420

mukatee__pypro	1_FocusedShot_n_FROZEN	2	22460
NPRA__EmissionCalculatorLib	1_FocusedShot_n_FROZEN	2	20780
schimmy__shorty	1_ALMOST_FROZEN	2	22660
spaceboats__busbus	3_FocusedShot_n_LOW	2	20640
teresko__palladium	1_ALMOST_FROZEN	2	20860
Terry-Mao__gopush-cluster	1_FocusedShot_n_FROZEN	2	19840
thesues__catkeeper	1_ALMOST_FROZEN	2	22020
voxpelli__node-connect-pg-simple	0_FROZEN	2	21880
zphalcon__phalcon-tip	0_FROZEN	2	21900
devture__silex-user-bundle	1_FocusedShot_n_FROZEN	3	46020
EPICPaaS__appmsgsrv	4_ACTIVE	3	37780
georgringer__logging	1_ALMOST_FROZEN	3	45460
h2oai__steam	3_FocusedShot_n_LOW	3	42120
keybase__node-client	3_FocusedShot_n_LOW	3	41280
leighmacdonald__php_rbac	0_FROZEN	3	36780
marssa__footprint	0_FROZEN	3	45540
soapboxsys__ombudslib	2_MODERATE	3	22020
williamespindola__field	1_FocusedShot_n_FROZEN	3	26940
yiier__forum	1_ALMOST_FROZEN	3	26620
ZachBergh__spark-mysql-protocol	2_MODERATE	3	22540
Attendly__maillist	2_MODERATE	4	93000
byteball__byteballcore	2_MODERATE	4	46560
MorpheusXAUT__eveauth	2_MODERATE	4	51380
ranaroussi__qtpylib	2_MODERATE	4	26480
scherersoftware__cake-wiki	1_ALMOST_FROZEN	4	32420
seatgeek__djjob	1_ALMOST_FROZEN	4	27660
wskm__deruv	2_MODERATE	4	27840
senecajs__seneca-postgres-store	1_ALMOST_FROZEN	5	31460
DevMine__repotool	1_ALMOST_FROZEN	6	50160
dotkernel__frontend	1_ALMOST_FROZEN	6	49660
lamBc__abc	2_MODERATE	6	43440
magikcypress__slim-boot-boilerplate	0_FROZEN	6	48440
SalesforceEng__cucumber-metrics	1_ALMOST_FROZEN	6	25660
snakerflow__snakerflow	1_FocusedShot_n_FROZEN	6	31280

CityGrid__twonicorn	3_FocusedShot_n_LOW	7	69380
damnpoet__yiicart	0_FROZEN	7	75940
HaliteChallenge__Halite-II	4_ACTIVE	7	48120
the42__ogdat	1_ALMOST_FROZEN	7	25780
cartalyst__sentry	2_MODERATE	8	54920
dburry__indexed_search	1_FocusedShot_n_FROZEN	8	66380
sqlectron__sqlectron-core	1_ALMOST_FROZEN	8	51920
comforme__comforme	2_MODERATE	9	58580
enova__prodder	1_ALMOST_FROZEN	9	62820
nats-io__nats-streaming-server	2_MODERATE	9	34700
outbrain__orchestrator	0_FROZEN	9	41820
hurad__hurad	3_FocusedShot_n_LOW	10	43520
neos__flow-development-collection	1_ALMOST_FROZEN	10	41280
nooku__joomla-todo	4_ACTIVE	10	25700
pw-press__web-project	3_FocusedShot_n_LOW	10	28580
thewhitetulip__Tasks	2_MODERATE	10	28800
webadmin87__rzwebsys7	1_FocusedShot_n_FROZEN	10	34220
atomjump__loop-server	1_ALMOST_FROZEN	11	147160
conceptsandtraining__libtree	1_ALMOST_FROZEN	11	93620
duythien__blog	1_FocusedShot_n_FROZEN	11	52220
jalkoby__squasher	0_FROZEN	11	51200
openzipkin__zipkin	1_ALMOST_FROZEN	11	44020
RiotingNerds__sails-hook-audittrail	0_FROZEN	11	42920
SeldonIO__seldon-server	1_ALMOST_FROZEN	11	50100
twitter__zipkin	1_ALMOST_FROZEN	11	44100
AA-ALERT__frbcatdb	4_ACTIVE	12	148360
jaredbeck__paper_trail-sinatra	1_ALMOST_FROZEN	12	60900
blueriver__MuraCMS	1_ALMOST_FROZEN	14	147600
dlds__yii2-mlm	1_FocusedShot_n_FROZEN	14	69140
hugodias__cakegallery	1_ALMOST_FROZEN	14	74380
accgit__acl	1_FocusedShot_n_FROZEN	15	220420
foodcoopshop__foodcoop-shop	4_ACTIVE	15	69720
processone__ejabberd	4_ACTIVE	15	37960
etsy__mixer	1_ALMOST_FROZEN	16	81640

symfony__security-acl	0_FROZEN	16	72640
vzex__dog-tunnel	1_ALMOST_FROZEN	16	55780
builderscon__octav	4_ACTIVE	17	75980
matthewfranglen__post-gres-elasticsearch-fdw	0_FROZEN	17	74100
prooph__pdo-snapshot-store	0_FROZEN	17	63340
RubyMoney__money-rails	1_ALMOST_FROZEN	17	42820
enova__landable	4_ACTIVE	18	65720
portrino__px_hybrid_auth	0_FROZEN	18	68300
jasongrimes__silex-sim-pleuser	1_FocusedShot_n_FRO-ZEN	19	85940
guardian__alerta	1_ALMOST_FROZEN	20	183140
joomlatools__joomla-plat-form-categories	1_ALMOST_FROZEN	20	94340
gem__oq-engine	1_FocusedShot_n_FRO-ZEN	21	58100
jcoppieters__cody	1_ALMOST_FROZEN	21	59820
joomlatools__joomla-plat-form-finder	1_ALMOST_FROZEN	21	48240
lisong__code-push-server	2_MODERATE	21	55080
mapbox__osm-comments-parser	2_MODERATE	21	90440
rolfvreijdenberger__izzum-statemachine	1_ALMOST_FROZEN	21	49500
bgentry__que-go	0_FROZEN	23	172060
josephspurrier__gowebapp	1_ALMOST_FROZEN	23	74520
n2n__page	1_ALMOST_FROZEN	23	61380
3ev__tev_label	1_FocusedShot_n_FRO-ZEN	24	804260
joomlatools__joomla-plat-form	4_ACTIVE	24	66260
quickapps__cms	4_ACTIVE	25	43500
lamassu__lamassu-scripts	3_FocusedShot_n_LOW	26	60840
mem__padron	1_ALMOST_FROZEN	26	123400
n2n__rocket	3_FocusedShot_n_LOW	27	51020
TalkingData__OWL-v3	3_FocusedShot_n_LOW	27	42580
brettkromkamp__topic_db	3_FocusedShot_n_LOW	28	90000
joomlatools__joomla-plat-form-content	1_ALMOST_FROZEN	28	92500
lamassu__lamassu-admin	2_MODERATE	28	80040
umpirsky__tld-list	1_ALMOST_FROZEN	28	78420
mozilla-services__go-bouncer	1_ALMOST_FROZEN	29	78260
neocogent__sqlchain	2_MODERATE	29	95700

pinterest__teletraan	4_ACTIVE	29	39400
gousiosg__github-mirror	2_MODERATE	30	98680
scorelab__Bassa	2_MODERATE	31	49820
chill117__express-mysql-session	1_ALMOST_FROZEN	32	158920
mapbox__node-mbtiles	1_ALMOST_FROZEN	32	126220
mozilla__tls-observatory	2_MODERATE	32	40220
imsamurai__cakephp-task-plugin	2_MODERATE	33	87100
kronusme__dota2-api	3_FocusedShot_n_LOW	34	63960
arnoldasgudas__Hang-fire.MySqlStorage	4_ACTIVE	35	143280
studygolang__studygolang	4_ACTIVE	35	56060
ironsmile__httpms	1_ALMOST_FROZEN	36	184620
spring-projects__spring-social	1_ALMOST_FROZEN	37	119780
energine-cmf__energine	4_ACTIVE	38	77060
TwitchScience__rs_ingester	3_FocusedShot_n_LOW	38	71920
BotBotMe__botbot-bot	1_FocusedShot_n_FROZEN	41	291840
mozilla__mig	2_MODERATE	41	53180
teaminmedias-pluswerk__ke_search	2_MODERATE	41	51100
MDSLab__s4t-iotronic-standalone	4_ACTIVE	42	102960
gugoan__economizzer	3_FocusedShot_n_LOW	43	76480
imbo__imbo	2_MODERATE	46	86100
tstack__lnav	1_FocusedShot_n_FROZEN	47	127420
blabla1337__skf-flask	4_ACTIVE	48	147160
tronsha__cerberus	4_ACTIVE	48	67240
benoitletondor__TwitterBot	2_MODERATE	49	120940
anchorcms__anchor-cms	3_FocusedShot_n_LOW	51	218780
GoBelieveIO__im_service	3_FocusedShot_n_LOW	53	95820
aimeos__aimeos-typo3	2_MODERATE	54	446100
pods-framework__pods	4_ACTIVE	55	59400
shouldbee__reserved-usernames	0_FROZEN	56	171620
alextselegidis__easyappointments	3_FocusedShot_n_LOW	57	244960
tpolecat__doobie	1_ALMOST_FROZEN	63	86600
intelliants__subrion	4_ACTIVE	66	90080
symphonycms__symphony-2	2_MODERATE	68	64140
shopware__shopware	0_FROZEN	69	214260
cgrates__cgrates	4_ACTIVE	72	110680

torrentpier__torrentpier	4_ACTIVE	85	78560
simplepie__simplepie	1_ALMOST_FROZEN	99	334180
nawork__nawork-uri	2_MODERATE	100	75420
opencart__opencart	4_ACTIVE	105	80600

Table 6.1 Sum of execution time needed for the Same Label Merge+ algorithm in relation to the taxa and months taken for each project

APPENDIX B

DURATION OF MERGE ALL BUT STEEP ALGORITHM FOR EACH ICDE 2021 DATABASE

Project	TAXON	#months	Avg Sum of Duration A0+A1+A2 (ns)
aiyi__go-user	2_MODERATE	1	6980
APTrust__exchange	1_ALMOST_FROZEN	1	5100
azzlack__Sentinel.OAuth	0_FROZEN	1	5020
colbygk__ARS	1_ALMOST_FROZEN	1	7640
dneustadt__majima	1_ALMOST_FROZEN	1	4540
eldersantos__winston-postgre	1_ALMOST_FROZEN	1	6680
EricDepagne__Astrodb	1_FocusedShot_n_FROZEN	1	5920
fastpress__fastpress	1_FocusedShot_n_FROZEN	1	5400
goproj__note	0_FROZEN	1	7900
ichthus-soft__bible-api	0_FROZEN	1	3920
jasdel__harvester	3_FocusedShot_n_LOW	1	3460
jessemillar__stalks	1_ALMOST_FROZEN	1	4020
jingweno__jqplay	1_ALMOST_FROZEN	1	5520
jmcneese__bitmasked	0_FROZEN	1	3560
knightliao__disconf	0_FROZEN	1	3780
leapp-to__prototype	1_ALMOST_FROZEN	1	6940
mbilbille__jpnforphp	0_FROZEN	1	1140
mgilangjanuar__slimedoo	1_ALMOST_FROZEN	1	1000
mozilla__ichnaea	0_FROZEN	1	1160

protosam__hostcontrol	0_FROZEN	1	1080
purefn__hipbot	1_ALMOST_FROZEN	1	1020
remind101__empire	1_ALMOST_FROZEN	1	1060
RichMercer__ContentMetadata	0_FROZEN	1	1060
rill-event-sourcing__rill	0_FROZEN	1	1000
rogeriopvl__nodo	0_FROZEN	1	1060
royzhao__prot-coderun	1_FocusedShot_n_FROZEN	1	1140
rvadym__languages	1_ALMOST_FROZEN	1	1020
saltzm__yadi	0_FROZEN	1	1060
shiftcurrency__shift	1_ALMOST_FROZEN	1	1220
skarllot__netpaper	1_ALMOST_FROZEN	1	1320
starbs__yeh	0_FROZEN	1	1220
taskrabbit__empujar	0_FROZEN	1	960
theskyinflames__bpulse-go-client	0_FROZEN	1	1020
tracer__tracer	1_ALMOST_FROZEN	1	1140
travis-ci__jupiter-brain	1_ALMOST_FROZEN	1	1360
UlricQin__beego-blog	1_ALMOST_FROZEN	1	1020
wanlitao__HangfireExtension	1_ALMOST_FROZEN	1	1160
webinverters__win-with-logs	1_ALMOST_FROZEN	1	940
webnuts__post_json	1_FocusedShot_n_FROZEN	1	1000
ankitjain28may__registration-module	1_ALMOST_FROZEN	2	88120
archan937__cached_record	1_FocusedShot_n_FROZEN	2	119700
curt-labs__GoSurvey	3_FocusedShot_n_LOW	2	79380
flynn__flynn-subdomainer	1_ALMOST_FROZEN	2	48820
HXLStandard__hxl-proxy	0_FROZEN	2	49160
jadekler__git-go-d3-concertsap	1_FocusedShot_n_FROZEN	2	52720
jaybennett89__thorium-go	2_MODERATE	2	43240
jgauffin__griffin.mvccontrib	0_FROZEN	2	34660
joyplus__o2oadmin	3_FocusedShot_n_LOW	2	48540
JRonak__OnlineJudge	1_FocusedShot_n_FROZEN	2	50920
liujianping__scaffold	1_FocusedShot_n_FROZEN	2	46240
magnus-lycka__gocddash	1_FocusedShot_n_FROZEN	2	43600
marmelab__comfygure	0_FROZEN	2	38280

mattinsler__work-it	1_ALMOST_FROZEN	2	35900
milogert__ocdns	3_FocusedShot_n_LOW	2	41960
mozilla-services__autograph	1_ALMOST_FROZEN	2	28480
mukatee__pypro	1_FocusedShot_n_FRO-ZEN	2	22480
NPRA__EmissionCalcula-torLib	1_FocusedShot_n_FRO-ZEN	2	20820
schimmy__shorty	1_ALMOST_FROZEN	2	22720
spaceboats__busbus	3_FocusedShot_n_LOW	2	28300
teresko__palladium	1_ALMOST_FROZEN	2	28280
Terry-Mao__gopush-cluster	1_FocusedShot_n_FRO-ZEN	2	28520
thesues__catkeeper	1_ALMOST_FROZEN	2	28020
voxpelli__node-connect-pg-simple	0_FROZEN	2	21920
zphalcon__phalcon-tip	0_FROZEN	2	21960
devture__silex-user-bundle	1_FocusedShot_n_FRO-ZEN	3	55560
EPICPaaS__appmsgsrv	4_ACTIVE	3	51960
georgringer__logging	1_ALMOST_FROZEN	3	54440
h2oai__steam	3_FocusedShot_n_LOW	3	71280
keybase__node-client	3_FocusedShot_n_LOW	3	49340
leighmacdonald__php_rbac	0_FROZEN	3	36820
marssa__footprint	0_FROZEN	3	45560
soapboxsys__ombudslib	2_MODERATE	3	26920
williamespindola__field	1_FocusedShot_n_FRO-ZEN	3	31260
yiier__forum	1_ALMOST_FROZEN	3	31340
ZachBergh__spark-mysql-protocol	2_MODERATE	3	30620
Attendly__maillist	2_MODERATE	4	114720
byteball__byteballcore	2_MODERATE	4	71300
MorpheusXAUT__eveauth	2_MODERATE	4	57820
ranaroussi__qtpylib	2_MODERATE	4	36380
scherersoftware__cake-wiki	1_ALMOST_FROZEN	4	42180
seatgeek__djjob	1_ALMOST_FROZEN	4	34060
wskm__deruv	2_MODERATE	4	32880
senecajs__seneca-postgres-store	1_ALMOST_FROZEN	5	56280
DevMine__repotool	1_ALMOST_FROZEN	6	59500
dotkernel__frontend	1_ALMOST_FROZEN	6	60320
IamBc__abc	2_MODERATE	6	62640
magikcypress__slim-boot-boilerplate	0_FROZEN	6	48480

SalesforceEng__cucumber-metrics	1_ALMOST_FROZEN	6	33240
snakerflow__snakerflow	1_FocusedShot_n_FROZEN	6	38540
CityGrid__twonicorn	3_FocusedShot_n_LOW	7	120880
damnpoet__yiicart	0_FROZEN	7	76180
HaliteChallenge__Halite-II	4_ACTIVE	7	55420
the42__ogdat	1_ALMOST_FROZEN	7	30820
cartalyst__sentry	2_MODERATE	8	85720
dburry__indexed_search	1_FocusedShot_n_FROZEN	8	79000
sqlectron__sqlectron-core	1_ALMOST_FROZEN	8	58380
comforme__comforme	2_MODERATE	9	71800
enova__prodder	1_ALMOST_FROZEN	9	70440
nats-io__nats-streaming-server	2_MODERATE	9	46640
outbrain__orchestrator	0_FROZEN	9	41880
hurad__hurad	3_FocusedShot_n_LOW	10	51420
neos__flow-development-collection	1_ALMOST_FROZEN	10	47920
nooku__joomla-todo	4_ACTIVE	10	32700
pw-press__web-project	3_FocusedShot_n_LOW	10	39280
thewhitetulip__Tasks	2_MODERATE	10	34600
webadmin87__rzwebsys7	1_FocusedShot_n_FROZEN	10	36780
atomjump__loop-server	1_ALMOST_FROZEN	11	167540
conceptsandtraining__libtree	1_ALMOST_FROZEN	11	109620
duythien__blog	1_FocusedShot_n_FROZEN	11	57700
jalkoby__squasher	0_FROZEN	11	51440
openzipkin__zipkin	1_ALMOST_FROZEN	11	49900
RiotingNerds__sails-hook-audittrail	0_FROZEN	11	42980
SeldonIO__seldon-server	1_ALMOST_FROZEN	11	56700
twitter__zipkin	1_ALMOST_FROZEN	11	49080
AA-ALERT__frbcatdb	4_ACTIVE	12	183840
jaredbeck__paper_trail-sinatra	1_ALMOST_FROZEN	12	69240
blueriver__MuraCMS	1_ALMOST_FROZEN	14	163360
dlds__yii2-mlm	1_FocusedShot_n_FROZEN	14	74240
hugodias__cakegallery	1_ALMOST_FROZEN	14	83000
accgit__acl	1_FocusedShot_n_FROZEN	15	261160

foodcoopshop__foodcoop-shop	4_ACTIVE	15	100760
processone__ejabberd	4_ACTIVE	15	53340
etsy__mixer	1_ALMOST_FROZEN	16	92720
symfony__security-acl	0_FROZEN	16	72720
vzex__dog-tunnel	1_ALMOST_FROZEN	16	61560
builderscon__octav	4_ACTIVE	17	101060
matthewfranglen__post-gres-elasticsearch-fdw	0_FROZEN	17	74160
prooph__pdo-snapshot-store	0_FROZEN	17	63380
RubyMoney__money-rails	1_ALMOST_FROZEN	17	54780
enova__landable	4_ACTIVE	18	83440
portrino__px_hybrid_auth	0_FROZEN	18	68380
jasongrimes__silex-sim-pleuser	1_FocusedShot_n_FRO-ZEN	19	91460
guardian__alerta	1_ALMOST_FROZEN	20	192840
joomlatools__joomla-plat-form-categories	1_ALMOST_FROZEN	20	101260
gem__oq-engine	1_FocusedShot_n_FRO-ZEN	21	63660
jcoppieters__cody	1_ALMOST_FROZEN	21	67500
joomlatools__joomla-plat-form-finder	1_ALMOST_FROZEN	21	59660
lisong__code-push-server	2_MODERATE	21	65240
mapbox__osm-comments-parser	2_MODERATE	21	97460
rolfvreijdenberger__izzum-statemachine	1_ALMOST_FROZEN	21	55300
bgentry__que-go	0_FROZEN	23	172420
josephspurrier__gowebapp	1_ALMOST_FROZEN	23	79920
n2n__page	1_ALMOST_FROZEN	23	66580
3ev__tev_label	1_FocusedShot_n_FRO-ZEN	24	843760
joomlatools__joomla-plat-form	4_ACTIVE	24	78860
quickapps__cms	4_ACTIVE	25	47920
lamassu__lamassu-scripts	3_FocusedShot_n_LOW	26	65920
mem__padron	1_ALMOST_FROZEN	26	128100
n2n__rocket	3_FocusedShot_n_LOW	27	61980
TalkingData__OWL-v3	3_FocusedShot_n_LOW	27	53840
brettkromkamp__topic_db	3_FocusedShot_n_LOW	28	102020
joomlatools__joomla-plat-form-content	1_ALMOST_FROZEN	28	97720
lamassu__lamassu-admin	2_MODERATE	28	86820

umpirsky__tld-list	1_ALMOST_FROZEN	28	81200
mozilla-services__go-bouncer	1_ALMOST_FROZEN	29	85500
neocogent__sqlchain	2_MODERATE	29	99680
pinterest__teletraan	4_ACTIVE	29	48460
gousiosg__github-mirror	2_MODERATE	30	116600
scorelab__Bassa	2_MODERATE	31	54020
chill117__express-mysql-session	1_ALMOST_FROZEN	32	184480
mapbox__node-mbtiles	1_ALMOST_FROZEN	32	131740
mozilla__tls-observatory	2_MODERATE	32	61760
imsamurai__cakephp-task-plugin	2_MODERATE	33	100000
kronusme__dota2-api	3_FocusedShot_n_LOW	34	74180
arnoldasgudas__Hang-fire.MySqlStorage	4_ACTIVE	35	171780
studygolang__studygolang	4_ACTIVE	35	70620
ironsmile__httpms	1_ALMOST_FROZEN	36	189640
spring-projects__spring-social	1_ALMOST_FROZEN	37	123460
energine-cmf__energine	4_ACTIVE	38	93720
TwitchScience__rs_ingester	3_FocusedShot_n_LOW	38	75480
BotBotMe__botbot-bot	1_FocusedShot_n_FROZEN	41	299140
mozilla__mig	2_MODERATE	41	79100
teaminmedias-pluswerk__ke_search	2_MODERATE	41	76140
MDSLab__s4t-iotronic-standalone	4_ACTIVE	42	118720
gugoan__economizzer	3_FocusedShot_n_LOW	43	86020
imbo__imbo	2_MODERATE	46	94000
tstack__lnav	1_FocusedShot_n_FROZEN	47	129900
blabla1337__skf-flask	4_ACTIVE	48	220880
tronsha__cerberus	4_ACTIVE	48	73980
benoitletondor__TwitterBot	2_MODERATE	49	134640
anchorcms__anchor-cms	3_FocusedShot_n_LOW	51	261720
GoBelieveIO__im_service	3_FocusedShot_n_LOW	53	109280
aimeos__aimeos-typo3	2_MODERATE	54	504680
pods-framework__pods	4_ACTIVE	55	73560
shouldbee__reserved-usernames	0_FROZEN	56	171660
alextselegidis__easyappointments	3_FocusedShot_n_LOW	57	344100
tpolecat__doobie	1_ALMOST_FROZEN	63	89320

intelliants__subrion	4_ACTIVE	66	113180
symphonycms__symphony-2	2_MODERATE	68	75400
shopware__shopware	0_FROZEN	69	214280
cgrates__cgrates	4_ACTIVE	72	169820
torrentpier__torrentpier	4_ACTIVE	85	88200
simplepie__simplepie	1_ALMOST_FROZEN	99	338000
nawork__nawork-uri	2_MODERATE	100	87120
opencart__opencart	4_ACTIVE	105	105220

Table 6.2 Sum of execution time needed for the Merge All but Steep algorithm in relation to the taxa and months taken for each project

SHORT BIOGRAPHICAL SKETCH

Christina Trialoni is a M.Sc. graduate student at the Department of Computer Science and Engineering (CSE) of the University of Ioannina, Greece. She is currently employed as a Software Developer, by Natech S.A., a company which creates software for financial purposes, in Ioannina. Her research interests lie in the area of Databases and Software Engineering.